UNIVERSIDADE FEDERAL DE SANTA CATARINA

Cálculo de Indicadores Financeiros com Auxílio do Processamento de Linguagem Natural

Lucas João Martins

Florianópolis
2021

Lucas João Martins

Cálculo de Indicadores Financeiros com Auxílio do Processamento de Linguagem Natural

Trabalho de conclusão de curso submetido como parte dos requisitos para obtenção do título de Bacharel, do curso de Ciências da Computação da Universidade Federal de Santa Catarina.

Orientador: Elder Rizzon Santos

Florianópolis, Setembro de 2021

Lucas João Martins

Cálculo de Indicadores Financeiros com Auxílio do Processamento de Linguagem Natural

Este Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de Bacharel em Ciências da Computação, e aprovado em sua forma final pelo Curso de Ciências da Computação da Universidade Federal de Santa Catarina.

Prof. Dr. Elder Rizzon Santos
Orientador
Universidade Federal de Santa Catarina

Prof. Dr. Alexandre Gonçalves Silva
Membro da Banca
Universidade Federal de Santa Catarina

Dr. Thiago Angelo Gelaim
Membro da Banca
Universidade Federal de Santa Catarina

Florianópolis, Setembro de 2021

Resumo

É possível dizer que o principal objetivo da inteligência artificial consiste em ajudar as pessoas, principalmente na execução de determinadas tarefas. Além disso, o processamento de linguagem natural pode ser considerado um domínio que abrange diversas áreas, sendo algumas delas pertencentes ao campo da inteligência artificial. Enquanto isso, o mercado financeiro está em constante crescimento nos últimos anos. Quem participa do mercado financeiro sabe que para se obter sucesso, um fator que ajuda bastante é a informação e sua qualidade. Com isso tudo associado, o presente trabalho busca juntar o processamento de linguagem natural ao mercado financeiro com a intenção de ajudar na execução de uma tarefa - saber indicadores financeiros de uma empresa. A associação desses dois campos é feita no trabalho ao propor um modelo que tem como objetivo calcular indicadores financeiros a partir de relatórios empresariais com o auxílio do processamento de linguagem natural.

Palavras-chaves: Inteligência Artificial. Processamento de linguagem natural. Indicadores financeiros.

Abstract

It is possible to say that the main purpose of artificial intelligence is to help people, especially in carrying out certain tasks. Furthermore, natural language processing can be considered a domain that connects several areas, some of which belong to the field of artificial intelligence. Meanwhile, the financial market has been constantly growing in recent years. Anyone who participates in the financial market knows that to be successful, a factor that helps a lot is information and its quality. With all this associated, the present work pursues to join natural language processing to the financial market with the intention of helping in the execution of a task - knowing a company's financial indicators. The association of these two fields is made in the work by proposing a model that aims to calculate financial indicators from business reports with the aid of natural language processing.

Keywords: Artificial intelligence. Natural language processing. Financial indicators.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Definir com precisão o que é inteligência artificial (IA) não é apenas difícil, mas sim praticamente impossível (LORICA; LOUKIDES, 2016). Principalmente pelo fato de que não há um real entendimento do que é a inteligência humana (LORICA; LOUKIDES, 2016). Apesar disso, é apropriado dizer que a inteligência artificial é uma área da ciência da computação preocupada com a automação do comportamento inteligente (LUGER, 2008). A IA possui uma grande variedade de subáreas, além de ser relevante para qualquer tarefa intelectual, por isso é considerada um campo de estudo universal (RUSSELL; NORVIG, 2009).

A ficção científica costuma idealizar situações com grandes evoluções tecnológicas, que de certa forma a IA busca alcançar, como o computador inteligente HAL 9000 apresentado em 2001 - Uma odisseia no espaço de Arthur C. Clarke (MOTE, 2012). Só que no fim, o principal objetivo da inteligência artificial é ajudar as pessoas (AN, 2018). Com o grande progresso da área de *hardware* computacional das últimas décadas, houve um aumento significativo de espaço disponível para armazenamento (LORICA; LOUKIDES, 2016). Uma vez que há mais espaço disponível e também com o advento da internet, a criação de mais dados para preencher esses espaços até então inexistentes é possível (LOUKIDES, 2011). Abordagens dirigidas por dados possuem a habilidade de acelerar o progresso no campo de processamento de linguagem natural, uma vez que pesquisas e aplicações desse setor costumam depender de grandes conjuntos de dados para que seja possível realizar o treinamento e a avaliação, além do que uma grande quantidade de dados de treinamento pode simplificar o modelo final em alguns casos (GUDIVADA *et al.*, 2015).

Processamento de Linguagem Natural (PLN) é um domínio interdisciplinar que abrange áreas como linguística, recebimento de informação, *machine learning*, probabilidade e estatística (GUDIVADA *et al.*, 2015). No geral, o campo de processamento de linguagem natural cria modelos computacionais e processos para resolver problemas práticos que envolvem o entendimento da linguagem humana (OTTER *et al.*, 2020). Historicamente esse domínio pode ser dividido em algumas fases: a primeira abrange o final da década de 40 até aproximadamente o final da década de 60 com foco em *machine translation* (MT); a segunda consiste do final da década de 60 até o final da década de 70 com bastante desenvolvimento em conjunto com o campo de inteligência artificial; já a terceira engloba o final da década de 70 até o final da década de 80 com foco na parte gramatical, e, por fim, a última fase, do fim da década de 80 até o começo do século XXI com foco na parte léxica e no conjunto de dados (JONES, 2001). A partir de então é um segmento praticamente onipresente no mundo, ao considerar sua aplicação em algumas atividades, como mecanismos de buscas (e.g. Google), assistentes pessoais (e.g. Siri) e plataformas de

propaganda (WIJERATNE *et al.*, 2019). Diversas técnicas são utilizadas para alcançar esse patamar, como pode ser visto, por exemplo, na mineração de textos, onde se pode usar algoritmo de Markov, algoritmo de Naive Bayes, ontologia, *conceptual vector model* e diversas outras possibilidades (SANTOS *et al.*, 2015).

Embora às vezes haja uma certa dificuldade nisso, é possível dividir os trabalhos em PLN em duas subáreas: subárea central e subárea de aplicações. A central lida com problemas fundamentais, como por exemplo modelagem linguística, enquanto que a de aplicações trata da solução de problemas práticos (OTTER *et al.*, 2020). Ao seguir essa classificação, o presente trabalho se encaixa na subárea de aplicações, já que busca automatizar o cálculo de indicadores financeiros com o auxílio de PLN. O mundo das finanças mudou drasticamente nas últimas décadas, ao ponto da sua globalização fazer com que trilhões de dólares estejam presentes nos mercados financeiros globais. Há diferentes tipos de mercados financeiros, onde cada um auxilia uma necessidade diferente do mundo econômico (PILBEAM, 2018). Recentemente, diversos pesquisadores têm investigado como prever informações nos mercados financeiros, seja através de técnicas de *machine learning* ou com mineração de dados em textos (como relatórios financeiros ou notícias). Isso é impulsionado devido ao fato de que o sucesso de um investidor nessa área depende da qualidade da informação que ele possui e da velocidade para obtê-la, já que a mesma servirá de ajuda na sua tomada de decisão (CAVALCANTE *et al.*, 2016).

Atualmente, ao mesmo tempo que é uma condição cada vez mais existente no futuro, a interação humana com a informação através de computadores irá exigir uma grande variedade de capacidade de processamento de dados, onde o centro de todas elas será o processamento de linguagem natural (MOTE, 2012). Portanto, o aprendizado na área financeira é facilitado pela existência de uma vasta quantidade de dados desse tipo, mas por outro lado, a grande quantidade de aleatoriedade nas informações pode dificultar o processo de aprendizado (PASTOR; VERONESI, 2009). Enquanto as instituições financeiras atuais e o modo de fazer negócio na área financeira tendem a ficar desatualizados com o passar do tempo, principalmente se considerar a corrente revolução *FinTech* que altera os paradigmas vigentes (PILBEAM, 2018). Os avanços tecnológicos e a disponibilidade cada vez maior de diferentes tipos de dados mostram que o campo de processamento de linguagem natural possui diversas possibilidades de estudo, com uma atestada relevância na academia e na indústria. Já o mercado financeiro está em constante evolução, com mudanças muitas vezes proporcionadas pela disponibilidade de novas tecnologias e que impactam a forma com que empresas e consumidores se relacionam com a área. Sendo assim, este trabalho utilizará técnicas de PLN para calcular indicadores financeiros com base em informações (textos não estruturados) provenientes de relatórios de resultados empresariais, os quais representam a saúde financeira da empresa e por isso ajudam na tomada de decisão no mercado financeiro ao fazer parte de uma análise fundamentalista da instituição.

1.1 Objetivos

1.1.1 Objetivo geral

Calcular indicadores financeiros a partir de relatórios de resultados empresariais com o auxílio de PLN.

1.1.2 Objetivos específicos

- Analisar o estado da arte das técnicas de PLN;
- Definir os requisitos do modelo (ou solução) delimitando o contexto do trabalho;
- Propor um modelo, com base em técnicas de PLN, para auxiliar no cálculo dos indicadores;
- Definir os experimentos que irão testar o modelo;
- Desenvolver o protótipo que executará os experimentos com as técnicas escolhidas;
- Verificar a corretude dos resultados obtidos através de testes e comparações.

2 Fundamentação teórica

Esta seção apresentará os conceitos que embasam o cálculo de indicadores financeiros a partir de técnicas de processamento de linguagem natural, ou seja, o desenvolvimento realizado para alcançar o objetivo do trabalho.

2.1 Processamento de linguagem natural

2.1.1 Histórico

Diferentes autores escreveram sobre o histórico do campo de processamento de linguagem natural. Para isso utilizaram técnicas diferentes para apresentar a história. Por exemplo, Jones (2001), divide a história em quatro diferentes fases. Enquanto que Collier (1994), mostra a história a partir de uma divisão realizada nela que considera quatro diferentes formas de aprendizagem possíveis. Por fim, Russell e Norvig (2009) utilizam duas formas: uma listagem direta dos acontecimentos e uma demonstração dos mesmos com base em uma divisão por tarefas práticas reais.

2.1.2 Fluxos de trabalho possíveis

Como dito por Martínez-Plumed *et al.* (2019), em 1999 a primeira versão do *CRoss-Industry Standard Process for Data Mining*, também conhecido como CRISP-DM foi apresentada ao público. Martínez-Plumed *et al.* (2019) afirmam que isso foi resultado do trabalho de um time de experientes engenheiros de *data mining* financiados pela União Europeia. O artigo dos autores cita que rapidamente o CRISP-DM tornou-se o padrão para desenvolvimento com *data mining* e ainda hoje é um dos métodos analíticos mais utilizados de acordo com diversas pesquisas de opinião. O CRISP-DM pode ser apresentado como o fluxo de trabalho apresentado na figura 1.

Apesar do alto uso, Martínez-Plumed *et al.* (2019) comentam que desde o surgimento do CRISP-DM muito se mudou, principalmente com relação a forma de se coletar e processar dados, logo, adaptações e outros fluxos de trabalho foram propostos para se adaptarem às mudanças. Nessa oportunidade surge o artigo online Sarkar (2018), onde também afirma que o modelo CRISP-DM é tipicamente o padrão na indústria para executar qualquer projeto de *data science*. Além disso, Sarkar (2018) generaliza e garante que qualquer problema baseado em PLN pode ser resolvido por uma metodologia de fluxo de trabalho que possui uma sequência de etapas. Sarkar (2018) estabelece que os passos desse fluxo de trabalho correspondem aos passos da figura 2.

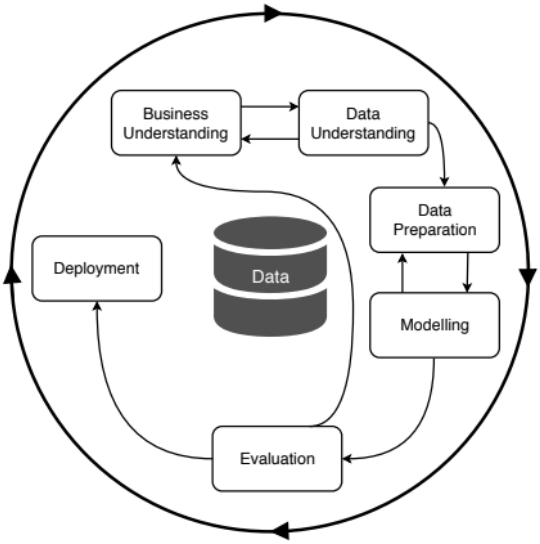

Figura 1 – Fluxo de trabalho do CRISP-DM por Martínez-Plumed *et al.* (2019)

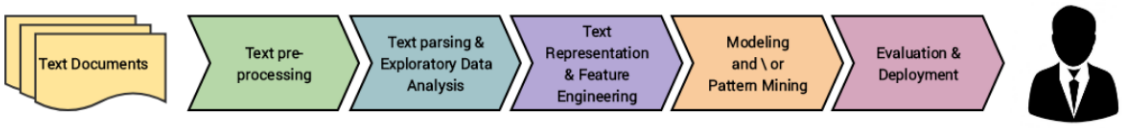

Figura 2 – Fluxo de trabalho proposto por Sarkar (2018)

Já na parte de extração de informação, Xue *et al.* (2011) apresentam uma simples arquitetura de *pipeline* para um sistema de extração de informação. O sistema dessa arquitetura teria como entrada um texto cru de algum documento e geraria uma lista de tuplas como resultado. Por exemplo, ao considerar um documento que indica que uma empresa chamada Georgia-Pacific encontra-se em Atlanta, o sistema geraria a seguinte tupla como resultado: `([ORG: 'Georgia-Pacific'] 'em' [LOC: 'Atlanta'])`. O *pipeline* pode ser visto na imagem 3.

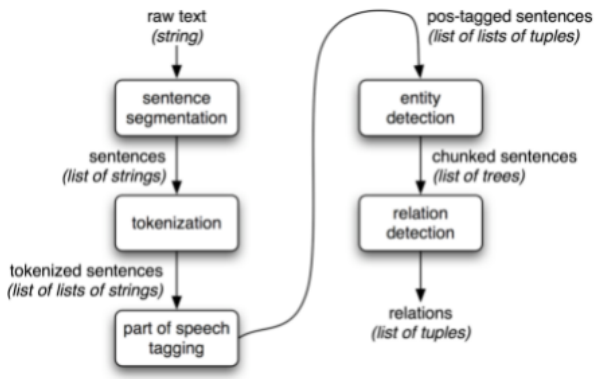

Figura 3 – *Pipeline* proposto por Xue *et al.* (2011)

Na mesma área de extração de informação, outro *pipeline* pode ser observado no tutorial de Feldman e Ungar (2014) para a Universidade Hebraica de Jerusalém e também para a Universidade de Pensilvânia. O *pipeline* é composto pelas seguintes etapas:

1. extração do texto cru;
2. *tokenização*;
3. detecção de termos limitantes;
4. detecção de limitantes de sentenças;
5. aplicação de *tag* nas informações;
6. *parse* do resultado;
7. determinação de referência;
8. extração de conhecimento.

2.1.3 *Stemming*

Como pode ser encontrado em Liu e Özsu (2009), *stemming* é o processo no qual uma palavra possui seu final ou outros afixos removidos ou modificados para que diferentes formas daquelas palavras que diferem de um modo não relevante possam ser consideradas iguais. Singh e Gupta (2017) dão um exemplo disso, as palavras em inglês *maintaining*, *maintained*, *maintenance* são transformadas na palavra raiz *maintain* através do *stemming*. Liu e Özsu (2009) definem que um programa de computador que aplica essa transformação na palavra é chamado de *stemmer* ou de algoritmo de *stemming*. Além disso, o resultado de um algoritmo de *stemming* é conhecido como *stem*. Singh e Gupta (2017) consideram o *stemming* com uma entre as numerosas técnicas de pré-processamento que são úteis nas áreas de recuperação de informação e processamento de linguagem natural.

Liu e Özsu (2009) cita que o primeiro *stemmer* para o inglês relatado na literatura foi desenvolvido no final da década de 60 por Julie Beth Lovins. Singh e Gupta (2017) afirmam que a área vem experienciando um crescimento considerável nos últimos anos, e com isso, um grande número de técnicas e algoritmos estão sendo propostos para ajudar na tarefa. Segundo os autores, isso acontece devido ao fato que o *stemming* é visto como uma técnica que melhora a acurácia da recuperação da informação, e, também porque das descobertas de que a técnica é mais benéfica em linguagens com complexa morfologia, onde uma única palavra pode ter muitas formas variadas de se escrever. No inglês, os dois algoritmos de *stemming* mais utilizados são o *Porter stemmer* e o *Paice/Husk stemmer*, segundo Liu e Özsu (2009).

Orengo e Huyck (2001) ao apresentar o *stemmer RSLP*, que é o utilizado no presente trabalho, dizem que o problema de *stemming* no inglês parece ser um problema resolvido. Isso por causa que o *Porter stemmer*, um simples algoritmo de remoção de sufixos baseado em regras, sem lista de exceções ou busca em dicionários, já foi provado ser efetivo nos sistemas mais elaborados. Por outro lado, os autores dizem que fazer um *stemming* para o português é muito mais complicado, já que ele possui uma morfologia mais complexa. Os pesquisadores apresentam os 8 passos que podem ser executados em uma palavra durante uma chamada ao RSLP, são eles:

1. redução do plural;
2. transformação da forma feminina para a masculina;
3. redução para advérbio com ajuda de lista de exceção;
4. redução da parte aumentativa ou diminutiva da palavra;
5. remoção dos afixos de nome;
6. remoção dos afixos de verbo;
7. remoção de vogal;
8. e por último; remoção de acentuação.

Por fim, Orengo e Huyck (2001) mostram que o RSLP foi avaliado através de três diferentes métodos, onde em todos foi melhor que a versão do *Porter stemmer* para o português.

2.2 Retorno sobre patrimônio líquido (ROE)

Zelgalve (2014) fala que o retorno sobre patrimônio líquido diz a porcentagem de lucro que uma empresa faz para cada unidade de dinheiro do patrimônio líquido da empresa. Enquanto Ichsani e Suhardi (2015) afirmam que o ROE é a razão utilizada para mensurar o sucesso de uma empresa em gerar lucro para os seus acionistas. Wet e Toit (2007) já afirmavam que o ROE é um do mais antigo e talvez mais utilizado, entre analistas financeiros e acionistas, indicador para mensurar o desempenho de uma empresa - informação que corroboram após apresentar essa afirmação feita por outros autores da área.

Apesar disso, Zelgalve (2014) faz questão de deixar claro que o ROE não especifica quanto de dinheiro será retornado para os acionistas, já que isso depende da decisão da empresa sobre o pagamento dos dividendos e também da variação do preço das ações. De

qualquer forma, é afirmado que trata-se de um bom indicador se a empresa é capaz de gerar um retorno que vale o risco do investimento.

Ichsani e Suhardi (2015) apresentam a versão mais comum para o cálculo do indicador. Essa versão considera que o ROE é a divisão do lucro líquido pelo patrimônio líquido, sendo que o resultado é multiplicado por 100. Além disso, Ichsani e Suhardi (2015) afirmam que historicamente a literatura considera que um bom ROE é quando esse valor é maior que 12.

2.3 Análise

2.3.1 Matriz de confusão

Sammut e Webb (2011) dizem que a matriz de confusão sumariza o desempenho de classificação de um classificador com relação a algum dado de teste. Na mesma linha, Beauxis-Aussalet e Hardman (2014), falam que a matriz de confusão é a principal forma de avaliar erros em problemas de classificação, quando é preciso colocar itens dentro de uma categoria. Também Freitas *et al.* (2007) falaram que a matriz de confusão fornece uma forma consistente de analisar o comportamento de um classificador.

Sammut e Webb (2011) explicam de maneira mais simplificada o que é uma matriz de confusão, ao dizer que se trata de uma matriz de duas dimensões, indexada em uma dimensão pelas classes verdadeiras de um objeto e na outra dimensão pelas classes que um classificador definiu. Já Freitas *et al.* (2007) concorda com esse tipo de definição, pois diz que esse tipo de matriz fornece uma representação de desempenho quantitativa para cada classificador com base no que o mesmo reconheceu, ou seja, definiu. Além disso, Freitas *et al.* (2007) fornece uma definição para a matriz de confusão através da equação (2.1).

$$A = \begin{bmatrix} RR_{1,1} & RR_{1,2} & \cdots & RR_{1,N} \\ \vdots & \vdots & & \vdots \\ RR_{2,1} & RR_{2,2} & \cdots & RR_{2,N} \\ \vdots & \vdots & & \vdots \\ RR_{N,1} & RR_{N,2} & \cdots & RR_{N,N} \end{bmatrix} \tag{2.1}$$

Onde RR_{ij} corresponde ao total de número de entidades na classe C_i que foram classificadas na classe C_j. Portanto, a diagonal principal de elementos indica o número de amostras na classe C_i corretamente reconhecida pelo classificador.

Beauxis-Aussalet e Hardman (2014) citam que uma aplicação típica da matriz de confusão é no aprendizado de máquina supervisionado. Elas também dizem os principais usos da matriz de confusão: inspecionar erros para cada classe; identificar possibilidades para melhorar os parâmetros de um software; comparar diferentes versões de um software

ou experimento. Em Sammut e Webb (2011) é falado que um caso especial da matriz de confusão é quando a mesma é utilizada com duas classes, uma que designa a classe positiva e a outra a classe negativa. Nesse contexto, as quatro células da matriz são chamadas de verdadeiros positivos, falsos positivos, verdadeiros negativos e falsos negativos. Beauxis-Aussalet e Hardman (2014) denominam esse caso especial de matriz de confusão como classificação binária e também define como erro tipo I os falsos positivos e como erro tipo II os falsos negativos.

Por fim, Sammut e Webb (2011) dizem que diversas métricas de desempenho de classificação podem ser definidas a partir da matriz de confusão. Como também, o trabalho de Beauxis-Aussalet e Hardman (2014) fala que os diferentes tipos de erro podem ter impactos diferentes (maior ou menor) conforme o contexto da aplicação muda.

2.3.2 *F1-score*

Chinchor e Sundheim (1993) fala que o *f-measure* (também conhecido como *f-score*) fornece uma forma de combinar *recall* e precisão para conseguir uma única métrica que fica entre ambos itens combinados. Isso acontece por causa que como dito por Sasaki *et al.* (2007), o *f-score*, na sua forma mais popular de *F1-score*, trata-se da média harmônica entre precisão (P) e *recall* (R), ou seja, $F = \frac{R}{P+R}$. Importante notar que *recall* e precisão podem ter pesos diferentes ao se calcular a *f-measure*, pois assim essa métrica possui mais flexibilidade para ser utilizada em diferentes aplicações (CHINCHOR; SUNDHEIM, 1993).

Como falado por Zhang *et al.* (2015), sabe-se que é mais informativo e mais útil utilizar o *F1-score* do que outras métricas como a acurácia. Para ficar claro a diferenciação entre *f-score* e *F1-score*, vale utilizar a explicação de Chinchor e Sundheim (1993), que define a fórmula para calcular o *f-score* como a equação (2.2), onde como é possível de se esperar, P é a precisão, R é o *recall* e β a importância relativa dada para o *recall* sobre a precisão. Por exemplo, se *recall* e precisão possuem o mesmo peso, então $\beta = 1$, já quando *recall* possui a metade da importância que a precisão possui, então $\beta = 0,5$, e, por fim, se *recall* possui o dobro de importância no sistema, o β será 2. Portanto, quando $\beta = 1$, então trata-se da média harmônica e do *F1-score*.

$$F = \frac{(\beta^2 + 1,0) \times P \times R}{\beta^2 \times P + R} \tag{2.2}$$

Sasaki *et al.* (2007) comenta em seu trabalho que recentemente ele percebe certa confusão na definição do que é *f-score*, o que segundo ele parece acontecer por uma falta de entendimento sobre como o *f-score* é calculado. Entendimento que pode ser obtido através do trabalho de Chinchor e Sundheim (1993), que preza por deixar bem claro que o *f-score* é uma combinação de *recall* e precisão para se ter uma única métrica de desempenho nos sistemas - definição que segundo Sasaki *et al.* (2007) é o suficiente para satisfazer a maioria

das pessoas que ficam em dúvida sobre o que é *f-score*. Além disso, Sasaki *et al.* (2007) acha curioso que haja uma informação em aberto sobre o *f-score*, que é a razão da fórmula ter sido chamada de *f*. Aliado isso ao fato de Zhang *et al.* (2015) e outros afirmarem que o *F1-score* é uma das métricas mais utilizadas para medir desempenho na classificação de texto. Tal dúvida sobre a nomenclatura tentou ser sanada por Sasaki *et al.* (2007), que no seu trabalho afirma ter feito contato pessoal com os envolvidos no trabalho de Chinchor e Sundheim (1993) e soube que o nome foi acidentalmente selecionado.

3 Trabalhos relacionados

De maneira simplificada, o presente trabalho busca através de técnicas de processamento de linguagem natural calcular indicadores financeiros a partir de relatórios de resultados empresariais. Logo, esta seção irá apresentar alguns dos trabalhos pesquisados que se relacionam de alguma forma com isso. A ideia aqui é resumir brevemente os selecionados, enquanto destaca os seus desenvolvimentos e contribuições. Por fim, uma tabela de comparação entre os trabalhos será apresentada.

3.1 *Understanding Financial Reports using NLP* (2019)

Um dos trabalhos selecionados é o *Understanding Financial Reports using NLP* de Sudhams e Saripalli (2019). O projeto visa consolidar e agregar em uma base de dados estruturada os relatórios de *Credit Default Swap* em participação de fundos mútuos que datam de 2004 até 2017. Além de extrair e agregar informações, busca-se montar uma aplicação web a partir da base de dados criada para facilitar futuros estudos associados aos relatórios de *Credit Default Swap*.

A principal motivação para a criação da base de dados é que a inexistência de uma acarreta em uma piora na condução compreensiva, na precisão e na realização de análises quantitativas em trabalhos que dependem dos relatórios de *Credit Default Swap*. O *Credit Default Swap* (CDS) é uma técnica associada ao risco de crédito e frequentemente atribuída como uma das responsáveis pela crise financeira de 2007/2008. Apesar desses relatórios possuírem informações valiosas, trabalhos anteriores ao de Sudhams e Saripalli (2019) utilizaram apenas pequenas quantidades de dados como tópico de estudo, principalmente devido à falta de padronização dos documentos. Para alcançar os objetivos propostos, Sudhams e Saripalli (2019), dividiram o trabalho em duas partes: uma específica para processamento de dados e outra para o processamento de linguagem natural. Na primeira parte busca-se unificar as informações estruturadas dos relatórios de maneira normalizada em uma base de dados através de um sistema baseado em regras, além de se fazer uma preparação das informações não estruturadas para aplicação das técnicas de processamento de linguagem natural. Já a segunda etapa, propõe uma técnica de processamento de linguagem natural, mais precisamente do campo de reconhecimento de entidade mencionada, que utiliza de anotações sequenciais (*sequence labelling*) com o uso de *Conditional Random Field* para destacar as informações mais importantes em sentenças não estruturadas dos documentos, pois essa característica de não padronização é algo consideravelmente presente nos documentos.

A parte de processamento de dados inicia com a etapa de coleta de dados, composta

Figura 4 – Fluxo de trabalho executado em (SUDHAMS; SARIPALLI, 2019)

pela execução de um *web crawler* composto por *scripts* em Python para coletar dezenas de gigabytes de relatórios financeiros dos sites que os disponibilizam. Com os dados em mãos, primeiro aplica-se uma fase de organização das informações, que consiste na alteração da estrutura dos diretórios obtidos via *script* para uma melhor rastreabilidade dos fundos e suas mudanças de nomes, além da remoção de arquivos desses diretórios que não mencionam *Credit Default Swap*. Após os arquivos necessários para extração das informações estarem disponíveis, percebe-se que muitos dos relatórios possuem os valores pertinentes ao *Credit Default Swap* dentro de tabelas em tags HTML, logo uma rotina para conversão desses valores dos relatórios para arquivos no formato csv é criada e aplicada Sudhams e Saripalli (2019), que também verificam a existência de dados não estruturados - que por serem dessa forma são separados para tratamento futuro. Após isso, os arquivos csv precisam que seu formato de dados sejam corrigidos e também que suas informações sejam limpas, já que há lixo do resultado da extração, como caracteres não desejados, tokens desnecessários e até mesmo ainda há arquivos que não possuem relação com *Credit Default Swap*.

Todos esses arquivos csv são unificados em um único arquivo csv, pois isso possibilitará um melhor uso e uma melhor apresentação dos dados. Para alcançar esse único arquivo csv é necessário a execução de alguns métodos: unificação de *headers* que representam o mesmo dado, mas que não possuem a mesma escrita em arquivos diferentes; normalização de valores e a sinalização de células que não possuem valores. O resultado da aplicação desses métodos é um único arquivo csv com 16813 linhas.

Com os dados estruturados organizados, Sudhams e Saripalli (2019) iniciam a preparação dos dados não estruturados para a aplicação posterior de PLN. Essa preparação é composta por alguns passos. Primeiro, há a remoção de todas as tags HTML dos relatórios.

Segundo, há uma filtragem, através de um *script*, para manter uma base somente com as frases que possuam relação com *Credit Default Swap*. Por fim, os textos filtrados são etiquetados manualmente com labels para serem exportados e servirem de entrada no treinamento dos modelos *conditional random field* de PLN.

Sudhams e Saripalli (2019) verificam a necessidade da aplicação do reconhecimento de entidade mencionada, mais precisamente do *sequence labelling*, nos dados não estruturados provenientes dos relatórios de *Credit Default Swap*. De maneira resumida, o *sequence labelling* serve para definir uma *label* para cada elemento de uma sentença. Sabe-se que algoritmos de *sequence labelling* baseados em *deep learning* geralmente possuem um melhor resultado, por outro lado o uso desses algoritmos dependem de uma grande quantidade de dados e a quantidade de informações não estruturadas nesse trabalho não atendem esse critério. Com isso, Sudhams e Saripalli (2019) optam por usar algoritmos de *sequence labelling* baseados em probabilidade, mais especificamente no modelo de *Conditional Random Field* (CRF). Só que antes da utilização desse modelo há a aplicação de alguns processos nos dados. Primeiro, associa-se uma *label* para cada palavra através da ferramenta de anotação de texto que os autores desenvolveram durante o trabalho. Depois, cada palavra tem sua tag de *part-of-speech* definida através da aplicação da biblioteca *NLTK*. Após essas duas etapas constata-se que dentro dos dados não estruturados não restam mais muitas palavras que não pertencem a nenhuma entidade relevante ao domínio do problema, e, para evitar um treinamento de modelo ruim devido a essa grande quantidade, essas palavras são removidas. Por fim, gera-se um conjunto de *feature* para cada palavra que possui características que serão relevantes para a aplicação do modelo de *Conditional Random Field*. A aplicação é realizado por último na sequência com o auxílio de uma biblioteca chamada CRFSuite implementada em C++ e conhecida por seu rápido desempenho.

O trabalho de Sudhams e Saripalli (2019) possui diversos resultados positivos. Um deles foi, devido a necessidade, a criação de uma ferramenta para anotação de texto, ou seja, possibilitar com isso a determinação manual de *labels* para palavras de um texto informado. Outro foi a criação de um modelo para aplicação do *sequence labelling* através do *Conditional Random Field*, modelo que teve sua performance validada de duas formas. A primeira foi com um *dataset* clássico do CONLL de 2003, que permite uma fácil comparação com outros modelos similares já propostos na área, onde o trabalho de Sudhams e Saripalli (2019) alcançou uma acurácia de 81,21%, o pior valor se comparar com os outros 4 modelos apresentados em comparação - o melhor dos 4 teve uma acurácia de 90,90%. Já o outro experimento de validação foi o modelo aplicado nos dados não estruturados de CDS, onde Sudhams e Saripalli (2019) conseguiram as seguintes médias ponderadas, dentro de todas as categorias de labels definidas, como resultado: 0,97 de precisão, 0,96 de score F1 e de sensibilidade. Além disso, os processos de extração de dados foram avaliados através da métrica de acurácia com um teste manual de fidelidade de dados onde foi feita a coleta de 100 relatórios aleatórios e a comparação das suas informações com os valores extraídos -

os processos geram uma acurácia de 91,33%. E também, o *dataset* consolidado associado com alguns outros resultados já citados foram unificados em uma aplicação web que tem dois propósitos: buscar informações dos relatórios utilizados como *input* do projeto de pesquisa, e, também permitir que se insira um diferente relatório de CDS para que se obtenha indicadores do mesmo. Finalmente, também mostrou-se que é possível através de consultas SQL no *dataset* gerado realizar análises qualitativas sobre os dados presente.

3.2 *CoFiF: a corpus of financial reports in french language* (2019) e *Stock Price Movement Prediction Using Representative Prototypes of Financial Reports* (2011)

Outros trabalhos selecionados foram o *CoFiF: a corpus of financial reports in french language* de Daudert e Ahmadi (2019) e o *Stock Price Movement Prediction Using Representative Prototypes of Financial Reports* de Lin *et al.* (2011). Ambos são similares nas técnicas que utilizadas para alcançar seus objetivos. Daudert e Ahmadi (2019) criam um *corpus* a partir de relatórios financeiros em francês, ele possui cerca de 188 milhões de *tokens* gerados após a consulta de 2655 documentos - desde relatórios trimestrais até documentos de referências. Enquanto isso, o trabalho de Lin *et al.* (2011) busca prever o preço de ações em um curto espaço de tempo a partir de informações quantitativas e qualitativas coletadas de cerca de 26255 relatórios financeiros de empresas.

A motivação para a realização do CoFiF, segundo Daudert e Ahmadi (2019), foi a falta de um corpora em francês para um domínio mais específico, como o de relatórios financeiros, associado com a necessidade cada vez mais presente do uso de processamento de linguagem natural em dados textuais na realização de pesquisas de *machine learning* ou inteligência artificial. Para compor esse corpus, o CoFiF utiliza de relatórios das principais companhias francesas, já que selecionou como fonte 40 empresas que compõem o CAC 40, ou seja, 40 das 100 empresas com mais capitalização de mercado no mercado de ações de Paris. Além das 40 citadas, também foi utilizada as informações das empresas do CAC Next 20 - 20 maiores companhias após as que já estão selecionadas no CAC40.

Já para o trabalho de Lin *et al.* (2011), a motivação surgiu da seguinte ideia, nos EUA há uma lei que obriga as empresas preencherem relatórios financeiros em uma base de dados centralizada, relatórios que por sua vez podem ser importante fonte para prever o preço de ações - conforme já apresentado em outros estudos, já que não possuem apenas informações que refletem o passado, mas também dados sobre performance futura. Esses relatórios possuem elementos quantitativos e qualitativos, e, alguns pesquisadores sugerem que incorporar esses dois tipos de elementos na predição do preço de ações pode melhorar a habilidade preditiva, porém ambos elementos foram considerados separadamente na

tarefa em trabalhos anteriores - nisso que Lin *et al.* (2011) se diferem, pois utilizam um esquema de ponderação para combinar variáveis quantitativas e qualitativas para predizer a movimentação em um curto espaço de tempo do preço das ações.

Para ser criado o CoFiF, Daudert e Ahmadi (2019) passaram pelas seguintes etapas: primeiro selecionou-se os tipos de documentos que iriam compor o *corpus*, que foram documentos de referências, relatórios financeiros anuais, relatórios financeiros semestrais e relatórios financeiros trimestrais. Depois escolheram o intervalo de tempo para coleta dos documentos - arquivos divulgados entre 1995 e 2019. Após isso, com as empresas definidas, do modo explicado anteriormente, houve o trabalho manual de acessar os sites de cada uma das empresas para buscar os documentos necessários, no formato *Portable Document Format* (PDF). Com os documentos em mãos, utilizou-se do software *pdftotext* para a extração do texto presente nos arquivos - nenhum pré-processamento foi aplicado nos *datasets* para evitar interferência na etapa de extração do documento. Por fim, realizou-se uma análise no corpus com o auxílio da *Natural Language Toolkit* (NLTK) que permitiu a segmentação e contagem de *tokens* e sentenças. Enquanto isso, o trabalho de Lin *et al.* (2011) propõe um método chamado HRK (*Hierarchical agglomerative and Recursive K-means clustering*) para a predição em curto espaço de tempo do preço das ações. O HRK possui quatro fases, onde a primeira consiste da extração de um vetor de *feature* a partir de informações quantitativas e qualitativas de cada relatório financeiro. A segunda é a divisão desses vetores de *features* em diferentes *clusters* através do HAC. Na terceira etapa é aplicado um *K-means* recursivamente em cada *cluster* até que o mesmo atinja um valor limiar adequado, esse valor é definido como o número de vetores de *features* da classe dominante dividido pelo número total de vetores de *features* no *cluster*. Após a execução do *K-means*, então é computado o centroide de cada *cluster*, que são chamados de protótipos representativos dos *clusters*. Por fim, o último passo é o uso desses protótipos representativos para predição do preço da ação em um curto espaço de tempo. O processo de predição pode ser visto da seguinte forma, quando um relatório financeiro é lançado, o mesmo serve de alimento para a primeira etapa, onde após a execução do método terá um protótipo representativo mais próximo possível de si que representa uma classe de ação - que pode ser de comprar ações, vender ações ou de não movimentação.

O corpus resultante a partir do CoFiF foi avaliado para demonstrar seu potencial em tarefas de PLN no campo econômico através de alguns experimentos. Para a realização dos experimentos criou-se dois modelos de linguagem treinados a partir de uma *forward RNN* (*recurrent neural network*) e de uma *backward RNN* com os seguintes parâmetros: `hidden_size 2048`, `nlayers 1`, `sequence_lenght 250`, `mini_batch_size 100` e `epochs 3`. Houve uma limpeza nos dados antes da criação dos modelos: remoção de linhas em branco repetidas, remoção de quebras de linhas, além do alinhamento do conteúdo no começo da linha se necessário. Os modelos de linguagens foram experimentados a partir de seus *words embeddings*, em uma tarefa para definir onde uma sentença começa e onde ela

termina (*sentence boundary detection*) foi obtido 0,91 no *F1-score*. Um outro experimento verificou a capacidade do modelo de prever se uma sentença provavelmente estaria no *corpus* - o teste passou para 200 sentenças (100 modificadas e 100 originais) com baixas medidas de perplexidade. Já o outro experimento treinou um modelo *word2vec* a partir do texto limpo do CoFiF e verificou a qualidade dos *word embeddings* obtidos de maneira qualitativa, por exemplo, o modelo sugere que a palavra "economia" terá como prováveis vizinhos as palavras "agricultura", "energia" e "inovação", e, na realidade esses termos realmente são essenciais para a economia francesa (e.g. a França foi o sexto maior produtor agrícola na união europeia em 2011). Com os resultados demonstrados concluiu-se que é possível capturar de maneira adequada relações entre os termos no CoFiF.

Enquanto isso, o método HRK, proposto por Lin *et al.* (2011), foi verificado através da complexidade de tempo do seu algoritmo e também por experimentação. De ambas as formas, comparou-se o HRK com SVM, *Naïve Bayes* e PFHC. Na parte de complexidade de tempo, uma análise separada foi feita para a parte de treinamento e outra para a predição, onde `m` é o número de *features*, quantitativas e qualitativa, e `N` é o número de relatórios financeiros. Na fase de treinamento o *Naive Bayes* se destaca com uma complexidade de `O(mN)`, enquanto que na parte de predição todos os algoritmos possuem resultados similares e tendem a ser `O(m)`, pois o número de vetores de suporte ou número de centroides (outros fatores que impactam na complexidade dessa etapa) costumam ser bem menores do que o número de *features*.

Para os experimentos, Lin *et al.* (2011), consideraram como *dataset* os relatórios financeiros anuais e trimestrais presentes na base de dados EDGAR das empresas que compõem o índice S&P 500 no período de 01/01/1995 até 31/12/2008. Além disso, o preço de abertura e fechamento do valor das ações dessas empresas em cada dia desse período também foram coletados. As companhias foram então classificadas em diferentes setores industriais conforme o GICS (*Global Industrial Classification System*) e utilizou-se os relatórios obtidos do período inicial até o começo de 2006 como uma base de treinamento, enquanto o restante ficou como uma base de teste. Duas métricas foram utilizadas para avaliar a performance nos experimentos: acurácia da predição e a média de lucro por trade no curto espaço de tempo. Os resultados encontrados no trabalho concluem que utilizar informações qualitativas e quantitativas leva a um melhor desempenho do que considerar somente uma delas.

Por fim, novamente comparou-se o HRK com outros métodos (SVM, *Naïve Bayes* e PFHC) onde se considera as métricas citadas anteriormente para verificar quem possui melhores resultados, e, o HRK, de maneira resumida, se sai melhor. Na métrica de acurácia de predição, o HRK obteve melhores números do que o método de *Bayes* e o método PFHC para todos os setores do GICS e também ultrapassou os valores obtidos pelo SVM em 7 dos 10 setores de GICS, sendo que o HRK obteve como melhor resultado 76,9% de

acurácia para o setor de bens de consumo e o pior valor de 57,7% para o setor de energia. Enquanto que na métrica de lucro médio por *trade*, o HRK possui todos os resultados melhores ao ser comparado com o PFHC, também ganha em 9 setores ao ser comparado ao *Bayes* e em 8 setores na comparação com o SVM - o destaque positivo de valor para o HRK foi a média de lucro no setor financeiro de 2,49%, por outro lado, o destaque negativo é a média de lucro de 0,03% no setor de saúde.

3.3 *Using Financial Reports to Predict Stock Market Trends With Machine Learning Techniques* (2015)

Também foi escolhido como trabalho relacionado a dissertação de mestrado, Zhang (2015), *Using Financial Reports to Predict Stock Market Trends With Machine Learning Techniques*. A dissertação propõe um *framework* que utiliza técnicas de *machine learning* em relatórios financeiros para prever a movimentação no preço de ações, principalmente técnicas de *deep learning* e algoritmos não supervisionados. É apresentada também uma revisão de literatura na aplicação de técnicas de *machine learning* e na obtenção de informações. Além disso, o *framework* é testado através da comparação dos seus resultados.

A principal motivação para a dissertação de Zhang (2015) foi o espaço aberto para melhoria na área de pesquisa que trata com predição dos preços de ações com o uso de técnicas de *machine learning*. É sabido que os mercados financeiros são sistemas não lineares e complexos com um significante nível de barulho nas informações, mas tais características não fazem esses mercados se comportarem de forma aleatória. O processo de negociação de ações é um dos modos que empresas podem conseguir capital, enquanto para os investidores uma forma de renda extra, processo que é influenciado por diferentes fatores, como desempenho da companhia e situações políticas. Com isso, devido a sua importância e seu caráter incerto, técnicas de *machine learning* e *data mining* vem sendo aplicadas na descoberta de padrões em séries de preço de ações, para que assim possa se prever a movimentação do preço atual das ações, só que por outro lado, trabalhos somente com informações numéricas e temporais não possuem uma grande acurácia e um desempenho estável. Evoluções nesses estudos que consideraram informações textuais provenientes de redes sociais e de portais de notícias as vezes conseguiram melhores resultados, mas ainda pecam na necessidade de grande processamento para cumprir a tarefa. Ao considerar todos esses fatores, Zhang (2015) utiliza de três modelos de *deep learning*, campo que vem atraindo bastante atenção recentemente por causa dos seus feitos ao resolver determinados tipos de problemas, para verificar a aplicações e eficiência deles na predição de movimentações nos preços de ações quando aplicados em relatórios financeiros ·

Para desenvolver o *framework* Zhang (2015) utiliza dois diferentes tipos de infor-

mações como *datasets*. O primeiro tipo de informação são os relatórios financeiros 8-K das empresas listadas no índice S&P 500 americano do período de 2002 até 2012, que estão disponíveis publicamente através de outra pesquisa. Já o segundo tipo de informação é o preço histórico das ações das empresas que fazem parte do primeiro *dataset*, informações obtidas através do site *Yahoo! Finance*.

Com os *datasets* em mãos, Zhang (2015) primeiro pega os textos dos relatórios financeiros e aplica um processo de normalização: "unigramas" que representam metadados são removidos, *stemming* (transformar uma palavra derivada na sua raiz) é aplicado, o texto passa a ser todo em caixa baixa e palavras não necessárias também são removidas. Depois ele aplica a etapa de seleção de *feature* com o propósito de treinar o modelo de aprendizado com *features* mais relevantes, aqui a seleção de *feature* é realizada com *bag-of-words* e o auxílio do algoritmo *Chi-square*. Após essa aplicação, vetores que representam relatórios do mesmo dia são unificados em um único vetor, já que no *framework* só pode haver uma única predição para o dia. Por fim, com os vetores unificados, uma matriz é obtida, nela cada linha representa um vetor de *feature* de um documento, mas para comparar diferentes documentos é necessário aplicar uma normalização nas linhas com o objetivo de verificar a importância daquela *feature* - para isso é utilizada a técnica *tf-idf*.

Além de todos esses passos nos relatórios financeiros, Zhang (2015) também realizou um processamento no preço histórico das ações, já que a mudança de valor de uma ação em um único dia pode não representar adequadamente um *dataset* temporal, logo, diferentes indicadores financeiros foram calculados a partir do *dataset* de preço histórico das ações. Os três modelos de *deep learning* escolhidos para serem adaptados e aplicados nos datasets até então preparados foram *Stacked Denoising Autoencoders* (SDA), *Deep Belief Network* (DBN) e *Recurrent Neural Networks-Restricted Boltzmann Machine* (RNN-RBM). Cada um desses modelos necessita de dois passos para implementação: pré-treinamento e ajuste (*fine-tuning*). No passo de pré-treinamento Zhang (2015) percebeu que, geralmente, ao aumentar o número de *epochs* há uma queda nas perdas da entropia cruzada (*cross-entropy loss*[1]). Enquanto que no passo de ajuste verificou-se que o aumento do número de *epochs* pode levar a um aumento do custo com entropia cruzada devido ao sobreajuste necessário quando os *epochs* de treino são muito largos. Fora os modelos de *deep learning*, Zhang (2015) utilizou dois clássicos métodos de *machine learning* para efeitos comparativos: SVM e *Random Forests*. Para implementação, Zhang (2015) operou principalmente com a linguagem de programação *python* e bibliotecas dessa linguagem, como *Natural Language Toolkit* (NLTK), *sklearn.svm*, *sklearn.ensemble*, *theano*, *matplotlib*, *progressbar* e *NumPy*.

O processo de desenvolvimento pode ser sintetizado com o seguinte *pipeline*:

1. *Dataset* de relatórios financeiros e *dataset* dos preços das ações são os dados de

[1] Mede o desempenho de um modelo de classificação. Quanto maior, mais chance do valor predito divergir do valor real.

entrada

2. Processamento aplicado sobre os dados de entrada

3. Construção dos modelos citados anteriormente

4. Operação dos modelos implementados

5. Avaliação dos resultados com o auxílio de métricas como matriz de confusão, taxa de precisão e *recall* que formam o *F1-score*, acurácia de precisão e estabilidade do modelo

6. Retorno para a etapa 2 (processamento) para ajustes que visam uma melhor avaliação na etapa 5

O *framework* proposto por Zhang (2015) teve sua performance avaliada em duas etapas, primeiro cada modelo teve seus resultados analisados individualmente, para depois uma comparação de todos contra todos foi aplicada. As métricas adotadas para as avaliações foram: matriz de confusão, taxas de precisão e *recall* para a formação do *F1-score*, acurácia da predição e estabilidade do modelo. Sendo que as métricas de precisão, *recall* e formação do *F1-score* foram separadas de um modo que avaliassem como o modelo se comporta quando busca predizer uma baixa ou uma alta nos preços das ações. Devido ao fato de que é necessário treinar um modelo para cada *dataset* relacionado a uma empresa e fazer isso para 500 itens seria irreal, Zhang (2015) então opta por realizar a avaliação de performance descrita anteriormente nas 15 companhias com mais quantidade de dados disponíveis que compõem o S&P 500. Zhang (2015) usa os relatórios financeiros de 2002 até 2008 como conjunto de treinamento, os de 2009 até 2010 como conjunto de validação e o de teste são os que foram liberados entre 2011 e 2012. Os três modelos de *deep learning* obtiveram bons resultados na comparação com os modelos tradicionais de *machine learning*, fato que pode ser observado ao considerar a acurácia de precisão média de cada modelo: 53,8% (SVM), 53,9% (*random forests*), 58,3% (SDA), 59% (DBN) e 59,4% (RNN-RBM). Além disso, já que Zhang (2015) criou um modelo para cada empresa com o intuito de evitar a mistura de fatores intrínsecos a cada uma delas que podem afetar o preço de suas ações, e, se considerar que os relatórios financeiros utilizados não são produzidos com muita frequência, então realizar a predição de ações somente com relatórios financeiros não é a melhor solução, pois combinar essa informação textual com outros tipos de informações financeiras irá gerar mais dados de entrada para o treinamento. Por fim, o *framework* proposto ainda precisa considerar alguns obstáculos da vida real na sua modelagem para ser totalmente efetivo na obtenção de lucro no processo de predição para realização de *trade* no mercado financeiro de ações.

3.4 *Detecting Document Structure in a Very Large Corpus of UK Financial Reports* (2014)

O artigo *Detecting Document Structure in a Very Large Corpus of UK Financial Reports* publicado por El-Haj *et al.* (2014) é outro trabalho selecionado como relacionado. Ele faz parte do CFIE, um projeto maior, sigla para *Corporate Financial Information Environment*. O CFIE possui como objetivo utilizar técnicas de PLN para verificar as consequências dos lançamentos de relatórios financeiros empresariais.

Já o artigo de El-Haj *et al.* (2014) possui os seguintes objetivos: identificar a estrutura de um relatório financeiro automaticamente e também determinar diferentes métricas como legibilidade e frequência de palavras para as seções individuais presentes nesses relatórios. Conforme relatado pelos autores, até o momento da escrita do artigo, a área de pesquisa para detecção de estruturas em documentos é mais focada em artigos científicos, livros e jornais, sendo que em 2009 houve até uma competição para estimular o desenvolvimento de mais pesquisa na área. Contudo, avanços nos resultados desse segmento de pesquisa e também a sua aplicação em outros tipos de documentos vêm sendo mais alcançados recentemente.

O *dataset* utilizado por El-Haj *et al.* (2014) para experimentação no trabalho consiste de 1500 relatórios financeiros de cerca de 200 das maiores empresas listadas no *London Stock Exchange*. Os relatórios datam do período de 2003 até 2012, onde em média cada empresa publicava sete relatórios por ano. O texto enfatiza, que diferente de outras pesquisas que utilizam relatórios financeiros de empresas listadas nas bolsas de valores americanas, não existe um padrão de relatório financeiro no Reino Unido, o que faz com que seja muito mais problemática a extração de informações a partir deste *dataset*. Com isso, os relatórios são arquivos no formato PDF que diferem entre si em quantidade de páginas, estilo e nomenclatura, ou seja, diferentes terminologias podem ser utilizadas por diferentes relatórios para identificarem o mesmo tipo de informação.

Para alcançar os objetivos propostos, os autores definem um processo com cinco grandes etapas para realizar a extração das estruturas: primeiro, detecção do conteúdo a partir do sumário do relatório; segundo, *parsing* do conteúdo detectado e extração dos *headers*; terceiro, detecção / correção dos números das páginas; quarto, criação de links entre os *headers* e suas páginas; e, por último, usar os marcadores para extrair as narrativas de cada seção. Vale frisar que esse processo só é aplicado em arquivos PDFs que são baseados em textos e não funciona em arquivos PDFs baseado em imagens.

No processo proposto, a primeira etapa que trabalha com o sumário é necessária por causa que os dados presente nesse tipo de página auxiliam os pesquisadores na detecção da estrutura do relatório, só que para realizar essa etapa foi necessário a extração manual dos nomes mais comuns presentes nos relatórios a partir de 50 documentos. Já a segunda

etapa serve para a partir do sumário do relatório financeiro obtido na etapa anterior se conseguir os nomes das seções e as páginas em que elas se encontram para futura extração, alguns detalhes, como nome de seções em mais de uma linha, tiveram que ser considerados para que um resultado satisfatório fosse alcançado. A terceira etapa serve para corrigir (quando necessário) o número de página obtido para cada seção no passo anterior, isso acontece porque nem sempre o número de página que o sumário aponta é a mesma página que o conteúdo está presente no arquivo PDF. Na quarta etapa os autores criaram uma ferramenta para inserir links que ligassem os títulos das seções e suas respectivas páginas, com o sentido de auxiliar o processo de extração de texto da quinta etapa. Por fim, nesta quinta e última etapa, determinou-se uma lista de *headers* genéricos de seções de maneira semiautomática com o auxílio de um especialista da área de finanças para que essa lista fosse usada como padrão na execução no *dataset* (já que como dito anteriormente, não há padrão nos relatórios financeiros britânicos). A execução no *dataset* para extração das narrativas (texto) foi realizada com a biblioteca *iText*.

As métricas de legibilidade de *Fog* e de *Flesch-Kincaid* foram aplicadas em uma amostra de 250 relatórios financeiros anuais do *dataset*, e, além delas, coletou-se a frequência de palavras nessa amostra. Com essas informações foi possível concluir que a legibilidade de um relatório completo nem sempre está associada com a legibilidade da primeira seção do mesmo, pois há uma diferença considerável se comparar o valor da métrica para cada item. Por exemplo, para a empresa *Inchape* o valor computado através de *Flesch-Kincaid* para todo o relatório é de 18, enquanto que para a primeira seção do mesmo relatório foi 26. Então, os autores concluíram que melhores conclusões nesse sentido requerem mais estudos. Com o intuito de validar que o processo proposto possui qualidade, os autores realizaram uma avaliação manual dos textos extraídos com *experts* da área de finança. Para isso, pegaram uma amostra de 100 relatórios financeiros aleatórios que nunca foram vistos pelos avaliadores e aplicaram o processo de extração, o resultado disso foi entregue para os avaliadores, junto com o relatório correspondente e um formulário para que eles avaliassem se os resultados batiam parcialmente ou completamente. A partir das respostas da avaliação manual calculou-se então métricas para o processo: o *F1-score*, precisão e sensibilidade. A avaliação manual pelos avaliadores foi realizada em duas etapas, para que fosse possível entre as etapas a realização de melhorias no processo proposto pelo artigo. Os resultados melhorados alcançados após a segunda etapa indicaram que essas "correções" foram de boa ajuda, dado que, por exemplo, o indicador de número de páginas detectadas de maneira subiu de 76,2% para 89,5% e o número de correspondências exatas foi de 84,8% para 88,01%.

3.5 Tabela comparativa

Trabalho	**Área do dataset**	**Dataset aberto?**	**Tamanho do dataset**	**Origem dataset**
Sudhams e Saripalli (2019)	Financeiro	Sim	14724 documentos	Site do SEC (EUA)
Daudert e Ahmadi (2019)	Financeiro	Sim	2655 documentos	CAC 40 + CAC Next 20 (França)
Lin *et al.* (2011)	Financeiro	Sim	26255 documentos	S&P 500 (EUA)
Zhang (2015)	Financeiro	Sim	Documentos de 2002 até 2012 de 15 empresas do S&P 500	S&P 500 (EUA)
El-Haj *et al.* (2014)	Financeiro	Sim	1500 relatórios financeiros	Empresas listadas na bolsa de Londres

Tabela 1 – Comparação dos trabalhos relacionados (continua)

Trabalho	**Técnicas utilizadas**	**Compara técnicas utilizadas?**
Sudhams e Saripalli (2019)	*scripts* para coleta de dados; *scripts* para limpeza de dados; modelo de *Conditional Random Field*	Somente o modelo de *Conditional Random Field* é comparado com trabalhos externos
Daudert e Ahmadi (2019)	*nltk* (ferramenta criação corpus); *pdftotext* (ferramenta criação corpus); RNN (experimento); *Word2vec* (experimento)	Não
Lin *et al.* (2011)	HAC (método); *K-means* (método); SVM (comparação); *Naïve bayes* (comparação); PFHC (comparação)	Sim
Zhang (2015)	SDA (modelo); DBN (modelo); RNN-RBM (modelo); SVM (modelo); *Random Forests* (modelo)	Sim
El-Haj *et al.* (2014)	*iText* (ferramenta)	Sim, compara resultados em duas etapas do processo

Tabela 2 – Comparação dos trabalhos relacionados (continua)

Trabalho	**Propósito**	**Forma de avaliação**
Sudhams e Saripalli (2019)	Criar um *dataset* estruturado e uma aplicação web dos relatórios de *Credit Default Swap*	Avaliação de desempenho das etapas propostas
Daudert e Ahmadi (2019)	Criar um corpus da língua francesa a partir de relatórios financeiros	Experimentação do corpus através de três tarefas de PLN
Lin *et al.* (2011)	Prever o preço de ações em um curto espaço de tempo a partir de relatórios financeiros	Comparação do desempenho do método proposto com outras técnicas
Zhang (2015)	Verificar a aplicabilidade de modelos de *deep learning* na predição do preço de ações com o uso de relatórios financeiros	Comparação após experimentação dos modelos de *deep learning* com modelos clássicos de *machine learning*
El-Haj *et al.* (2014)	Identificar e extrair diferentes seções de relatórios financeiros	Manual com *experts* da área

Tabela 3 – Comparação dos trabalhos relacionados (continua)

Trabalho	**Métricas**	**Resultado**
Sudhams e Saripalli (2019)	Acurácia na extração de dados; *F1-score*; Precisão; Sensibilidade;	*Dataset* construído, aplicação web desenvolvida e também gerou-se outros produtos menores durante o projeto
Daudert e Ahmadi (2019)	*F1-score*; Perplexidade; Avaliação qualitativa.	*Corpus* gerado possui potencial de ser utilizado na criação de novas pesquisas na área
Lin *et al.* (2011)	Acurácia da predição; Média de lucro por *trade* no curto espaço de tempo; Complexidade de tempo	Método alcança o propósito determinado inicialmente e permite a aplicação, se adaptado, em outras áreas
Zhang (2015)	Matriz de confusão; Taxa de precisão e *recall* que formam o *F1-score*; Acurácia de precisão; Estabilidade do modelo	Modelos de *deep learning* possuem um desempenho relativamente bom
El-Haj *et al.* (2014)	*Gunning fog index* para legibilidade; *Flesch-Kincaide score* para legibilidade; Precisão; Sensibilidade; *F1-score*	Uma porcentagem alta de extrações corretas (no geral acima de 85%) conforme métricas apresentadas e obtidas a partir da avaliação manual

Tabela 4 – Comparação dos trabalhos relacionados (conclusão)

4 Modelo que calcula indicadores financeiros

4.1 Tecnologias utilizadas

A linguagem de programação escolhida para ser utilizada no desenvolvimento do trabalho foi o *python*, na sua versão `3.7.6`. Os principais motivos para tal escolha foram: trata-se de uma linguagem muito utilizada em trabalhos de inteligência artificial, mais precisamente de processamento de linguagem natural, fato que pode ser verificado através de alguns trabalhos relacionados; a linguagem possui um vasto ambiente com diversas bibliotecas disponíveis que auxiliam na execução das tarefas; por fim, também a facilidade do aluno com a linguagem, já que o mesmo teve contato anteriormente com ela.

Vale citar também tecnologias e ferramentas que permitiram a escrita do código de maneira mais adequada:

- *git* e *github*: o primeiro sendo um sistema de versionamento de código distribuído e o segundo uma plataforma online que fornece hospedagem online para utilização do primeiro. O uso de ambos facilita na preservação do histórico do projeto e também na divulgação do trabalho, já que o código fonte pode ser acessado através do seguinte link[1].

- *unittest*: *framework* de teste unitário padrão do *python*. Utilizado no projeto para validar se o código desenvolvido funciona conforme o esperado e também nos experimentos finais que buscam os indicadores financeiros, pois dessa forma percebe-se mais facilmente o impacto que as alterações realizadas geram no resultado final apenas executando uma bateria de testes.

- *Flake8*: *linter* para *python* que força determinados padrões de estilo no código fonte. Com essa ferramenta se almeja um código final do projeto mais limpo e organizado. Versão `3.7.6` utilizada.

- *poetry*: ferramenta que entre as suas funcionalidades faz a organização de dependências de um projeto em *python*. Com ele, a instalação e execução do trabalho em diferentes computadores é facilitada, pois todas as dependências necessárias e suas versões já estão especificadas diretamente no código do projeto, o que também diminui a probabilidade de erros devido a configuração de ambientes diferentes.

[1] http://www.github.com/lucasjoao/tcc

Como o *python* não possui por padrão todas as bibliotecas necessárias para a realização do trabalho, foi necessário instalar as seguintes bibliotecas que compõem as dependências do projeto:

- *PyPDF2*: biblioteca escrita puramente em *python* para manipulação de arquivos no formato PDF. Desenvolvida como um projeto *open source* por uma empresa americana de desenvolvimento de software chamada *Phaseit*. No trabalho é uma das bibliotecas utilizadas em determinadas configurações dos experimentos que é responsável por transformar o conteúdo do arquivo no formato PDF em texto do tipo *string* em *python*. Versão`1.26` utilizada.

- *nltk*: biblioteca especializada no auxílio de construção de programas em *python* que lidam com a linguagem humana, ou seja, no processamento de linguagem natural. No projeto foi utilizado em diversas etapas: *tokenização*, auxílio para a remoção de *stopwords* e no processo de *stemming*. Versão `3.5` utilizada.

- *numpy*: considerada uma biblioteca fundamental para a execução de projetos científicos em *python*. Instalada por ser a dependência de outras bibliotecas, como o *nltk*. Versão `1.19` utilizada.

- *tesseract*: uma ferramenta que possui um programa de reconhecimento ótico de caracteres (sigla OCR em inglês), ou seja, especializado em reconhecer textos a partir de imagens. Desenvolvido inicialmente pela *HP* em meados da década de 80, vem sendo mantido pelo *Google* desde 2006. De forma similar ao *PyPDF2* foi uma das bibliotecas utilizadas para transformar os dados dos arquivos em formato PDF, que foram convertidos para imagens, para *string* em *python*. Versão `4.1.1` utilizada.

- *pytesseract*: um *wrapper* em *python* do *tesseract*. No trabalho seu uso foi necessário para facilitar as chamadas e diferentes configurações via código da biblioteca *tesseract*. Versão `0.3.6` utilizada.

- *pdf2image*: biblioteca com a única funcionalidade de converter arquivos em formato PDF em imagens, faz isso com o auxílio de outras duas bibliotecas: *pdftoppm* e *pdftocairo*. No presente trabalho serviu para auxiliar o *pytesseract* no processo de transformação de arquivo PDF para *string* em *python*, o *pdf2image* foi responsável por fazer parte do processo e transformar os arquivos PDFs em imagens que serviriam de *input* para o *pytesseract*. Versão `1.14` utilizada.

- *pillow*: biblioteca que quando presente em uma aplicação adiciona para ela a capacidade de processamento de imagem. Aqui serviu para fornecer a tipagem PIL, que é a representação da imagem que o *pdf2image* utiliza quando converte o arquivo PDF. Com isso, essa é uma dependência existente por causa da utilização do *pdf2image*. Versão `8.0.1` utilizada.

4.2 Fluxo de trabalho

Durante a execução do trabalho houve uma pesquisa de recomendações e exemplificações de possíveis fluxos de trabalho para a aplicação de processamento de linguagem natural e extração de informação. Dessa pesquisa pode-se destacar os seguintes resultados: o *pipeline*[2] para extração de informação proposto pelo livro *Natural Language Processing with Python - Analyzing Text with the Natural Language Toolkit* de Steven Bird e outros autores; o *Cross-industry standard process for data mining* (CRISP-DM) exemplificado na prática através do seguinte artigo[3] na plataforma do *medium towards data science*; e, também o *pipeline*[4] de extração de informação proposto no slide 61 em um tutorial sobre *Sentiment Mining from User Generated Content* da *University of Pennsylvania*, dos professores *Feldman* e *Ungar*. Isso serviu de insumo, junto com os aprendizados da parte prática, para a determinação dos três fluxos de trabalho utilizados no projeto, um fluxo que mostra o funcionamento do código de modo macro, outro para o funcionamento do código de maneira mais detalhada e outro fluxo que apresenta como o processo de experimentação foi realizado.

O funcionamento do código pode ser visto de um modo mais macro através desse fluxo:

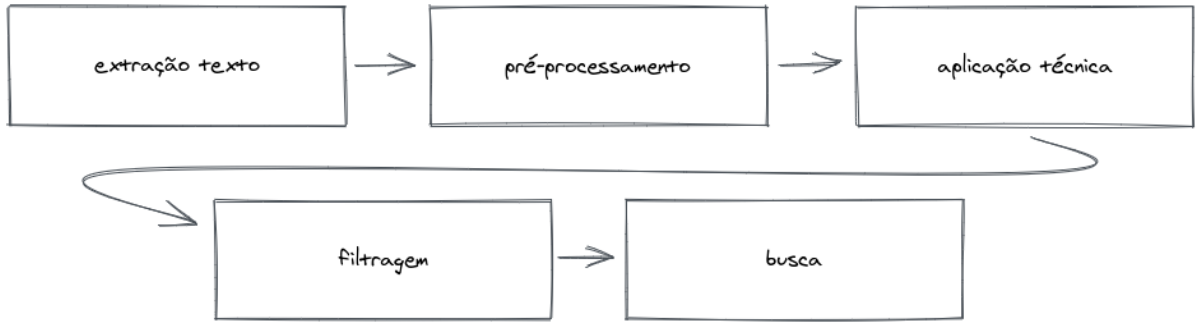

Figura 5 – Fluxo de trabalho macro (figura do autor)

Esse funcionamento macro pode ser detalhado ainda mais através desse outro fluxo de trabalho:

[2] https://www.nltk.org/book/ch07.html#fig-ie-architecture
[3] https://towardsdatascience.com/a-practitioners-guide-to-natural-language-processing-part-i-processing-understanding-text-9f4abfd13e72
[4] https://www.cis.upenn.edu/ ungar/AAAI/

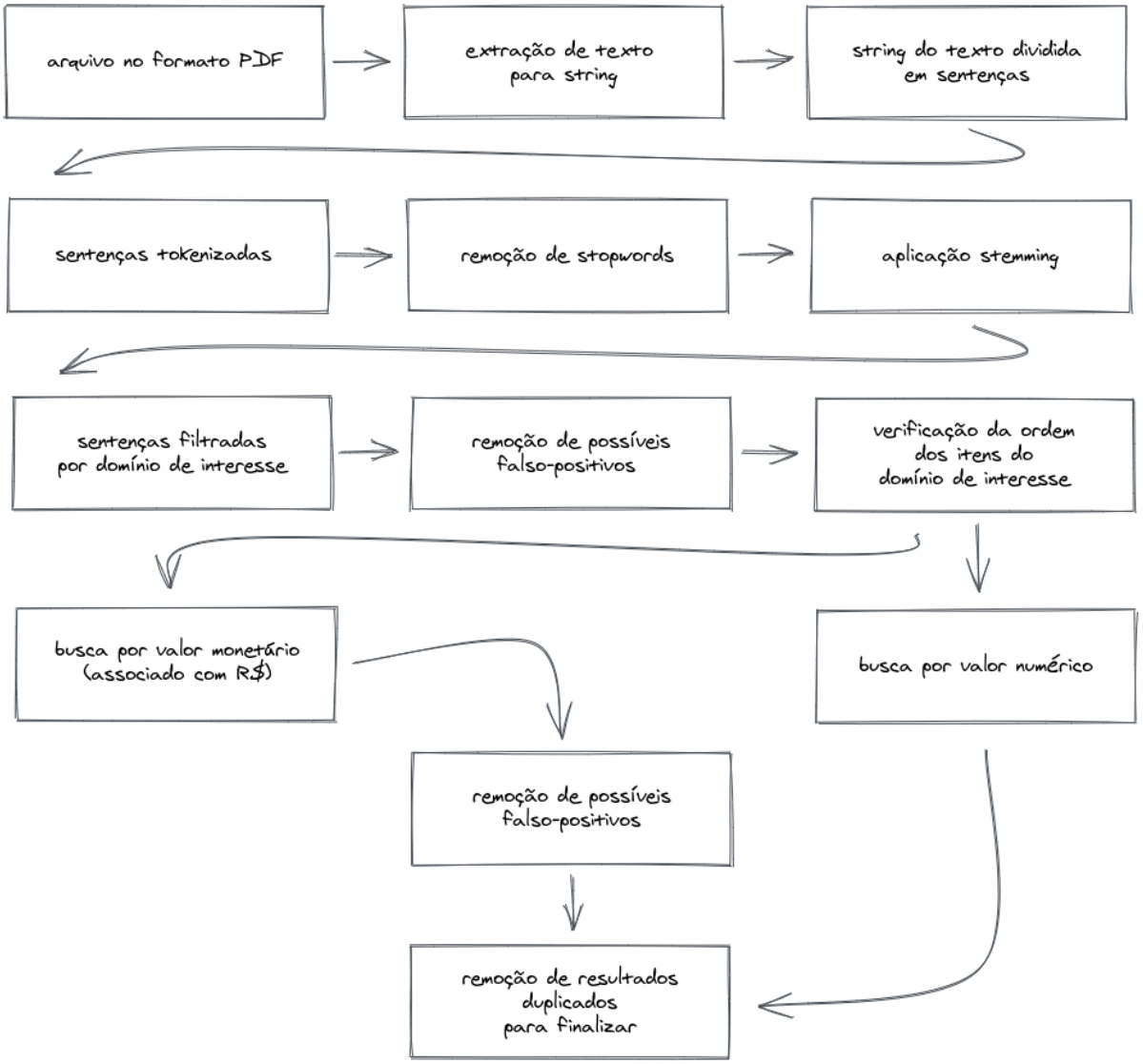

Figura 6 – Fluxo de trabalho detalhado (figura do autor)

Mais informações sobre o último fluxo de trabalho serão concedidas nas seções adequadas posteriormente. Para finalizar, o processo de experimentação possui o seguinte fluxo:

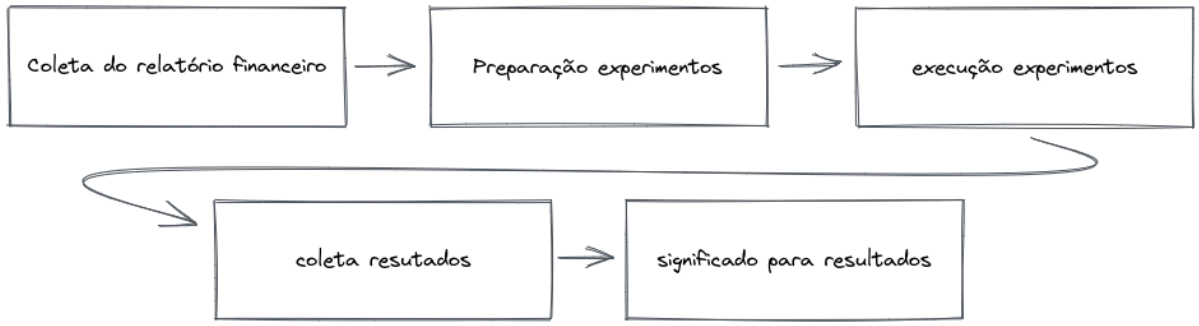

Figura 7 – Fluxo de trabalho experimentação (figura do autor)

Fluxo que possui os seguintes detalhes:

1. **Coleta do relatório financeiro**: coleta manual do arquivo com formato PDF através dos sites de RI (relação com investidores) das empresas alvos. No fluxo de experimentação, essa etapa não precisa ser sempre executada, já que cada novo relatório adicionado no *dataset* exige um incremento em todos os experimentos já realizados.

2. **Preparação de experimentos**: somente executado quando trata-se da verificação de uma nova configuração específica de alguma biblioteca utilizada pela solução que pode alterar o resultado final. Quando isso acontece então cria-se uma nova bateria de testes para essa configuração. Mais informações sobre os experimentos serão fornecidas na seção correta do presente trabalho.

3. **Execução de experimentos**: se o passo anterior tiver sido realizado, então a bateria de testes unitários criados lá será executada e observada. Já se o passo anterior não tiver sido executado, o que significa uma mudança no código ou adição de um novo relatório financeiro, então todas as baterias de testes já escritas devem ser executadas e observadas novamente.

4. **Coleta resultados**: observações e resultados são compilados em tabelas e textos / comentários. Mais informações sobre os resultados serão fornecidas na seção correta do presente trabalho.

5. **Significado para resultados**: resultados compilados no passo anterior são organizados de maneira a passar um conhecimento sobre o resultado dos experimentos, isso pode ser feito através da transformação de informações qualitativas em quantitativas conforme critérios estabelecidos ou através da comparação com outros experimentos.

4.3 Histórico de tentativas

O presente trabalho é resultado de uma série de tentativas experimentais visando o alcance dos objetivos do trabalho. Inicialmente buscou-se encontrar e calcular os indicadores dos relatórios financeiros sem a ajuda de uma biblioteca externa de processamento de linguagem natural. O código dessa tentativa pode ser analisado no seguinte link[5] que representa uma *tag* do versionamento do trabalho no *github*.

Nesse esforço inicial o código possuía a seguinte estrutura:

- Um leitor de arquivos no formato PDF através da biblioteca *PyPDF2*
- Um buscador (classe `Searcher`) responsável por duas tarefas:
    1. Fazer um pré-processamento básico, que era transformar todas as palavras envolvidas no processo em letras minúsculas.
    2. Buscar em um determinado texto de entrada as palavras alvos da busca. Processo feito da seguinte forma: cada palavra do conjunto alvo determinado na inicialização do buscador (e.g. conjunto com "patrimônio líquido") era procurada no texto de entrada com a função `find` nativa do *python*. Se encontrava algo, então a busca parava. O `find` era utilizado ao invés do `in` devido a necessidade de se saber onde a palavra foi encontrada no texto de busca, pois com essa posição definida, se retornava a palavra (possivelmente um valor numérico) à direita da palavra encontrada. De maneira exemplificada, considerando um buscador inicializado para pegar as palavras "patrimônio líquido" e uma frase de *input* "A empresa X tem o patrimônio líquido 100", com o `find` ou o `in` é possível saber se na frase as palavras patrimônio líquido estão presentes, mas somente o `find` retorna a posição dessa palavra na frase, o que faz ser possível através de uma manipulação retornar o provável valor numérico posterior as palavras encontradas na frase.
- Um calculador (classe `RoeCalculator`) responsável por calcular o indicador ROE. Cálculo realizado da seguinte forma:
    1. Utilizar um `Searcher` de ROE no texto alvo, com o conjunto de busca "ROE" e "*retorno on equity*" indicados na inicialização.
    2. Se o `Searcher` de ROE encontrar algo, então retornar o valor encontrado. Se o `Searcher` de ROE não encontrar algo, então a próxima etapa seria dois novos `Searcher` no texto alvo para buscar patrimônio líquido e lucro líquido, um iniciado com o conjunto de busca "patrimônio líquido" e o outro com "lucro líquido".

[5] https://github.com/lucasjoao/tcc/tree/v0.1

3. Se os dois `Searcher` anteriores encontrarem algo, teoricamente o patrimônio líquido e o lucro líquido, então calcular o ROE a partir desses valores com a simples fórmula de $(lucro\_liquido/patrimonio\_liquido) * 100$. Se algum dos `Searcher` não encontrar nada, então retornar o valor `None` do *python*.
4. Importante citar que quando existia resultado a partir do passo anterior, ele era arredondado em duas casas decimais. Também vale notar que o calculador não considera possíveis diferenças em escalas no texto (i.e. se o valor numérico encontrado está em milhares ou milhões).

Ao colocar a "mão na massa" para testar a abordagem apresentada anteriormente percebeu-se duas grandes dificuldades que demandaram as alterações até o resultado final. A primeira dificuldade foi a que com frases simplórias de *inputs* para serem processadas a busca já não tinha o resultado esperado, logo assumiu-se que com os textos bem maiores dos relatórios isso seria bem pior. A segunda dificuldade foi com o leitor de arquivos no formato PDF, o *PyPDF2* mostrou bem pouco suporte para a extração de texto, verificação que levou a experimentação de um novo leitor. Um exemplo da dificuldade do *PyPDF2* pode ser visto na figura 9, proveniente do relatório financeiro da empresa *WEG* do segundo trimestre de 2019, já no mesmo relatório, o *PyPDF2* funciona de maneira positiva quando se depara com dados formatados como na figura 8.

Resultado Líquido

O lucro líquido no 2T19 foi de R$ 389,0 milhões, com crescimento de 15,6% em relação ao 2T18 e de 26,8% em relação ao 1T19. A margem líquida atingiu 11,8%, 0,8 ponto percentual superior ao 2T18 e 1,3 ponto percentual superior ao 1T19.

Figura 8 – Texto que o *PyPDF2* extrai corretamente (página 5 do relatório)

Balanço Patrimonial Consolidado

Valores em R$ mil

	junho 2019 (A)		dezembro 2018 (B)		junho 2018 (C)			
	R$	%	R$	%	R$	%	(A)/(B)	(A)/(C)
ATIVO CIRCULANTE	**8.803.475**	**59%**	**9.438.581**	**61%**	**9.968.378**	**64%**	**-7%**	**-12%**
Disponibilidades	2.782.754	19%	3.529.888	23%	4.257.196	28%	-21%	-35%
Créditos a Receber	2.474.176	17%	2.440.844	16%	2.589.700	17%	1%	-4%
Estoques	2.583.387	17%	2.458.410	16%	2.328.357	15%	5%	11%
Outros Ativos Circulantes	963.158	6%	1.009.439	7%	793.125	5%	-5%	21%
REALIZÁVEL A LONGO PRAZO	**1.106.020**	**7%**	**1.178.926**	**8%**	**791.825**	**5%**	**-6%**	**40%**
Aplicações Financeiras	573.757	4%	562.782	4%	-	0%	2%	n.m
Impostos Diferidos	148.729	1%	142.669	1%	174.450	1%	4%	-15%
Outros Ativos não circulantes	383.534	3%	473.475	3%	617.375	4%	-19%	-38%
PERMANENTE	**4.998.311**	**34%**	**4.782.343**	**31%**	**4.703.843**	**30%**	**5%**	**6%**
Investimentos	19.974	0%	20.362	0%	17.013	0%	-2%	17%
Imobilizado Líquido	3.586.364	24%	3.541.954	23%	3.406.149	22%	1%	5%
Intangível	1.188.174	8%	1.220.027	8%	1.280.681	8%	-3%	-7%
Direito de uso	203.799	1%	-	0%	-	0%	n.m	n.m
TOTAL DO ATIVO	**14.907.806**	**100%**	**15.399.850**	**100%**	**15.464.046**	**100%**	**-3%**	**-4%**
PASSIVO CIRCULANTE	**4.400.792**	**30%**	**5.034.004**	**33%**	**4.953.204**	**32%**	**-13%**	**-11%**
Obrigações Sociais e Trabalhistas	374.793	3%	240.346	2%	330.444	2%	56%	13%
Fornecedores	809.232	5%	842.957	5%	973.788	6%	-4%	-17%
Obrigações Fiscais	144.544	1%	88.183	1%	141.475	1%	64%	2%
Empréstimos e Financiamentos	1.486.659	10%	2.049.093	13%	1.988.080	13%	-27%	-25%
Dividendos e Juros S/ Capital Próprio	162.448	1%	165.441	1%	144.820	1%	-2%	12%
Adiantamento de Clientes	511.514	3%	655.242	4%	544.865	4%	-22%	-6%
Participações nos Resultados	117.287	1%	167.941	1%	107.363	1%	-30%	9%
Instrumentos Financeiros Derivativos	1.785	0%	12.070	0%	32.693	0%	-85%	-95%
Arrendamento Mercantil	48.220	0%	-	0%	-	0%	n.m	n.m
Outras Obrigações	744.310	5%	812.731	5%	689.676	4%	-8%	8%
PASSIVO NÃO CIRCULANTE	**2.338.568**	**16%**	**2.512.589**	**16%**	**2.965.058**	**19%**	**-7%**	**-21%**
Empréstimos e Financiamentos	1.424.176	10%	1.723.021	11%	2.169.171	14%	-17%	-34%
Outras Obrigações	141.828	1%	155.394	1%	156.633	1%	-9%	-9%
Arrendamento Mercantil	153.613	1%	-	0%	-	0%	n.m	n.m
Impostos Diferidos	76.751	1%	86.537	1%	97.613	1%	-11%	-21%
Provisões para Contingências	542.200	4%	547.637	4%	541.641	4%	-1%	0%
PARTICIPAÇÕES MINORITÁRIAS	**150.546**	**1%**	**138.983**	**1%**	**140.218**	**1%**	**8%**	**7%**
PATRIMÔNIO LÍQUIDO	**8.017.900**	**54%**	**7.714.274**	**50%**	**7.405.566**	**48%**	**4%**	**8%**
TOTAL DO PASSIVO	**14.907.806**	**100%**	**15.399.850**	**100%**	**15.464.046**	**100%**	**-3%**	**-4%**

Figura 9 – Tabela que o *PyPDF2* não reconhece e por isso não extrai (página 13 do relatório)

Contudo, apesar das alterações realizadas, algumas ideias da versão `0.1` foram reaproveitadas no resultado final:

- A necessidade de uma classe `Searcher` que realiza as buscas conforme algoritmos específicos para os casos de uso;
- O leitor de arquivos PDF com mais de uma biblioteca;
- A ideia do calculador, ou seja, uma classe responsável por ter informações específicas, como o cálculo, de cada indicador envolvido.

Por fim, outras mudanças não citadas aqui dessa versão inicial até o resultado atual surgiram devido a necessidades emergentes no trajeto de desenvolvimento e que serão explicitadas posteriormente ao explicar o código final.

4.4 *Dataset*

O *dataset* é composto por arquivos no formato PDF que representam as demonstrações de resultados financeiros trimestrais divulgados por empresas que estão listadas na B3, a bolsa de valores brasileira. Apesar da similaridade no conteúdo dos relatórios de diferentes empresas, esses arquivos, no geral, não apresentam um padrão comum de *layout* e informações disponíveis.

Em 2019 a B3 apresentava mais de 300 empresas listadas, ou seja, mais de 300 possibilidades de empresas que seriam escolhidas para terem seus relatórios analisados no presente trabalho. Para realizar um filtro inicial, alguns critérios para a escolha das empresas que poderiam ter seus relatórios escolhidos foram definidos:

- Participação no *Ibovespa*: o *Ibovespa* (IBOV) é o principal indicador de desempenho das ações negociadas na B3, reavaliado a cada quatro meses, ele é o resultado de uma carteira teórica de ativos. Para fazer parte do índice, a ação de uma empresa deve atender a determinados critérios de seleção. De maneira simplificada, pode-se considerar que as empresas mais importantes do mercado de ações brasileiro fazem parte desse índice. Por isso, um critério de aceitação para que a empresa fosse escolhida para ter seu relatório analisado no trabalho, era ter participação no IBOV de no mínimo 0,25
- Datas dos relatórios: com o intuito de trabalhar com empresas que estão a mais tempo listadas na B3, as datas dos relatórios foram outro critério importante de aceitação. Vale citar também que a possibilidade de utilizar relatórios de datas diferentes permite que o processo não fique enviesado por causa de um estilo gráfico

ou de um acontecimento de um determinado ano. Logo, outro critério de filtragem para a empresa poder ser escolhida foi a necessidade de ser possível ter acesso aos relatórios de resultados trimestrais da empresa de, no mínimo, 2010, ou seja, se a empresa não divulga um resultado trimestral de 2010, então ela não poderia ser selecionada.

- Classificação setorial: a B3 classifica as empresas em três níveis distintos: setor, subsetor e segmento. Para essa classificação se considera principalmente os tipos e usos dos produtos/serviços realizados pela empresa que mais auxiliam na formação da sua receita. Sendo assim, o último critério de filtragem é que não tenha mais de uma empresa por segmento, ou seja, cada segmento só pode ter no máximo uma empresa selecionada para estudo no presente trabalho. Dessa forma, diferentes segmentos seriam verificados e o trabalho não ficaria com um viés específico para um único segmento.

Para aplicar o filtro do critério de participação no *Ibovespa* utilizou-se de uma planilha distribuída pelo analista CNPI Fabio Holder que possui diversas informações sobre as empresas listadas na bolsa de valores, dentro dessas informações está a participação da empresa no *Ibovespa*. Dessa forma, de 330 empresas possíveis de serem escolhidas, ficaram 77 opções. Após essa primeira filtragem, empresas foram escolhidas arbitrariamente para então verificar nos seus sites de relação com investidores o critério citado anteriormente de "datas relatórios", ou seja, desde quando os relatórios trimestrais eram divulgados. Se tudo estivesse certo nesses dois quesitos e ela fosse de um segmento ainda não utilizado no trabalho, então a empresa seria uma boa candidata. Com isso, as empresas escolhidas foram:

Empresa	**Participação no IBOV$^{\text{dez/2020}}$**	**Relatórios desde**	**Classificação (segmento)$^{\text{conforme B3}}$**
Ambev	3,1%	2002	Cervejas e refrigerantes
ENGIE	0,5%	2005	Energia elétrica
Fleury	0,4%	2008	Serviços médico-hospitalares, análises e diagnósticos
Gerdau	1,1%	1999	Siderurgia
WEG	2,6%	2010	Motores, compressores e outros

Tabela 5 – Empresas escolhidas para compor o *dataset*

Dispondo das empresas definidas, precisava-se escolher quais relatórios em específicos seriam empregados no trabalho. Como a quantidade possível de relatórios era muito

alta (no mínimo 200, pois 5 empresas com 4 relatórios por ano em um período de 10 anos) e o processo de inclusão no trabalho é de certa forma manual (conforme será visto futuramente), não houve nenhum critério para a escolha dos períodos dos relatórios - a única restrição inicial era que no mínimo 10 relatórios deveriam ser utilizados.

Por fim, chegou-se na seguinte tabela que apresenta exatamente os relatórios escolhidos para cada empresa.

Empresa	**Período relatório**
Ambev	2019 - 1T (primeiro trimestre)
ENGIE	2019 - 2T (segundo trimestre)
ENGIE	2020 - 2T (segundo trimestre)
Fleury	2019 - 3T (terceiro trimestre)
Fleury	2020 - 2T (segundo trimestre)
Gerdau	2015 - 3T (terceiro trimestre)
Gerdau	2017 - 1T (primeiro trimestre)
Gerdau	2018 - 4T (quarto trimestre)
Gerdau	2019 - 2T (segundo trimestre)
WEG	2010 - 2T (segundo trimestre)
WEG	2015 - 1T (primeiro trimestre)
WEG	2017 - 2T (segundo trimestre)
WEG	2019 - 2T (segundo trimestre)

Tabela 6 – Relatórios financeiros escolhidos para compor o *dataset*

A coleta dos relatórios foi realizada manualmente da seguinte forma: entrar no site de relação com investidores da empresa; encontrar o relatório do período definido; realizar download e colocar o arquivo com formato PDF no diretório apropriado (`data/`) do projeto.

4.5 Extração do texto

A extração do texto corresponde ao primeiro item, de mesmo nome, do fluxo de trabalho macro (figura 5), enquanto que no fluxo de trabalho detalhado (figura 6) corresponde aos dois primeiros itens: **arquivo no formato PDF** e **extração de texto para *string***. Ao começar a falar sobre essa etapa torna-se necessário começar a apresentar o código da plataforma, um dos resultados desse trabalho, com mais detalhes.

Portanto, é importante ressaltar que na plataforma o grande responsável por orquestrar a operação no trabalho é a classe `manager.py`. Essa classe será dissecada com o decorrer das próximas seções.

Para a extração de texto, quem é encarregado dessa tarefa é a classe `pdf_extract.py`,

ela possui três métodos, dois públicos e um privado. Um método público para cada biblioteca de extração de texto utilizada até o presente momento no trabalho. Enquanto que o método privado foi necessário para encapsular uma pequena lógica que determina o diretório do arquivo no formato PDF que terá seu texto extraído.

O método `get_text_pypdf2` possui como argumento o nome de um arquivo em formato PDF e retorna o texto extraído desse arquivo, para isso utiliza da biblioteca *PyPDF2*. A lógica é simples: primeiro cria-se uma instância de um `PdfFileReader` do arquivo, operação que segundo a documentação pode demorar um pouco, já que a *cross reference table* (estrutura de dados que possui o *byte* inicial de cada objeto dentro do arquivo) é carregada em memória. No *PyPDF2* quem possui a habilidade de extrair o texto do PDF é a classe `PageObject`, que representa uma única página dentro de um arquivo PDF. O acesso a uma instância do `PageObject` é feito através do método `getPage(pageNumber)` do `PdfFileReader`. Com isso, basta iterar sobre todas as páginas do arquivo PDF, e, a cada iteração concatenar o texto extraído na variável que representa o resultado final, que é o texto completo do arquivo PDF. Importante ressaltar que conforme citado pela própria documentação do *PyPDF2*, o funcionamento do método que extrai texto terá um resultado melhor ou pior conforme a ferramenta que o gerou, já que o método fornecido pela biblioteca localiza e retorna todos os comandos de desenho de texto na ordem em que eles são encontrados no *content stream* do arquivo PDF.

```
1  @staticmethod
2  def get_text_pypdf2(filename):
3      path = pdf_extract.__path_from_filename(filename)
4      pdf_reader = PyPDF2.PdfFileReader(path)
5
6      pdf_text = ''
7      for i in range(pdf_reader.numPages):
8          pdf_page = pdf_reader.getPage(i)
9          pdf_text += pdf_page.extractText()
10
11     return pdf_text
```

Já o método `get_text_pytesseract` possui uma *string* que permite a configuração do processo de extração, além do nome do arquivo no formato PDF como argumento. Como esperado, esse método também retorna o texto extraído do arquivo através do uso do *wrapper* do *tesseract* - o *pytesseract*. O processo aqui também é simples: com o auxílio da biblioteca *pdf2image*, cada página do arquivo representado pelo nome recebido como argumento é transformada em uma imagem PIL, da biblioteca *Pillow*, de 500 DPI. Essa lista de imagens então é iterada, cada imagem PIL é passada para a função `image_to_string` que retorna o resultado do processamento *tesseract* OCR sem modificação. Essa função,

além da imagem PIL, recebe a especificação da linguagem português como linguagem para o processamento e uma *string* montada anteriormente no método com possíveis configurações para a biblioteca. Como o *tesseract* leva um tempo considerável para fazer a extração do texto, a configuração `tessedit_do_invert` é sempre definida como `0` por padrão, em uma tentativa de melhoria de desempenho. A configuração dessa forma é recomendada para textos que não estão invertidos (o que se aplica na maior parte das vezes para os relatórios financeiros) e é recomendada pelo próprio *tesseract* como uma das formas de se obter ganho de desempenho. O texto retornado pelo *pytesseract* é sempre concatenado em uma variável que possui o resultado final da extração - retornada no final da execução do método.

```
1  @staticmethod
2  def get_text_pytesseract(filename, custom_config_extact_lib):
3      path = pdf_extract.__path_from_filename(filename)
4      pages = convert_from_path(path, 500)
5
6      config_to_speed_up = '-c tessedit_do_invert=0'
7      config = config_to_speed_up + ' ' + custom_config_extact_lib
8
9      pdf_text = ''
10     for page in pages:
11         pdf_text = pdf_text + ' ' +
               pytesseract.image_to_string(page, lang='por',
               config=config)
12
13     return pdf_text
```

Por fim, no `pdf_extract` temos o `__path_from_filename`, que recebe o nome de um arquivo no formato PDF como argumento e retorna a *string* que representa o diretório onde esse arquivo se encontra no computador. Esse diretório é necessário para as bibliotecas de extração de texto conseguirem desempenhar o seu papel, ou seja, elas só conseguem identificar o texto do arquivo se souberem onde o arquivo está no disco do computador. A identificação do diretório é feita com a assistência da classe `data_dir_scan.py` através do método `get_data_directory`. Essa função verifica em qual diretório os experimentos/testes estão sendo executados, para a partir dele identificar qual o caminho relativo até a pasta `data`, que possui os arquivos em formato PDF.

Para simplificação no processo de definição de diretório, é uma recomendação do presente trabalho que o código fonte esteja em um diretório chamado `tcc`. Além disso, um requisito para o correto funcionamento é que os experimentos sejam executados em

Diretório em que se encontra	**Caminho relativo até diretório *data***
Termina com `data`	`./`
Termina com `tcc`	`data/`
Possui `src` no caminho	*../data/* onde *../* é multiplicado pela quantidade de níveis abaixo da raiz do projeto está o *src*
Possui `tests` no caminho	*../data/* onde *../* é multiplicado pela quantidade de níveis abaixo da raiz do projeto está o *tests*
Outros	Exceção é lançada

Tabela 7 – Funcionamento método *get_data_directory*

ambiente *Linux*. Para validação dessa parte escreveu-se os seguintes testes unitários[6] da classe `data_dir_scan`.

```
1  @staticmethod
2  def get_data_directory():
3      current_directory = os.getcwd()
4      directory_splitted = current_directory.split('/')
5
6      if data_dir_scan.__data == directory_splitted[-1]:  # last
           element from list
7          return './'
8
9      if 'tcc' == directory_splitted[-1]:
10         return data_dir_scan.__data + '/'
11
12     directory_founded = None
13     if data_dir_scan.__src in directory_splitted:
14         directory_founded = data_dir_scan.__src
15     elif data_dir_scan.__tests in current_directory:
16         directory_founded = data_dir_scan.__tests
17
18     if directory_founded is None:
19         raise Exception('Diretório inválido!')
20
21     position = directory_splitted.index(directory_founded)
22     levels_below = len(directory_splitted) - position
23     return ('../' * levels_below) + data_dir_scan.__data + '/'
```

[6] https://github.com/lucasjoao/tcc/blob/master/tests/plataform/test_data_dir_scan.py

Como citado anteriormente, quem organiza toda a execução do trabalho é o `manager`, logo, também é ele quem interage com o `pdf_extract`. Para isso, o `manager` é construído, somente uma vez para cada experimento, com uma lista dos nomes dos arquivos que representam os relatórios financeiros em formato PDF das empresas. Além dessa lista de nomes, pode-se especificar qual a biblioteca de extração de texto deve ser utilizada e se ela terá alguma configuração personalizada. Se nada for especificado para esses itens citados anteriormente, então as opções *default* são utilizadas (*PyPDF2* e sem configuração personalizada).

```
1 class manager:
2
3     def __init__(self, reports_filename,
          text_extract_lib='pypdf2', custom_config_extract_lib=' '):
```

Assinatura do construtor do manager

A interação com o `pdf_extract` é direta. Para cada arquivo na lista de nomes, a função `__get_pdf_text` é chamada no construtor do `manager` e o método adequado no `pdf_extract` é invocado conforme a biblioteca de extração de texto especificada. Detalhes sobre a configuração da biblioteca de extração serão fornecidos posteriormente na seção de experimentos.

```
1 def __get_pdf_text(self, report_filename, text_extract_lib,
     custom_config_extract_lib):
2     pdf_text = ''
3     if text_extract_lib == 'pypdf2':
4         pdf_text =
              pe.pdf_extract.get_text_pypdf2(report_filename)
5     elif text_extract_lib == 'pytesseract':
6         pdf_text =
              pe.pdf_extract.get_text_pytesseract(report_filename,
              custom_config_extract_lib)
7     return pdf_text
```

Método ___get__pdf__text no manager

4.6 Pré-processamento

Já o pré-processamento corresponde ao segundo item, de mesmo nome, do fluxo de trabalho macro (figura 5), enquanto que no fluxo de trabalho detalhado (figura 6) representa do terceiro ao quinto item, que são: ***string* do texto dividida em sentenças**, **sentenças *tokenizadas*** e **remoção de *stopwords***. Todas essas etapas se encontram na

classe `preprocessor.py` e são realizadas pelo método `execute` com a ajuda da biblioteca *nltk*.

```
1  def execute(self, text):
2      text_sentences = sent_tokenize(text, self.__pt_br)
3      text_sentences_in_tokens = [word_tokenize(sentence) for
           sentence in text_sentences]
4
5      stopwords_pt_br = stopwords.words(self.__pt_br)
6      preprocess_result = []
7
8      for sentence in text_sentences_in_tokens:
9          sentence_result = [token for token in sentence if token
               not in stopwords_pt_br]
10         preprocess_result.append(sentence_result)
11
12     return preprocess_result
```

O método `execute` recebe o texto em *string* do arquivo em formato PDF coletado na etapa anterior e retorna uma espécie de matriz (lista de listas). Nessa estrutura, o número de linhas representa o número de sentenças no texto e o número de colunas será a quantidade de *tokens* da maior sentença presente no texto.

Para fazer isso, o método primeiramente se beneficia do `sent_tokenize` da biblioteca *nltk*, essa função recebe o texto e uma especificação para tratar a linguagem do texto em português, e, retorna uma lista de sentenças do texto. Após isso, a lista de sentenças citada anteriormente é iterada e *tokenizada* com o auxílio do `word_tokenize`, ou seja, a lista de sentenças é transformada em uma lista de listas, onde cada item da primeira lista é uma sentença, que agora está representada como uma lista de *tokens*. Logo então é necessário criar uma estrutura que possui todas as palavras vazias de significado do português (`stopwords`), como por exemplo a preposição "de". Essa estrutura é criada a partir do método `words`, da classe `stopwords`, também da biblioteca *nltk*. Por fim, a lista de listas citada anteriormente é iterada novamente, onde para cada item da matriz há uma verificação se o *token* é uma *stopword* ou não, caso seja uma *stopword*, então o *token* não fará parte do resultado final.

Importante citar que a biblioteca *nltk* utiliza de uma ferramenta interna chamada *punkt*[7] para fazer as segmentações citadas até então. Essa ferramenta pode ser treinada, mas no presente trabalho optou-se por utilizar o modelo já pré-treinado para o português disponibilizado pela própria biblioteca. A seguir (figura 10) está um exemplo de uma possível entrada fictícia e sua respectiva saída do método execute da classe `preprocessor`:

[7] https://www.nltk.org/_modules/nltk/tokenize/punkt.html

Figura 10 – Funcionamento *preprocessor* (figura do autor)

De modo similar a extração do texto, o uso do `preprocessor` pelo `manager` é simples e direto. Para cada arquivo na lista de nomes passado na construção do `manager`, realiza-se a extração do texto, que depois de extraído é enviado para o método `execute`, que foi detalhado anteriormente do `preprocessor`.

```
1 pdf_text = self.__get_pdf_text(report_filename,
     text_extract_lib, custom_config_extract_lib)
2 preprocessed_text = preprocessor.execute(pdf_text)
```

4.7 Técnica: *stemming*

O *stemming* é uma técnica de processamento de linguagem natural utilizada no presente trabalho. Essa etapa corresponde ao terceiro item, chamado de **aplicação técnica**, do fluxo de trabalho macro (figura 5), enquanto que no fluxo de trabalho detalhado (figura 6) trata-se do sexto item, chamado de **aplicação *stemming***. No código, a classe `stemming.py` é a responsável pela realização dessa fase com a ajuda da biblioteca *nltk* e através dos métodos `stem_word` e `stem_text_matrix`.

Na instanciação de um objeto da classe `stemming` define-se que o algoritmo utilizado para a realização do *stemming* é o *RSLP Stemmer*, onde RSLP é a sigla para Removedor de Sufixos da Língua Portuguesa. Esse algoritmo foi apresentado no artigo *A Stemming Algorithm for the Portuguese Language*, de 2001, escrito por Moreira Orengo e Christian Huyck, e, possui, segundo o artigo, melhores resultados do que o *stemmer Snowball* escrito por Martin Porter na sua versão em português. Por essa razão e também por ser o algoritmo recomendado pelo *nltk* para o português que se escolheu o RSLP.

```
1  def __init__(self):
2      self.stemmer = RSLPStemmer()
3
4  def stem_word(self, word):
5      return self.stemmer.stem(word)
6
7  def stem_text_matrix(self, text_matrix):
8      """
9      Args
10         text_matrix e o resultado da etapa de pre-processamento
```

```
11      """
12      result = []
13      for sentence in text_matrix:
14          sentence_result = [self.stem_word(token) for token in
                sentence]
15          result.append(sentence_result)
16
17      return result
```

O método `stem_word` é quem interage com o algoritmo de *stemmer*. Ele faz isso através da chamada do método `stem` da biblioteca. Ambos os métodos, o da biblioteca e o do presente trabalho, recebem uma *string* que representa a palavra que terá a técnica aplicada e retornam o *stem* dessa palavra. A função `stem_word` é um método por si só e é pública porque ela também é utilizada na parte de indicadores financeiros - que corresponde a um trecho de código que será detalhado mais à frente no trabalho. Optou-se por isolar a chamada da função da biblioteca de *stemming* como uma forma de abstração que garante facilidade para possíveis futuras alterações.

Já o `stem_text_matrix` aplica a técnica de *stemming* em uma estrutura de dados diferente, pois possui como argumento a matriz (lista de listas) resultante da etapa de pré-processamento. Ele itera sobre cada item dela e chama o método, já apresentado, `stem_word` para executar a técnica, onde o resultado de cada iteração é adicionado em uma nova lista de listas - que representa o retorno do método. Por exemplo, se considerar a continuação do exemplo fictício (figura 10) apresentado anteriormente na seção de pré-processamento, o resultado após a passagem daquele trecho de texto pelo *stemming* seria (figura 11):

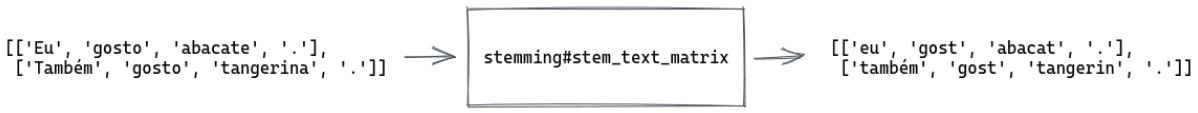

Figura 11 – Funcionamento *stem_text_matrix* (figura do autor)

Na mesma linha das seções anteriores, a interação do `manager` com o `stemming` é trivial. O resultado do pré-processamento é passado para o `stem_text_matrix`, e, o resultado é armazenado em um dicionário. Esse dicionário é uma propriedade (`reports_stemming`) global do `manager`, que é dessa forma para futuro acesso durante a execução. O `reports_stemming` possui como chave o nome do arquivo no formato PDF e como valor a matriz que é o texto *tokenizado* do arquivo com a técnica de *stemming* aplicada. A imagem a seguir apresenta o trecho de código que mostra a criação do `reports_stemming`, esse trecho é o último código executado no construtor do manager - antes dele, o construtor somente cria

atributos privados para a classe.

```
self.reports_stemming = {}
for report_filename in reports_filename:
    pdf_text = self.__get_pdf_text(report_filename,
       text_extract_lib, custom_config_extract_lib)
    preprocessed_text = preprocessor.execute(pdf_text)
    self.reports_stemming[report_filename] =
       stemming.stem_text_matrix(preprocessed_text)
```

4.8 Indicadores financeiros

A parte referente aos indicadores financeiros não corresponde a nenhuma etapa no fluxo de trabalho, já que ela serve principalmente para definir o domínio de interesse da busca que será realizada durante a filtragem na execução. No código, esse domínio é especificado no diretório `src/indicator`, onde cada indicador trabalhado possui uma classe em um arquivo. Logo, há três arquivos: `lucro_liquido.py`, `patrimonio_liquido.py` e `roe.py`.

```
def __init__(self):
    self.stemmer = s.stemming()

def get_target_sets(self):
    return [frozenset([self.stemmer.stem_word('lucro'),
       self.stemmer.stem_word('liquido')])]

def get_filter_sets(self):
    return [frozenset([self.stemmer.stem_word('imposto'),
       self.stemmer.stem_word('renda')]),
            frozenset([self.stemmer.stem_word('receita'),
               self.stemmer.stem_word('operacional'),
                       self.stemmer.stem_word('liquida')]),
            frozenset([self.stemmer.stem_word('wege3')]),
            frozenset([self.stemmer.stem_word('ativos'),
               self.stemmer.stem_word('fixos')]),
            frozenset([self.stemmer.stem_word('investimentos')]),
            frozenset([self.stemmer.stem_word('variacao')]),
            frozenset([self.stemmer.stem_word('dividendos')]),
            frozenset([self.stemmer.stem_word('dívida')]),
            frozenset([self.stemmer.stem_word('imposto')]),
            frozenset([self.stemmer.stem_word('recuperacao')])]
```

Conteúdo do arquivo que representa o indicador financeiro lucro líquido

Durante a execução do trabalho percebeu-se que um indicador financeiro pode ser categorizado conforme sua disponibilidade nos relatórios financeiros concedidos pelas empresas. Na amostragem utilizada, todos os relatórios apresentam o lucro líquido e o patrimônio líquido trimestral da empresa, ou seja, categoria dos indicadores presentes. Por outro lado, somente 3 dos 13 relatórios que compõem o *dataset* apresentam o ROE, portanto, ele corresponde a categoria do indicador não presente, que precisa ser calculado. Essa categorização implica em como a classe que representa o domínio do indicador financeiro será estruturada.

A estruturação na classe é a seguinte:

Categoria do indicador financeiro	**Regras para serem seguidas**
Indicador sempre presente	- Criação de um *stemmer* no construtor para uso posterior; - Método *get_target_sets* - Método *get_filter_sets*
Indicador que necessita ser calculado	- Regras especificadas para a categoria anterior; - Método *calculate* - Método *calculate_iterating*

Tabela 8 – Estruturação das classes de indicadores financeiros

As regras para essa estruturação poderiam ser representadas através de uma abstração via código, mas esse aspecto fica como melhoria futura do presente trabalho. Vale citar que o método `get_target_sets` deve retornar uma lista de conjuntos imutáveis (por isso a escolha do `frozenset`), onde cada conjunto possui, com o *stemming* aplicado em cada elemento, uma maneira de se referir/chamar o indicador financeiro, que será utilizada para a busca no relatório financeiro. Enquanto isso, o `get_filter_sets` também deve retornar o mesmo tipo de estrutura que o `get_target_sets`, porém aqui o significado é diferente, são valores que quando presentes na mesma sentença que o indicador financeiro, devem invalidar a possibilidade da sentença conter um resultado - esses valores foram encontrados a partir de diversas execuções dos experimentos. Enquanto o `calculate` recebe os argumentos necessários para o cálculo do indicador financeiro, quantidade que varia conforme o indicador financeiro, e retorna o indicador financeiro calculado a partir dos argumentos. Para o indicador ROE há um `FIXME`, que é também outra melhoria futura, da necessidade de se considerar a escala dos valores ao realizar o cálculo. Por fim, o `calculate_iterating` é uma forma de abstrair a chamada do `calculate` para um grupo

de valores e assim retornar uma lista de resultados.

Aqui também o papel do `manager` ao conversar com os indicadores financeiros não é complexa. Ao construir uma instância do `manager`, também será criado um objeto de cada classe de indicador financeiro. Isso é realizado dessa forma para que essas propriedades possam ser utilizadas pelo `manager` na "execução", ou seja, quando o mesmo é solicitado que se busque uma informação. Esse trecho que demonstra o que o `manager` faz durante a "execução" será detalhado na próxima seção do trabalho.

```
1  def __init__(self):
2      self.stemmer = s.stemming()
3
4  def get_target_sets(self):
5      return [frozenset(['roe']),
6              frozenset([self.stemmer.stem_word('retorno'),
                   self.stemmer.stem_word('sobre'),
7                          self.stemmer.stem_word('patriminio'),
                              self.stemmer.stem_word('liquido')])]
8
9  def get_filter_sets(self):
10     return [frozenset([])]
11
12 def calculate(self, lucro_liquido, patrimonio_liquido):
13     # FIXME: considerar escala na hora de calcular
14     return (lucro_liquido / patrimonio_liquido) * 100
15
16 def calculate_iterating(self, lucro_liquido_result_list,
       patrimonio_liquido_result_list):
17     result = []
18     for lucro_liquido in lucro_liquido_result_list:
19         for patrimonio_liquido in patrimonio_liquido_result_list:
20             result.append(self.calculate(lucro_liquido['number'],
                   patrimonio_liquido['number']))
21     return result
```

Conteúdo do arquivo que representa o indicador financeiro ROE

4.9 Execução

O item com nome de execução do presente trabalho corresponde ao código que é invocado diretamente pelos experimentos. Esse código utiliza de tudo que já foi apresentado até então para buscar em uma forma específica o indicador financeiro solicitado no relatório

com os resultados financeiros da empresa. A execução é dividida em duas grandes fases: a filtragem e as buscas possíveis. Ambas serão detalhadas posteriormente.

Enquanto isso, no `manager` essa interface que está visível para ser chamada pelos experimentos corresponde aos seguintes métodos - que possuem nomes autoexplicativos:

- `run_lucro_liquido_monetary`
- `run_lucro_liquido_number`
- `run_patrimonio_liquido_monetary`
- `run_patrimonio_liquido_number`
- `run_roe_monetary`
- `run_roe_number`
- `run_calculate_roe`

4.9.1 Filtragem

A fase de filtragem, uma parte da execução, corresponde ao quarto item, de mesmo nome, do fluxo de trabalho macro (figura 5), enquanto que no fluxo de trabalho detalhado (figura 6) representa do sétimo ao nono item: **sentenças filtradas por domínio de interesse**, **remoção de possíveis falso-positivos** e **verificação da ordem dos itens do domínio de interesse**. No código, a classe `filters.py` é quem realiza essa etapa por meio dos métodos `candidate_sentences` e `is_searcher_words_in_sequence`.

A função `candidate_sentences` retorna uma matriz, lista de listas, onde cada linha dessa matriz é uma sentença proveniente do argumento `text` que possui todos os elementos de pelo menos um conjunto pertencente ao outro argumento, a lista de conjuntos `target_sets`. Esse resultado é obtido da seguinte forma, a matriz que representa o texto é iterada, onde para cada elemento se visita também a lista de conjuntos alvos, com isso, se o conjunto alvo não for vazio ao mesmo tempo que é um subconjunto do conjunto que representa a sentença atual da iteração do texto, então essa sentença atual é uma possível candidata que estará no retorno da função.

```
1 def candidate_sentences(self, text, target_sets):
2     candidates = []
3     for sentence in text:
4         for target_set in target_sets:
5             empty_target_set = len(target_set) == 0
6             if not empty_target_set and
                 target_set.issubset(frozenset(sentence)):
```

```
                candidates.append(sentence)
    return candidates
```

Já o método `is_searcher_words_in_sequence` é um pouco mais complexo, mas possui o simples objetivo de retirar sentenças candidatas oriundas de um processamento a partir do `candidate_sentences` que na verdade são falso-positivos. O método retorna uma lista de sentenças candidatas sem os falso-positivos citados anteriormente, e, possui dois argumentos: `candidates`, uma matriz onde cada linha é uma sentença candidata, e, `target_sets`, uma lista de conjuntos que representa o alvo da busca para verificação de falso positivo. Para alcançar seu objetivo ele itera a matriz, onde para cada linha é feita a iteração sobre a lista de conjuntos alvos. Logo após, descobre-se em quais posições da sentença encontra-se o primeiro item do conjunto alvo, pois assim é possível verificar se antes ou depois dessa posição na sentença estão os outros itens pertencentes ao conjunto alvo. Portanto, se esse pertencimento for comprovado, então não é um falso positivo.

```
def is_searcher_words_in_sequence(self, candidates, target_sets):
    candidates_filtered = []
    for sentence in candidates:
        for target_set in target_sets:
            first_searcher_element, *_ = target_set
            target_set_size = len(target_set)
            positions = [i for i, token in enumerate(sentence)
               if token == first_searcher_element]

            for position in positions:
                hits = 1
                for i in range(1, target_set_size):
                    try:
                        is_hit = sentence[position + i] in
                           target_set or sentence[position - i]
                           in target_set
                    except IndexError:
                        is_hit = False
                    if is_hit:
                        hits += 1

                if target_set_size == hits:
                    candidates_filtered.append(sentence)
    return lh.list_helper
        .remove_duplicates_list_of_lists(candidates_filtered)
```

Teste unitários[8] foram codificados para ambos os métodos. Para um melhor entendimento, segue uma exemplificação fictícia na mesma linha das apresentadas anteriormente de ambos os métodos:

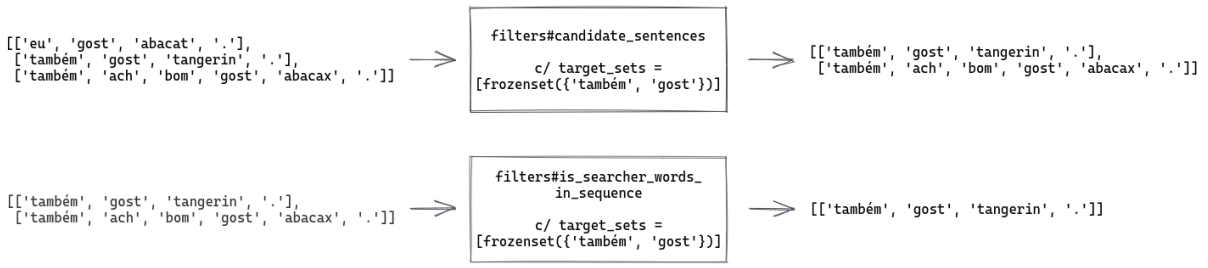

Figura 12 – Funcionamento *filters.py* (figura do autor)

Importante citar que a filtragem não tinha sido idealizada inicialmente em uma etapa separada, como pode ser visto pela seção anterior (Histórico de tentativas). A necessidade de estruturar o processo dessa forma surgiu ao decorrer do trabalho com a experimentação e a ocorrência de muitos falso-positivos que foram então removidos através de algumas filtragens.

```
1  def __common_process(self, indicator):
2      target_sets = indicator.get_target_sets()
3
4      result = {}
5      for filename, stem_text_matrix in
           self.reports_stemming.items():
6          candidate_sentences =
               self.filters.candidate_sentences(stem_text_matrix,
               target_sets)
7          false_candidate_sentences =
               self.filters.candidate_sentences(candidate_sentences,
               indicator.get_filter_sets())
8          candidate_sentences = [sentence for sentence in
               candidate_sentences if sentence not in
               false_candidate_sentences]
9          candidate_sentences =
               self.filters.is_searcher_words_in_sequence(
10         candidate_sentences, target_sets)
11         result[filename] = candidate_sentences
12     return result
```

[8] https://github.com/lucasjoao/tcc/blob/master/tests/plataform/test_filters.py

Por fim, o uso que o `manager` faz da classe `filters` é onde realmente os filtros são aplicados para remoção de falso-positivos. Essa interação acontece no método privado do `manager` chamado de `__common_process`, que é utilizado por todos os métodos disponíveis de execução. O método recebe um indicador financeiro como argumento (`indicator`), que por sua vez possui os conjuntos alvos para a busca do indicador financeiro. Com isso, para cada relatório financeiro presente na propriedade `reports_stemming` há um processamento que consiste das seguintes etapas:

1. filtrar todas as possíveis sentenças candidatas de se ter o indicador financeiro a partir do texto completo do relatório;
2. filtrar todas as possíveis sentenças candidatas que são falso positivos, conhecidos através de experimentação prévia, do resultado do item anterior;
3. filtrar das possíveis sentenças candidatas, provenientes do item anterior, as sentenças que não possuem o conjunto alvo na sequência correta;
4. matriz com as possíveis sentenças candidatas de possuírem o indicador financeiro pronta para ser armazenada na variável que será retornada pelo método.

4.9.2 Buscas possíveis

Já a etapa de buscas possíveis, a outra parte da execução, corresponde ao quinto e último item, chamado **busca**, do fluxo de trabalho macro (figura 5), enquanto que no fluxo de trabalho detalhado (figura 6) representa todos os itens presentes nas duas ramificações finais:

1. **Busca por valor monetário + remoção de possíveis falso-positivos + remoção de resultados duplicados**
2. **Busca por valor numérico + remoção de resultados duplicados**

A classe `searcher.py` é a responsável no código por realizar essas atividades e faz isso com a ajuda dos métodos `monetary_value` e `after_target_set_number_value`. Assim, é possível concluir que há duas buscas possíveis no presente trabalho, uma busca por valores monetários e outra busca por valores numéricos.

A pesquisa por valores monetários é realizada pelo método `monetary_value`. A função recebe como argumento a matriz já citada no decorrer das seções anteriores, onde cada linha representa uma sentença e as colunas representam os *tokens* (cada palavra) da sentença. A partir desse valor recebido por argumento, itera-se sobre cada item da matriz e uma verificação é realizada se esse item é um número ou não. Essa verificação é feita

através de uma função auxiliar, `is_number` da `classe number_helper`. Se esse valor é um número, então ele pode ser um valor monetário. Para saber se é um valor monetário, há uma checagem dos dois itens anteriores ao número encontrado, se ambos forem iguais a `R` e `$`, ou seja, o símbolo que representa o Real, então trata-se de um valor financeiro. Por fim, se tudo indicar que é um valor monetário, então ele é adicionado em uma lista de resultados, que o armazena em um formato específico de dicionário criado pela própria classe `searcher`.

Alguns aspectos importantes dessa função são que seu funcionamento é totalmente atrelado à moeda brasileira. Além disso, no histórico de construção do trabalho, essa foi a primeira busca codificada após o processo de tentativas inicial, pois acreditava-se muito no seu potencial de sucesso - o que se mostrou parcialmente verdade. Por último, notou-se por experimentação que esse tipo de busca possui uma taxa de sucesso maior em textos formatados em parágrafos, ao invés de tabelas nos relatórios de indicadores financeiros.

```
1  def monetary_value(self, candidate_sentences):
2      invalid_positions = frozenset([0, 1])
3
4      results = []
5      for sentence in candidate_sentences:
6          position = 0
7          for token in sentence:
8              was_casted, number =
                   nh.number_helper.is_number(token)
9              if was_casted and position not in invalid_positions:
10                 if sentence[position - 2] == 'r' and
                       sentence[position - 1] == '$':
11                     results.append(self.__safe_create_result_item(
12                     number, sentence, position))
13
14             position += 1
15
16     return results
```

No intuito de gerar uma compreensão melhor, segue um exemplo fictício da execução do método `monetary_value`:

Figura 13 – Funcionamento método *monetary_value* (figura do autor)

O método `after_target_set_number_value` executa a pesquisa por valor numérico. Esse método possui dois argumentos, a matriz `candidate_sentences` que representa o texto (no mesmo formato que o outro método da classe `searcher` recebe) e o `target_sets`, uma lista de conjuntos que possuem as palavras que representam o indicador financeiro. A ideia desse método é buscar valores numéricos que estão logo após as palavras que representam o indicador financeiro. Isso é feito da seguinte forma: itera-se cada *token* de cada sentença do `candidate_sentences` para cada conjunto recebido, e, caso o *token* seja um valor numérico, então há uma verificação dos *tokens* anteriores - eles devem estar no conjunto representativo do indicador financeiro. Se isso tudo acontecer, então trata-se de um resultado possível, logo um dicionário é criado com esse dado, no mesmo formato específico dessa classe comentado anteriormente, para que seja adicionado na lista de resultados finais.

Os aspectos importantes dessa rotina são que a verificação para saber se é um número também utiliza do `number_helper.is_number`. Também trata-se da evolução da primeira busca codificada na seção anterior (Histórico de tentativas), e por fim, ela mostrou obter resultados melhores ao trabalhar com dados em tabelas nos relatório de resultados financeiros que compõem o *dataset*.

```
1  def after_target_set_number_value(self, candidate_sentences,
       target_sets):
2      results = []
3      for target_set in target_sets:
4          target_set_size = len(target_set)
5          for sentence in candidate_sentences:
6              curr_position = 0
7              for token in sentence:
8                  possible_result = True
9
10                 was_casted, number =
                       nh.number_helper.is_number(token)
11                 if was_casted:
12
13                     for i in range(1, target_set_size + 1):
14                         lookup_position = curr_position - i
15                         if lookup_position < 0 or
                               sentence[lookup_position] not in
                               target_set:
16                             possible_result = False
17                             break
18
19                     if possible_result:
```

```
20                          results.append(
21                          self.__safe_create_result_item(number,
                               sentence, curr_position))
22
23                  curr_position += 1
24
25      return results
```

Na linha de se ter uma melhor compreensão da função, segue um exemplo fictício da utilização do `after_target_set_number_value`:

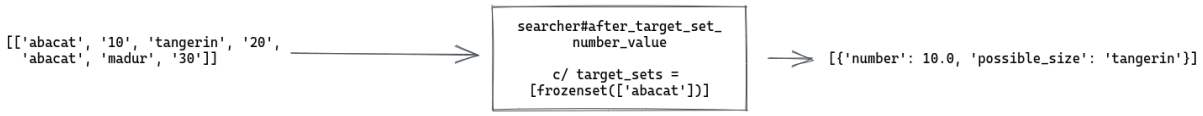

Figura 14 – Funcionamento método *after_target_set_number_value* (figura do autor)

Aqui também testes unitários[9] com os principais cenários foram codificados para ambos os métodos. A classe `searcher` utiliza de duas outras classes *helper* para executar o seu trabalho:

- o `number_helper`, já citado e que serve para verificar se um texto em *string* é um número através do método `is_number`. Esse método recebe a *string* com o texto e retorna uma tupla, onde o primeiro valor indica se o valor recebido como argumento foi transformado em número, ou seja, se o *token* era um número, e, o segundo valor da tupla é o número propriamente dito ou um `NaN` (*not a number*) - dependendo do resultado. Definir se uma *string* representa um número foi uma situação simples que se tornou incômoda durante a execução do trabalho devido à falta de padronização nos relatórios financeiros, e, por isso uma extensa bateria de testes[10] foi criada para esse *helper*, que garante que todos os valores exemplificados a seguir em *string* serão considerados números:
  - 100
  - 1.234,50
  - 1234.12
  - 1234.0
  - 1.234.567
  - 1.234.567.89

[9] https://github.com/lucasjoao/tcc/blob/master/tests/plataform/test_searcher.py
[10] https://github.com/lucasjoao/tcc/blob/master/tests/helper/test_number_helper.py

– 1.234,5

- o `result_helper`, que a partir do do método interno `__safe_create_result_item` é invocado, possui a função de criar o resultado da busca em um formato específico. Esse formato é um dicionário, onde uma chave é o `number` que aponta para o valor encontrado e a outra chave é o `possible_size`, que é uma *string* que pode representar a magnitude do valor numérico encontrado (e.g. milhões). Inicialmente, a magnitude é de maneira simplória o *token* posterior ao valor numérico encontrado - quando esse *token* não existe ou é um número, então a constante `undefined` é atribuída para o `possible_size`. Essa solução foi codificada dessa maneira para ir de encontro com a necessidade de se saber a escala do valor numérico, pois dessa forma seria possível calcular o indicador financeiro na escala correta, ou, ao menos informar o usuário em qual escala o indicador financeiro calculado se encontra. Todavia, como já citado anteriormente, com o decorrer das execuções dos experimentos percebeu-se que o procedimento dessa forma nem sempre estava correto e por isso não houve muitos avanços na utilização do mesmo.

```
1 def __safe_create_result_item(self, number, sentence, position):
2     result_item = {}
3     try:
4         result_item =
              rh.result_helper.create_result_item(number,
              sentence[position + 1])
5     except IndexError:
6         result_item =
              rh.result_helper.create_result_item(number,
              'undefined')
7     return result_item
```

Na interação com o `manager` é onde se percebe melhor no código as ramificações citadas no começo dessa seção, já que a primeira ramificação é representada pelo método `__common_process_monetary`, enquanto que a segunda ramificação é representada pelo método `__common_process_number`. Ambos os métodos são similares entre si, recebem o mesmo argumento, que é o indicador financeiro, chamam o `__common_process` (explicado no fim da seção de filtragem) e iteram sobre cada relatório financeiro que o manager está trabalhando. Para cada relatório, a busca específica da ramificação é realizada e resultados duplicados são removidos do resultado final antes do retorno - através da função `remove_duplicates_list_of_dicts`. A única diferença no fluxo está que a busca financeira há um processo extra de limpeza do resultado após a realização da busca, pois por conhecimento de observação adquirido durante a execução do trabalho, se percebeu

que quando na busca financeira o valor encontrado possuía um `possible_size` com o valor `undefined`, então não era o valor numérico esperado.

```
1  def __common_process_monetary(self, indicator):
2      reports_candidate_sentences =
           self.__common_process(indicator)
3
4      result = {}
5      for filename, candidate_sentences in
           reports_candidate_sentences.items():
6          dirty_result =
               self.searcher.monetary_value(candidate_sentences)
7          result_with_duplicate =
               rh.result_helper.clean_search_result(dirty_result)
8          result[filename] =
               lh.list_helper.remove_duplicates_list_of_dicts(
9          result_with_duplicate)
10     return result
11
12 def __common_process_number(self, indicator):
13     reports_candidate_sentences =
           self.__common_process(indicator)
14
15     target_sets = indicator.get_target_sets()
16
17     result = {}
18     for filename, candidate_sentences in
           reports_candidate_sentences.items():
19         result_with_duplicate =
               self.searcher.after_target_set_number_value(
20         candidate_sentences, target_sets)
21         result[filename] =
               lh.list_helper.remove_duplicates_list_of_dicts(
22         result_with_duplicate)
23     return result
```

5 Avaliação do modelo

5.1 Métricas utilizadas para avaliação

Para possibilitar uma melhor avaliação do modelo, algumas métricas serão obtidas a partir da execução dos experimentos. Dessa forma, é possível mensurar a qualidade dos resultados obtidos e também fica mais fácil de comparar quantitativamente um experimento com o outro. Essa subseção possui como objetivo descrever as métricas utilizadas e por qual motivo as mesmas foram escolhidas. As seguintes métricas serão discutidas: matriz de confusão; acurácia; *recall*; precisão; *F1-score*; resultados que consideram unicidade; tempo.

A primeira métrica não é bem uma métrica, mas mais uma ferramenta que irá ajudar no cálculo de outras métricas - trata-se da matriz de confusão. Frequentemente utilizada nas avaliações de modelos de classificação e predição, além de ajudar no cálculo de outras métricas, ela também permite uma visualização melhor do desempenho do algoritmo. No presente trabalho, apesar do algoritmo proposto no modelo realizar principalmente uma busca com um cálculo, ainda assim foi possível calcular uma matriz de confusão com o intuito de usufruir do que é possível com ela - auxílio no cálculo de métricas e verificação do desempenho.

Com a ajuda da matriz de confusão é possível calcular uma nova métrica, a acurácia de cada experimento. Pode-se dizer que a acurácia é a fração de predições que um modelo fez corretamente. No modelo da presente pesquisa indica então a porcentagem de busca com cálculo realizada corretamente por cada experimento. Dessa forma, quanto maior a acurácia do experimento, melhor ele realiza o objetivo geral do trabalho. A fórmula da acurácia é $(TP + TN)/T$, onde TP são os verdadeiros positivos, TN os verdadeiros negativos e T o total de resultados nos valores coletados com o trabalho. Contudo, é importante notar que a acurácia aqui não pode ser usada sozinha para fazer a avaliação dos resultados, pois ao observar somente ela não é possível fazer uma distinção entre a quantidade de vezes que o experimento trouxe o indicador financeiro do relatório quando ele existia lá com a quantidade de vezes que o experimento não trouxe nenhuma informação quando o relatório não possuía a informação.

Ainda com o auxílio da matriz de confusão consegue-se calcular outra métrica, o *recall*. Com o *recall* é possível saber o quão bom um modelo consegue prever positivos, ou seja, quão bom ele consegue prever o que se deseja prever no modelo. Logo, no trabalho desenvolvido esse valor irá indicar o quanto um experimento conseguiu buscar e calcular o indicador financeiro desejado. Sendo assim, mais uma vez, quanto maior o *recall*, melhor o experimento. O *recall* possui a fórmula $TP/(TP + FN)$, onde TP são os verdadeiros

positivos e FN os falsos negativos nos resultados do trabalho. Interessante perceber que o cálculo dessa métrica serve para resolver o ponto fraco citado anteriormente da acurácia, pois ao utilizar o *recall* junto com a acurácia consegue-se tirar conclusões melhores do comparativo entre os experimentos.

A última métrica calculada a partir da matriz de confusão é a precisão. A precisão indica a proporção de identificações positivas no modelo que estão de fato certas, ou pode-se dizer de modo informal, o quão bem o modelo trabalhou quando trouxe dados positivos. Na presente pesquisa isso indica quantas vezes a busca com cálculo só trouxe resultados incorretos para um determinado indicador financeiro no relatório empresarial, ou seja, a confiabilidade do resultado obtido pelo experimento. Novamente, quanto mais alto esse valor, melhor o resultado do experimento. A sua fórmula é $TP/(TP + FP)$, onde TP são os verdadeiros positivos e FP os falsos positivos do resultado da pesquisa. Além do valor obtido ao analisar essa métrica separadamente, vale citar que o cálculo da precisão também foi motivado devido ao fato de ser necessária para o cálculo da métrica que será discutida em seguida.

A próxima métrica calculada se chama *F1-score*. Ela é obtida a partir da média harmônica de outras métricas já apresentadas, o *recall* e a precisão, onde se dá o mesmo peso para cada uma das duas. Dessa forma, o mesmo grau de importância é dado ao *recall* e a precisão, e, somente se ambos forem bons, então o *F1-score* será alto, em outras palavras, um bom valor também. No atual trabalho, a motivação por trás do seu cálculo foi que o *F1-score* seria uma das principais métricas para determinar se um experimento tinha sido melhor que o outro ao realizar a busca com cálculo dos indicadores financeiros.

Haverá também a apresentação dos resultados considerando a unicidade, que serve para mostrar as respostas dadas pelos experimentos com uma outra visão, pois ao considerar a unicidade deseja-se saber quais valores o experimento entrega de resultado quando se deseja que a busca traga somente um valor possível para o indicador financeiro - diferente da premissa utilizada para calcular todas as métricas até então. De forma qualitativa, pode-se dizer que esse resultado é mais rígido que todas as métricas apresentadas até então, principalmente se considerar essa mudança de fator como uma possível necessidade ao em um trabalho futuro realizar a adaptação do estudo para uma plataforma completamente funcional onde se desejaria que o algoritmo de busca retorne somente um resultado ao informar um indicador financeiro de um determinado relatório empresarial.

Por fim, o tempo gasto em cada experimento também foi mensurado e apresentado. Já que a partir da associação desse dado com outras métricas pode-se fazer questionamentos interessantes, como: tudo bem levar muito mais tempo para se obter um resultado melhor para se alcançar os objetivos? Há uma associação entre a qualidade do resultado e o tempo levado para o experimento ser executado? Ou seja, um resultado melhor necessariamente será mais rápido ou mais demorado?

5.2 Experimentos

Os experimentos no presente trabalho foram realizados através de testes unitários. Cada conjunto de testes representa uma definição de configuração para a plataforma, ou seja, uma possível experimentação de determinada especificação que possivelmente impacta nos resultados esperados. Ao serem realizadas as experimentações, algumas métricas foram coletadas e também verificou-se a corretude do resultado obtido através de comparação com uma tabela verdade dos resultados.

Todos os experimentos fazem parte do entregável final do trabalho e estão presentes no diretório `tests/reports`. Cada diretório dentro do `reports` representa um conjunto de testes (um tipo de configuração para a plataforma). Por sua vez, cada conjunto de teste possui uma classe de teste unitário para cada relatório utilizado do `dataset`. Logo, em um conjunto de testes, todos os relatórios que compõem o `dataset` devem ser verificados através da configuração do momento da plataforma.

```
1  class TestsAmbev20191T(unittest.TestCase):
2
3      filename = None
4      manager_pytesseract = None
5
6      @classmethod
7      def setUpClass(cls):
8          cls.filename = 'ambev_2019_1T.pdf'
9          cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract')
10
11     @classmethod
12     def tearDownClass(cls):
13         cls.filename = None
14         cls.manager_pytesseract = None
15
16     def test_lucro_liquido_monetary(self):
17         lucro_liquido_monetary_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_monetary()
18         result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
19         numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(
20              result_pytesseract)
21
22         self.assertEqual(len(result_pytesseract), 4, 'lucro
               liquido (R$): tamanho resultado (pytesseract)')
```

```
23          self.assertNotIn(data.LUCRO_LIQUIDO[self.filename],
               numbers_from_result_pytesseract,
24                          'lucro liquido (R$): valor
                               (pytesseract)')
```

Parte de uma classe de teste para exemplificação

O código anterior mostra parte de uma classe de teste unitário de um relatório financeiro (nesse caso da empresa *Ambev*, do primeiro trimestre de 2019). A estrutura dessa classe de teste é replicada em todas as outras e consiste nos seguintes tópicos:

- Um método da classe que é executado antes de todos os testes do arquivo, o `setUpClass`, que define o nome do arquivo do relatório financeiro e também o `manager` que será utilizado em todos os testes da classe;
- Um método da classe que é executado após todos os testes do arquivo, o `tearDownClass`, que apaga as definições citadas no item anterior;
- Um teste unitário para cada associação de um indicador financeiro com um tipo de busca possível. Se o indicador financeiro for do tipo que é possível de ser calculado, então um teste para esse cálculo também. Cada teste unitário é simples e direto:
  - Uma chamada ao método adequado do `manager`, que traz o resultado da busca;
  - Se necessário, uma transformação no resultado é aplicada com o auxílio do `result_helper`;
  - Por último os *asserts* com base nos valores obtidos com aquela configuração para o relatório financeiro.

Parte dos experimentos entregues foram realizados evolutivamente com o decorrer do desenvolvimento do trabalho, isso quer dizer que não se esperou todo o código estar finalizado para começar a escrita do conjunto de testes, pois dessa forma foi mais fácil identificar erros e melhorias que somente um material de "produção" (o relatório de indicador financeiro) consegue fornecer. Logo, cada alteração exigia a reexecução de todos os experimentos para verificar se um impacto indesejado não tinha sido gerado. A partir do momento que o código da plataforma encontrou-se estável, o fluxo de trabalho da experimentação citado anteriormente (figura 7) foi seguido à risca.

Devido ao fato que um conjunto de testes (experimento) pode ser executado diversas vezes, optou-se por seguir uma estratégia em que os *assert* sempre devem passar se o comportamento da plataforma não tiver tido alteração. Para exemplificar isso, vamos considerar um teste unitário que realiza uma busca de um valor monetário do indicador financeiro Lucro líquido em algum relatório financeiro e essa busca retorna cinco possíveis

resultados, sendo que nenhum deles é o resultado esperado. Nessa estratégia, deve-se ter um *assert* que verifica que a busca retorna cinco itens, e, outro *assert* que verifica que nenhum item proveniente da busca é o resultado esperado - dessa forma, se garante que alterações realizadas na plataforma que impactam em métricas importantes do experimento conseguirão ser identificadas e analisadas. Além disso, todas as verificações são compiladas em uma tabela de resultados.

No sentido de proporcionar reprodutibilidade e comparação, o *hardware* em que os experimentos foram executados foi o seguinte:

- Sistema operacional: manjaro
- Processador: Intel Core i5-7500 3.40GHz
- Memória RAM: 16GB DDR4 2400MHz
- Placa de vídeo: GeForce GTX 1060 6GB
- Armazenamento: SSD Kingston 480GB

Para facilitar a execução dos experimentos e também dos diferentes testes unitários, um `makefile` foi criado com diversas regras que possibilitam a inicialização da execução de todos os testes (experimentos e unitários), de grupos de testes unitários específicos, de todos os experimentos sequencialmente, ou por fim, de cada grupo de experimento existente. Importante notar então que sempre que um novo grupo de experimento é adicionado, necessita-se adicionar uma nova regra para esse grupo no `makefile`. Além disso, com esse `makefile` também é possível executar rascunhos do projeto que estão codificados no `main` ou fazer a verificação da padronização do código com o *linter*.

A definição dos experimentos, da mesma forma que a realização, foi feita evolutivamente, já que inicialmente não se tinha todas as configurações de experimentos que poderiam ser executadas definidas. O processo de pesquisa, codificação da plataforma e dos rascunhos geraram insumos para o estabelecimento de todos os cenários utilizados até então e também para um ponto importante do presente trabalho, que é que a diferenciação entre os experimentos se daria principalmente através da mudança do modo de se extrair as informações dos relatórios financeiros presentes no *dataset*.

Já a fixação do ROE (Retorno sobre o patrimônio líquido) como indicador financeiro aconteceu após a primeira tentativa de se trabalhar com outro indicador, o ROI (Retorno sobre o investimento), tentativa ruim já que ele é um indicador praticamente nunca presente (informação constatada por experimentação), a fórmula do seu cálculo não ser muito precisa para uma pessoa não *expert* em contabilidade e os valores necessários para realizá-la estarem presentes nos relatórios financeiros com diferentes nomenclaturas.

Diferente do ROI, o ROE possui uma presença, apesar de baixa, maior em relatórios financeiros (informação também verificada empiricamente), e por outro lado, possui um modo de calcular mais claro e com variáveis sem ambiguidade na nomenclatura - fato que facilita muito a busca.

A seguir segue uma tabela que detalha cada experimento (conjunto de teste) executado:

Nome experimento	**Configuração relevante utilizada**
Extração com *PyPDF2* #0	*out of the box* da versão utilizada
Extração com *pytesseract* #0	*pytesseract* com parâmetro *tessedit_do_invert* = 0
Extração com *pytesseract* #1	*pytesseract* com parâmetros *tessedit_do_invert* = 0, *psm* = 10 e *oem* = 2
Extração com *pytesseract* #2	*pytesseract* com parâmetros *tessedit_do_invert* = 0 e *psm* = 6
Extração com *pytesseract* #3	*pytesseract* com parâmetros *tessedit_do_invert* = 0 e *psm* = 5

Tabela 9 – Especificação dos experimentos realizados

Como é possível perceber, de todos os experimentos, há apenas um com o *PyPDF2*, isso acontece porque não encontrou-se nenhuma forma de customizar a forma de extração realizada pela biblioteca. Aliás, a própria documentação da mesma fala que o método responsável por realizar a extração de texto só funciona bem para alguns arquivos em formato PDF, enquanto que pode funcionar muito mal em outros arquivos - comportamento dependente do modo que o PDF tenha sido gerado. Conforme já foi dito anteriormente, essa foi a principal razão para ao decorrer do desenvolvimento experimentar-se outra biblioteca para essa finalidade, ou seja, o *tesseract* através do *pytesseract*.

Todavia, apesar do *tesseract* permitir uma gama de configurações, a sua versão *out of the box* é lenta. Como citado anteriormente na seção de extração do texto, essa é a razão para o `tessedit_do_invert` ser sempre definido como `0`, na tentativa de melhora de desempenho. A melhora de desempenho ocorreu, mas não de forma drástica. A versão *out of the box* no experimento **Extração com pytesseract #0** levou uma vez 1376 segundos para ser executada (aproximadamente 23 minutos), enquanto que a versão com a tentativa de melhoria de desempenho para o mesmo experimento executou em 1253 segundos (aproximadamente 21 minutos), ou seja, cerca de 2 minutos de ganho.

Além do `tessed_do_invert`, trabalhou-se com dois outros parâmetros do *tesseract*: `psm` (*page segmentation mode*) e `oem` (*OCR engine mode*). O `psm` define a forma da imagem que o *tesseract* irá considerar ao lidar com a análise de *layout*, ou seja, o usuário deve verificar qual formato de imagem (no caso do presente trabalho, as páginas dos relatórios) e selecionar a que considera mais apropriadas dentre as 13 opções disponíveis. Conforme pode ser visto na documentação da biblioteca, o valor *default* para o `psm` é o `3 - auto` (*Fully automatic page segmentation, but no OSD*), ou seja, o que foi utilizado no experimento #0 do *pytesseract*. Essa opção define que a biblioteca fará a segmentação da imagem, mas não fará detecção por orientação e *script*. O experimento #1 serve como uma espécie de prova de veracidade do parâmetro e já se espera um resultado ruim, pois considera as imagens como um único caractere ao utilizar o `psm` com valor `10 - single_char` (*Treat the image as a single character*). O experimento #2 e #3 servem para verificar o impacto no desempenho se as imagens fossem consideradas um único bloco de texto, na vertical (quando `psm` é `5 - single_block_vert_text` (*Assume a single uniform block of vertically aligned tex*)) ou não (quando `psm` é `6 - single_block` (*Assume a single uniform block of text*)). A motivação para essa diferenciação nos dois últimos experimentos citados está associada ao fato dos relatórios financeiros possuírem várias tabelas com informações relevantes para as buscas.

Já o `oem` serve para especificar qual o tipo de algoritmo o *tesseract* irá utilizar para fazer o OCR (*optical character recognition*). A documentação cita 4 opções disponíveis:

- `0` = *Original Tesseract only.*
- `1` = *Neural nets LSTM only.*
- `2` = *Tesseract + LSTM.*
- `3` = *Default, based on what is available.*

Em um experimento não catalogado percebeu-se que a mudança somente desse parâmetro não causava impactos expressivos nas métricas e resultados finais. Não chegou a ficar claro se isso acontecia por causa que a opção *default* que utiliza o algoritmo disponível tinha como algoritmo o especificado então nesse experimento não catalogado (*tesseract + LSTM*). Devido a isso tudo, não aprofundou-se os estudos nessa parte da extração e manteve-se somente o experimento #1 com uma configuração que altera esse argumento.

Já todas as outras bibliotecas utilizadas na plataforma estão com suas configurações *out of box*, com somente as seguintes exceções:

- Especificação para executarem com a linguagem do português sendo utilizada:
  - *Tokenização*;

  - Conjunto de *stopwords*;
  - *tesseract*

- *pdf2image* que faz a conversão das páginas do arquivo PDF para imagem no formato PIL com 500 DPI. O valor 500 foi escolhido arbitrariamente, o único requisito é que fosse maior que o valor *default* (200) - na linha de raciocínio que uma conversão com mais DPI geraria um resultado melhor

Todo conjunto de testes entregue no trabalho gerou dois resultados: o tempo para a execução e o acerto da busca. O tempo levado para a execução do conjunto de teste foi fornecido pelo próprio *framework* de teste, o *unittest*, que apresenta a informação após a execução de um grupo de testes. Já o acerto da busca depende de alguns processos para ser computado. Primeiro é necessário manualmente procurar o valor referente ao indicador financeiro desejado no relatório trimestral da empresa que será testada. Após encontrar esse valor, preenche ele no arquivo `data.py` que será utilizado como tabela verdade para os *asserts* dos testes unitários (conforme também explicado anteriormente).

```
1  PATRIMONIO_LIQUIDO = {
2                        'ambev_2019_1T.pdf': 60490.0,
3                        'engie_2019_2T.pdf': 7194673.0,
4                        'engie_2020_2T.pdf': 7434743,
5                        'fleury_2019_3T.pdf': 1750.1,
6                        'fleury_2020_2T.pdf': 1549453.0,
7                        'gerdau_2015_3T.pdf': 36012381.0,
8                        'gerdau_2017_1T.pdf': 24916345.0,
9                        'gerdau_2018_4T.pdf': 25938571.0,
10                       'gerdau_2019_2T.pdf': 26568445.0,
11                       'weg_2010_2T.pdf': 2445405.0,
12                       'weg_2015_1T.pdf': 5302661.0,
13                       'weg_2017_2T.pdf': 6348046.0,
14                       'weg_2019_2T.pdf': 8017
15                      }
```

Parte do arquivo data.py para exemplificação

Ainda no processo para ter o resultado relacionado ao acerto, executa-se então o conjunto de testes unitários configurado, que irá dizer se naquela especificação a plataforma trouxe o resultado correto ou não para o indicador financeiro - através de uma comparação no *assert* com o resultado esperado e conhecido anteriormente.

5.3 Análise dos resultados

Conforme pode ser esperado, cada experimento executado gerou uma tabela de resultados. Essa tabela possui como linhas os relatórios financeiros presentes no *dataset* e como colunas os indicadores financeiros associados a cada tipo de busca possível desenvolvido no protótipo. Cada célula resultante da relação entre linhas e colunas possui um valor binário como resultado, onde "Sim" significa que a busca daquele indicador financeiro trouxe o resultado esperado para o *dataset* em questão e "Não" quer dizer que o valor esperado não foi encontrado. Além disso, comentários textuais sobre cada célula também foram adicionados na tabela para proporcionar mais significado ao resultado, de forma a permitir uma avaliação empírica de cada caso. Por fim, com essa tabela pode-se tirar diversas métricas que possibilitam a comparação entre os experimentos.

Para permitir uma análise melhor dos resultados e o cálculo de métricas, a matriz de confusão para os resultados dos experimentos foi contabilizada. A matriz resultante é a seguinte:

Nome experimento	**Atual (coluna) / Resultado (linha)**	*Sim*	*Não*
Extração com PyPDF2 #0	*Sim*	10	43
	Não	0	38
Extração com pytesseract #0	*Sim*	16	37
	Não	0	38
Extração com pytesseract #1	*Sim*	0	53
	Não	0	38
Extração com pytesseract #2	*Sim*	21	32
	Não	0	38
Extração com pytesseract #3	*Sim*	0	53
	Não	0	38

Tabela 10 – Resultado - matriz de confusão

Para deixar mais claro como se chegou nessa matriz de confusão, vale explicar o que significa cada associação de "Atual" com "Resultado". "Atual", a segunda coluna da esquerda para direita, quando "Sim" quer dizer que o relatório financeiro possui o indicador financeiro no formato que a busca deseja e quando "Não" significa que o relatório não possui o indicador no formato esperado. Já o "Resultado", primeira linha, quando "Sim" expressa que o experimento trouxe o valor correto do indicador financeiro a partir do relatório, mas no "Não" indica que o valor correto não foi encontrado. Com isso, pode-se fazer as seguintes associações:

- Atual Sim + Resultado Sim (TP): são os verdadeiros positivos, ou seja, o valor do indicador financeiro existe no relatório e o experimento conseguiu identificá-lo

- Atual Não + Resultado Sim (FP): são os falsos positivos, ou seja, o valor do indicador financeiro não existe no relatório, mas o experimento identificou algum "lixo" e considerou que fosse o indicador financeiro

- Atual Sim + Resultado Não (FN): são os falsos negativos, ou seja, o valor do indicador financeiro existe no relatório e o experimento não conseguiu identificá-lo

- Atual Não + Resultado Não (TN): são os verdadeiros negativos, ou seja, o indicador financeiro não existe no relatório e o experimento não identificou nada como o valor do indicador financeiro

Portanto, a primeira métrica que pode ser calculada a partir dela é a acurácia do experimento, ou seja, a porcentagem de resultado "Sim" perante ao total de buscas realizadas em cada experimento. Matematicamente, isso é o número de buscas corretas $(TP + TN)$ dividido por todas as buscas realizadas. Importante notar que se um relatório de empresa não possuir um indicador financeiro no formato que a busca espera encontrar, então na tabela de resultados considera-se que a busca obteve sucesso ao não retornar nada para aquele valor, por exemplo, se um relatório da empresa X não possui patrimônio líquido em um valor monetário (após um R$) e essa busca na plataforma trouxe zero resultados, então o valor resultante do teste será "Sim" - um sucesso. Para não poluir a tabela, vale citar que cada experimento possui 91 resultados, logo as acurácias observadas foram:

Nome experimento	**Predições corretas (TP + TN)**	**Acurácia**
Extração com PyPDF2 #0	48	52,7%
Extração com pytesseract #0	54	59,3%
Extração com pytesseract #1	38	41,7%
Extração com pytesseract #2	59	64,8%
Extração com pytesseract #3	38	41,7%

Tabela 11 – Resultado - acurácia

A segunda métrica possível de ser calculada é o *recall*, nela não se considera 91 como o total de resultados, mas sim somente as buscas na qual o relatório financeiro possuía o valor que deveria ser buscado. Dessa forma, saberemos o quão boa é a plataforma em encontrar indicadores financeiros quando os mesmos estão presentes. Matematicamente, a fórmula para o cálculo do *recall* é $TP/(TP + FN)$. Com isso, a população do total de resultados vai de 91 para 53 $(TP + FN)$. Dessa forma, os *recall* examinados foram:

A terceira métrica computada foi a precisão dos experimentos. Aqui também o total de resultados deixa de ser 91 e passa a ser as buscas que retornaram algum indicador financeiro, seja ele o esperado ou não. Desse jeito, pode-se saber a proporção

Nome experimento	Verdadeiros positivos (TP)	Recall
Extração com PyPDF2 #0	10	18,8%
Extração com pytesseract #0	16	30,1%
Extração com pytesseract #1	0	0,0%
Extração com pytesseract #2	21	39,6%
Extração com pytesseract #3	0	0,0%

Tabela 12 – Resultado - *recall*

de identificações positivas que foram realmente corretas. A fórmula matemática utilizada para o cálculo da precisão é então $TP/(TP + FP)$. Portanto, as precisões identificadas foram:

Nome experimento	Verdadeiros positivos (TP)	Total positivos (TP + FP)	Precisão
Extração com PyPDF2 #0	10	10	100%
Extração com pytesseract #0	16	16	100%
Extração com pytesseract #1	0	0	N/A[1]
Extração com pytesseract #2	21	21	100%
Extração com pytesseract #3	0	0	N/A[1]

Tabela 13 – Resultado - precisão

A quarta métrica que pode ser contada é o *F1-score*. O *F1-score* é uma forma mais robusta (considera mais nuances dos experimentos de classificação / busca) de se dizer o quanto de acurácia cada experimento possui. O valor vai de 0 a 1 e quanto mais próximo de 1, melhor foi o experimento. Essa métrica é dependente do *recall* e da precisão, pois sua fórmula é $2 \times \frac{precisao \times recall}{precisao + recall}$. Sendo assim, os *F1-score* vistos são:

Nome experimento	***F1-score***
Extração com PyPDF2 #0	0,31
Extração com pytesseract #0	0,46
Extração com pytesseract #1	0,00
Extração com pytesseract #2	0,56
Extração com pytesseract #3	0,00

Tabela 14 – Resultado - *F1-score*

Por fim, a última métrica é uma apresentação dos resultados dos experimentos de outra forma, já que até então tudo foi baseado na matriz de confusão, e, ela não considerou a unicidade dos resultados para fazer diferenciação entre as classes dos experimentos. Isso quer dizer, através de um exemplo, que se uma busca de um experimento traz mais de um valor para determinado indicador financeiro (*e.g.* 100 e 200), mas somente um desses

[1] não aplicável

valores é o valor correto (*e.g.* 100), considerou-se na matriz de confusão que essa busca foi um sucesso, ou seja, um TP. Logo, na apresentação da próxima tabela o total da população continua sendo todas as buscas realizadas - 91. Além disso, a segunda coluna apresenta quantas buscas do total da população trouxeram resultados, enquanto a terceira e quarta coluna são subconjuntos da segunda: onde a terceira é quantas buscas que trouxeram resultados trouxeram somente um único resultado e a quarta é quantas buscas trouxeram mais de um resultado. Já os valores da última coluna seriam o cenário perfeito de resposta, quando a busca traz um único valor e esse valor é o correto.

Nome experimento	**Encontrou algo**	**Encontrou um único valor**	**Encontrou + de um valor**	**Encontrou um único valor correto**
Extração com PyPDF2 #0	16	10	6	7
Extração com pytesseract #0	25	13	12	7
Extração com pytesseract #1	0	0	0	0
Extração com pytesseract #2	32	15	17	7
Extração com pytesseract #3	0	0	0	0

Tabela 15 – Resultado - valores únicos

Além disso, uma aferição importante é o tempo que cada experimento levou para ser executado. Esse dado não está na tabela de resultados compilada e citada no começo da presente seção, mas sim foi informado pelo *unittest* (*framework* de teste utilizado) após a execução dos experimentos. Os tempos verificados foram:

Nome experimento	**Tempo (valores aproximados)**
Extração com PyPDF2 #0	17 segundos $\sim$ 0,2 minutos
Extração com pytesseract #0	1792 segundos $\sim$ 29,8 minutos
Extração com pytesseract #1	739 segundos $\sim$ 12,3 minutos
Extração com pytesseract #2	1783 segundos $\sim$ 29,7 minutos
Extração com pytesseract #3	2362 segundos $\sim$ 39,3 minutos

Tabela 16 – Resultado - tempo

Com todas as métricas apresentadas é possível concluir que os resultados por parte dos experimentos podem ser considerados medianos. Ao fazer uma interpretação sem considerar os valores de *F1-score* zerados, ambos casos especiais, os experimentos tiveram um resultado entre 0,31 e 0,56, ou seja, não recomenda-se confiar completamente na plataforma para realizar a busca de algum indicador financeiro. Ao decorrer do desenvolvimento

do trabalho percebeu-se alguns fatores que levaram a esse tipo de resultado:

- A falta de padronização dos relatórios financeiros. Não há uma padronização de *layout* entre os relatórios de diferentes empresas. Além disso, com o decorrer dos anos, relatórios da mesma instituição mudam seu *layout* devido a alguns fatores, como por exemplo a mudança de identidade visual da companhia. A ausência de um padrão gera muitos casos específicos, que ainda não são sempre replicáveis em novos relatórios, para serem tratados nas filtragens e buscas, além de dificultar o *tuning* das bibliotecas que realizam a transformação do arquivo PDF para texto.
- O alto número de tabelas nos relatórios financeiros. Inicialmente esperava-se que as informações desejadas estariam no formato de "texto corrido" nos arquivos. Só que verificou-se que muito do desejado fica presente em tabelas, que também não são padronizadas. Isso gera a dificuldade de ser sempre necessário ao mesmo tempo identificar um "texto corrido" e uma tabela no mesmo arquivo, com as mesmas configurações para as bibliotecas utilizadas.
- A abrangência dos objetivos iniciais do trabalho. Ao querer buscar qualquer indicador financeiro, em diferentes formatos e posicionamento e em qualquer relatório empresarial, abre-se margem para muitas particularidades que precisam ser tratadas. Se por um outro lado, os objetivos tivessem sido mais restritos, como por exemplo, trabalhar só com as tabelas dos relatórios financeiros de determinada empresa, talvez os resultados tivessem sido mais positivos.

Todas as métricas comprovam que o melhor experimento foi o **Extração com pytesseract #2**, que considera cada página dos relatórios financeiros como um único bloco de texto, já que ele possui os melhores resultados em todas as avaliações. Por outro lado, a diferença de valores que faz seu desempenho ser melhor ao ser comparado com o segundo lugar dos experimentos, **Extração com pytesseract #0** - a versão *out of box* do *pytesseract*, é baixa, pois por exemplo, trata-se somente de 0,1 a mais de *F1-score*. Esse fato gera uma linha de raciocínio interessante: a configuração adequada das bibliotecas utilizadas podem melhorar o desempenho da plataforma, mas para o atual escopo do trabalho, suas versões *out of box* possuem um desempenho razoável, e, dependendo do tipo de configuração realizada, os resultados podem piorar drasticamente (vide **Extração com pytesseract #1** e **Extração com pytesseract #3**). Portanto, a configuração tem que ser realizada com cautela.

Alguns aspectos interessantes podem ser observados ao comparar os resultados dos experimentos entre si:

- O *pytesseract*, quando configurado adequadamente, possui resultados melhores do que o *PyPDF2*, mas há um preço relativo ao tempo de processamento pago para

conseguir métricas superiores. O tempo de execução dos experimentos que deram bons resultados com *pytesseract* é cerca de 90 vezes maior do que a execução com *PyPDF2*, que leva menos de 1 minuto para acontecer.

- Não houve nenhuma busca que trouxe um valor para o indicador financeiro que não existia no relatório. Ou seja, nenhum experimento teve falso positivo, o que leva a uma precisão de 100% para todos os experimentos que tiveram resultados.
- Devido à configuração utilizada, alguns experimentos com *pytesseract* tiveram um resultado extremamente negativo, sem nenhum verdadeiro positivo encontrado. Em outras palavras, esses experimentos não encontraram nenhum valor para o indicador financeiro. Isso fez com que em algumas métricas a expressão N/A (não aplicável) fosse utilizada para evitar o problema matemático de divisão por zero.

5.4 Plataforma experimental

Além dos resultados obtidos com os experimentos, pode-se considerar que o presente trabalho também possui uma contribuição relevante através do código que compõe o protótipo que realiza as experimentações. Esse protótipo pode até ser considerado como uma plataforma, em fase inicial, para a busca de indicadores financeiros em relatórios financeiros. A atual seção visa explicitar como funciona a execução dessa plataforma e apontar em quais pontos ela, a plataforma, pode ser expandida - dessa forma, as camadas da plataforma são explicadas como consequência.

Comentou-se previamente sobre a execução na seção de experimentos ao citar que tudo é executado através do *make* e um único arquivo `makefile`. Porém, para que o `makefile` funcione é necessário uma pré-configuração. Anteriormente, na seção de tecnologias utilizadas, explicitou que a versão do *python* utilizada no trabalho é a `3.7.6`. Por uma questão de boa prática, recomenda-se que a configuração e instalação dessa versão seja feita através da ferramenta *pyenv*. De modo resumido, após o *pyenv* estar instalado (ação que varia conforme sistema operacional), basta executar `pyenv install 3.7.6` (comando auto-explicativo) e depois `pyenv global 3.7.6` (comando que define como versão do *python* a `3.7.6`), e, pronto - versão do *python* configurada corretamente.

Após isso, é preciso instalar o *poetry*, o que também muda conforme o sistema operacional, e com o *poetry* instalado executar `poetry install` (comando que instalará todas as dependências do trabalho) para finalizar a pré-configuração. Por fim então, basta executar `make` com alguma das regras especificadas no `makefile`. As regras existentes até então são:

- `main`: executa rascunhos de baixa fidelidade e sem valor para o resultado final codificados na classe `main.py`

- `test-all`: executa todos os testes codificados no trabalho
- `test-helper`: executa todos os testes codificados para a camada `helper`
- `test-plataform`: executa todos os testes codificados para a camada `plataform`
- `test-report`: executa todos os experimentos explicitados na tabela
- `test-pypdf2`: executa os testes do experimento **Extração com PyPDF2 #0**
- `test-pytesseract-default`: executa os testes do experimento **Extração com pytesseract #0**
- `test-pytesseract-config01`: executa os testes do experimento **Extração com pytesseract #1**
- `test-pytesseract-config02`: executa os testes do experimento **Extração com pytesseract #2**
- `test-pytesseract-config03`: executa os testes do experimento **Extração com pytesseract #3**
- `linter`: executa a verificação da padronização do código através do *flake8*

Um ponto importante de se notar é que no geral, só será necessário adicionar uma nova regra ao `makefile` em duas situações: uma nova camada que possui testes unitários foi adicionada a plataforma, ou, um novo experimento com uma configuração diferente foi codificado.

Para explicar como o protótipo pode ser expandido é preciso também sintetizar de forma evidente todas as camadas (já pinceladas durante todo o texto do desenvolvimento) que ele possui. Logo, são elas e suas possíveis "expansões" básicas:

- `data`: camada com todos os relatórios de resultados empresariais em formato PDF. Há um desejo de rodar os experimentos em um novo relatório? Vai ser necessário então adicionar o arquivo do relatório aqui e atualizar o arquivo `data.py` (também presente no diretório) com os valores coletados manualmente dos indicadores financeiros;
- `src/helper`: camada que possui todas as funções consideradas auxiliares e genéricas, pois são invocadas em mais de algum lugar da plataforma. Há uma nova funcionalidade auxiliar (como, por exemplo, saber se uma *string* é um número válido) desenvolvida e não sabe onde colocar? Uma classe no diretório `src/helper` é o mais recomendado;

- `src/indicator`: camada com todos os indicadores financeiros disponíveis de serem buscados nos relatórios. Um novo indicador financeiro vai ser coletado? Além de alterar todos os experimentos já realizados e adicionar novas funções para cada tipo de busca associado ao novo item no `manager.py`, é imprescindível também que se crie uma nova classe no diretório conforme a modelagem para a nova adição;
- `src/plataform`: camada com a implementação do centro do fluxo de trabalho, como pode ser visto por suas principais classes:
  - `pdf_extract.py`: como já sabido, responsável pela extração do texto. Logo, se for necessário utilizar uma nova biblioteca para extração do texto dos relatórios financeiros, então vai ser preciso realizar alterações aqui;
  - `filters.py`: também já de conhecimento, é a responsável pelas filtragens. Um filtro novo está sendo estudado? Se a resposta for positiva, um método precisará ser criado então nessa classe;
  - `preprocessor.py`: responsável pelo pré-processamento, conforme já informado. Deseja-se limpar mais o texto com a remoção de determinados caracteres, então será inevitável atualizar esse arquivo;
  - `searcher.py`: encarregada de implementar as diferentes formas de buscas nos relatórios financeiros. Cogita-se a eficácia de uma busca onde o valor financeiro está antes de uma palavra chave? Será preciso então criar um novo método neste lugar;
- `src/technique`: camada que possui as técnicas de PLN do protótipo. Deseja utilizar uma nova técnica de PLN associada aos indicadores financeiros, como por exemplo a *lemmatisation*? Se sim, uma classe terá que ser adicionada aqui nessa camada e os ajustes necessários nas outras camadas também terão que ser realizados, como na definição dos indicadores ou no pré-processamento;
- `src/manager.py`: classe que é central da aplicação, pode-se dizer que é a camada que organiza toda a plataforma. Praticamente todas as possíveis expansões citadas até então iriam exigir algum ajuste nessa classe;
- `tests`: camada que possui os testes unitários que garantem a corretude de todas as outras camadas e também todos os experimentos realizados. Portanto, fácil perceber que alterações em outras camadas ou novos experimentos exigem manutenção no `tests`;

Com tudo isso é possível confirmar que o protótipo criado para a realização dos experimentos trata-se na verdade de uma plataforma que pode ser facilmente expandida em diferentes caminhos que confluem para a busca de indicadores financeiros em relatórios de resultados empresariais.

5.5 Discussão dos resultados

É possível também avaliar as contribuições do presente trabalho através de uma discussão dos seus resultados a partir da comparação dele com outros estudos da área (os trabalhos relacionados). A seguir segue a tabela apresentada na seção de trabalhos relacionados com a adição de uma linha que representa o estudo desenvolvido.

Trabalho	Área do dataset	Dataset aberto?	Tamanho do dataset	Origem dataset
Cálculo indicadores financeiros (atual)	**Financeiro**	**Sim**	**13 relatórios**	**Empresas listadas na bolsa brasileira**
Sudhams e Saripalli (2019)	Financeiro	Sim	14724 documentos	Site do SEC (EUA)
Daudert e Ahmadi (2019)	Financeiro	Sim	2655 documentos	CAC 40 + CAC Next 20 (França)
Lin *et al.* (2011)	Financeiro	Sim	26255 documentos	S&P 500 (EUA)
Zhang (2015)	Financeiro	Sim	Documentos de 2002 até 2012 de 15 empresas do S&P 500	S&P 500 (EUA)
El-Haj *et al.* (2014)	Financeiro	Sim	1500 relatórios financeiros	Empresas listadas na bolsa de Londres

Tabela 17 – Comparação trabalho atual com trabalhos relacionados (continua)

Ao fazer uma análise comparativa entre o trabalho desenvolvido com os outros na tabela pode-se tirar diversas conclusões. A primeira delas é que de forma similar aos outros estudos de referência da área, o presente trabalho utiliza um *dataset* público da área financeira. Logo em seguida é possível perceber que o tamanho do *dataset* é extremamente diferente entre o atual trabalho e as referências. Enquanto os relacionados trabalham na casa dos milhares de itens no *dataset*, o atual trabalho está na casa das dezenas. Isso acontece principalmente por causa que a construção do *dataset* no atual trabalho é totalmente manual e a adição de um item a mais no *dataset* acarreta no cálculo manual dos indicadores e na adição de uma bateria de testes. O ponto negativo dessa diferença é

Trabalho	**Técnicas utilizadas**	**Compara técnicas utilizadas?**
Cálculo indicadores financeiros (atual)	**OCR; tokenização; stemming; script que filtra e busca informação alvo**	**Sim, compara as diferentes ferramentas utilizadas para realizar o OCR**
Sudhams e Saripalli (2019)	*scripts* para coleta de dados; *scripts* para limpeza de dados; modelo de *Conditional Random Field*	Somente o modelo de *Conditional Random Field* é comparado com trabalhos externos
Daudert e Ahmadi (2019)	*nltk* (ferramenta criação corpus); *pdftotext* (ferramenta criação corpus); RNN (experimento); *Word2vec* (experimento)	Não
Lin *et al.* (2011)	HAC (método); *K-means* (método); SVM (comparação); *Naïve bayes* (comparação); PFHC (comparação)	Sim
Zhang (2015)	SDA (modelo); DBN (modelo); RNN-RBM (modelo); SVM (modelo); *Random Forests* (modelo)	Sim
El-Haj *et al.* (2014)	*iText* (ferramenta)	Sim, compara resultados em duas etapas do processo

Tabela 18 – Comparação trabalho atual com trabalhos relacionados (continua)

Trabalho	Propósito	Forma de avaliação
Cálculo indicadores financeiros (atual)	**Calcular indicadores financeiros a partir de relatórios financeiros**	**Avaliação se os resultados calculados e buscados são os esperados após comparação com uma tabela verdade dos valores**
Sudhams e Saripalli (2019)	Criar um *dataset* estruturado e uma aplicação web dos relatórios de *Credit Default Swap*	Avaliação de desempenho das etapas propostas
Daudert e Ahmadi (2019)	Criar um corpus da língua francesa a partir de relatórios financeiros	Experimentação do corpus através de três tarefas de PLN
Lin *et al.* (2011)	Prever o preço de ações em um curto espaço de tempo a partir de relatórios financeiros	Comparação do desempenho do método proposto com outras técnicas
Zhang (2015)	Verificar a aplicabilidade de modelos de *deep learning* na predição do preço de ações com o uso de relatórios financeiros	Comparação após experimentação dos modelos de *deep learning* com modelos clássicos de *machine learning*
El-Haj *et al.* (2014)	Identificar e extrair diferentes seções de relatórios financeiros	Manual com *experts* da área

Tabela 19 – Comparação trabalho atual com trabalhos relacionados (continua)

a confiabilidade do resultado, já que com mais dados em um *dataset*, mais confiável os resultados seriam.

Outra conclusão praticável é de que o corrente trabalho é o único que utiliza como origem do *dataset* informações de empresas brasileiras, enquanto a origem dos outros são de países europeus ou dos Estados Unidos. Esse detalhe pode parecer não muito impactante, mas implica no fato de que os relatórios financeiros nacionais não possuem padronização - o que dificulta os estudos. Ao passo que em alguns outros países há um padrão para esses documentos. Percebeu-se na prática que um *dataset* com conteúdo padronizado facilita o desenvolvimento de uma pesquisa. Além disso, na parte de técnicas utilizadas não há uma técnica que seja comum em todos os trabalhos relacionados. Nessa ideia, o presente trabalho segue essa linha ao utilizar as técnicas que fazem mais sentido para que os objetivos sejam alcançados. Por outro lado, a maioria das referências comparam

Trabalho	Métricas	Resultado
Cálculo indicadores financeiros (atual)	**Matriz de confusão; Acurácia; *Recall*; Precisão; *F1-score*; Tempo.**	**Uma plataforma que realiza busca e cálculo de indicadores financeiros com resultados medianos**
Sudhams e Saripalli (2019)	Acurácia na extração de dados; *F1-score*; Precisão; Sensibilidade;	*Dataset* construído, aplicação web desenvolvida e também gerou-se outros produtos menores durante o projeto
Daudert e Ahmadi (2019)	*F1-score*; Perplexidade; Avaliação qualitativa.	*Corpus* gerado possui potencial de ser utilizado na criação de novas pesquisas na área
Lin *et al.* (2011)	Acurácia da predição; Média de lucro por *trade* no curto espaço de tempo; Complexidade de tempo	Método alcança o propósito determinado inicialmente e permite a aplicação, se adaptado, em outras áreas
Zhang (2015)	Matriz de confusão; Taxa de precisão e *recall* que formam o *F1-score*; Acurácia de precisão; Estabilidade do modelo	Modelos de *deep learning* possuem um desempenho relativamente bom
El-Haj *et al.* (2014)	*Gunning fog index* para legibilidade; *Flesch-Kincaide score* para legibilidade; Precisão; Sensibilidade; *F1-score*	Uma porcentagem alta de extrações corretas (no geral acima de 85%) conforme métricas apresentadas e obtidas a partir da avaliação manual

Tabela 20 – Comparação trabalho atual com trabalhos relacionados (conclusão)

de alguma forma as técnicas utilizadas em determinados aspectos, o que também é uma verdade no atual trabalho - comparação que contribuiu de forma relevante nas conclusões do trabalho.

Ao observar os diferentes propósitos, é possível dizer que entre as referências há três grupos de propósito: predizer preço de ações com base em algum modelo; construir uma base de dados a partir de dados financeiros, e, extrair informações a partir de relatórios financeiros. O presente trabalho possui o último desses principais propósitos identificados nas referências. Outro ponto de observação é que como cada trabalho buscou utilizar de uma forma de avaliação diferente, devido a especificidade associada a cada assunto tratado, isso implicou em um conjunto de métricas diferentes entre as referências. Por outro lado, há uma métrica em comum a quase todos os relacionados - o *F1-score*. Com isso, pode-se dizer que o F1-score e as outras principais métricas calculadas no atual trabalho são fortemente inspiradas pela presença nos trabalhos da área. E por último, quanto aos resultados, no geral, todas as referências possuem resultados medianos para ótimos. De maneira esperada, cada trabalho apresenta dificuldades intrínsecas ao seu domínio e execução. Isso também é uma realidade do atual trabalho, que teve suas dificuldades (maioria convertida em possibilidade de trabalhos futuros), mas mesmo assim ainda conseguiu apresentar um resultado mediano do ponto de vista das métricas.

6 Conclusão

Esta seção tem como objetivo fazer o fechamento do presente trabalho. Isso é realizado ao verificar se os objetivos propostos foram cumpridos e também identificar melhorias e possíveis continuações do trabalho.

É importante perceber após uma averiguação dos objetivos definidos inicialmente que todos foram de certa forma cumpridos. A análise do estado da arte das técnicas de PLN foi feita de forma não exaustiva (um ponto que poderia ter sido melhor, pois há mais referências que poderiam ter sido abordadas) e, com ela, foi possível elencar as técnicas mais adequadas ao objetivo geral do trabalho, bem como fundamentar o escopo do modelo. Além disso, no decorrer da execução dessa análise o escopo do estudo foi restrito ao cálculo de um indicador devido a percepção adquirida de que para se calcular um indicador financeiro teria que se encontrar as variáveis que permitem o cálculo no próprio relatório. Sendo assim, a pesquisa mirou em buscar indicadores financeiros e também em calcular um indicador financeiro a partir do resultado das buscas. É importante perceber que uma vantagem de também se realizar a busca, é que em alguns casos (devido a falta de padrão entre os relatórios), o indicador financeiro que deseja ser calculado pode estar presente no documento e ser encontrado ao invés de ser calculado a partir de outros.

A definição dos requisitos do modelo do presente estudo foi influenciada por diversas frentes. Por exemplo, ao definir quais tecnologias e ferramentas seriam utilizadas dentro das diversas disponíveis para implementação do modelo, foi possível se aproveitar do insumo obtido na pesquisa de trabalhos relacionados para o estabelecimento desses tópicos - sem deixar de dar importância em nenhum momento na familiaridade do autor com as ferramentas e tecnologias. Só que mesmo com a utilização de referências para definição dos requisitos, erros aconteceram na implementação até se chegar na definição que se tem hoje das premissas do modelo. Esses requisitos foram construídos a partir da sedimentação do conhecimento adquirido com base nas tentativas executadas de se alcançar o objetivo. Vale citar também que outra frente relevante para a determinação da forma de trabalho que iria alcançar a contemplação dos requisitos definidos foi o estado da arte da área.

Como citado anteriormente, a organização da proposta de modelo para ajudar no cálculo de indicadores financeiros seguiu os modelos apresentados pelos trabalhos do campo de estudo. Por isso, a proposta possui as etapas mais comuns de uma pesquisa do gênero: construção do *dataset*, extração da informação, pré-processamento e aplicação da técnica definida. Após isso, o plano parte para a filtragem das informações e a realização das buscas dos indicadores financeiros que permitirão o cálculo posterior do indicador financeiro - essa foi a forma concebida no trabalho para alcançar o objetivo proposto.

Devido à dificuldade encontrada no objeto que compõe o *dataset*, o relatório financeiro, de não possuir um padrão de informações apresentadas e nem de em quais lugares elas estarão presentes ("texto corrido" ou tabelas), foi necessário adaptar a filtragem, busca e cálculo citados anteriormente para que fosse possível a busca se comportar de duas formas: uma forma mais propícia para informações em "texto corrido" e outra forma mais adequada para tabelas. Essa adaptação pode ser vista como uma vantagem que deixa o modelo mais genérico, mas ao mesmo tempo também é possível considerar ela como uma desvantagem, pois impacta nos resultados e nas especializações possíveis de se fazer no presente modelo.

Para se chegar na definição dos experimentos que executariam o modelo proposto, várias etapas foram necessárias. Uma dessas etapas foi a definição das métricas que seriam coletadas ao fim da aplicação do experimento, pois com elas então seria possível posteriormente comparar os diferentes experimentos entre si. A principal fonte, para definição de quais métricas seriam utilizadas dentro da grande diversidade possível, foi os trabalhos relacionados. Outra etapa realizada consistia na necessidade de saber se o resultado da busca ou do cálculo de um indicador financeiro (o objetivo principal do trabalho) estava correta, para aí então permitir a coleta e computação das métricas do experimento.

Nessa etapa concluiu-se que os experimentos seriam então compostos por diferentes bateria de testes unitários, onde cada bateria possuiria uma característica de configuração diferente da outra. Dessa forma, haveria uma facilidade na conferência dos resultados e na detecção de possíveis alterações do código. Houve também a etapa que serviu para definir qual configuração cada experimento teria para poder se definir como um experimento seria único perante aos outros. Assim como os outros passos, mas principalmente esse, essa definição aconteceu de forma evolutiva, ou seja, não foi algo esboçado desde o começo da fase de experimentação, mas sim algo que ocorreu gradualmente conforme tentativas eram realizadas e conhecimentos eram adquiridos. Uma importante característica da diferenciação entre os experimentos é a limitação feita de que a diferença entre eles iria residir somente em como a informação seria transformada do arquivo PDF para se tornar uma entrada no experimento - limitação na diferenciação que pode ser vista como um benefício no sentido de simplificação do trabalho.

Como dito anteriormente, em paralelo à definição dos experimentos que verificam o modelo proposto, o desenvolvimento do protótipo que executa esses experimentos começou a ser realizado. Iniciar antes gerou um impacto positivo no resultado do presente trabalho. O ganho foi a entrega também de uma plataforma que permite a busca e cálculo dos indicadores financeiros, dessa forma um direcionamento além de somente ser um conjunto de código que testa a modelagem. Essa plataforma é caracterizada principalmente pela organização através de diferentes camadas, o que facilita possíveis expansões em trabalhos futuros.

Finalmente, após os experimentos definidos e a plataforma pronta, a execução dos experimentos foi realizada com o intuito de computar os resultados para a futura verificação. Uma imperfeição nesse processo foi o descobrimento, somente após iniciar o mesmo, da demora de execução de alguns experimentos. Depois dos experimentos terem sido executados, é possível considerar que a verificação dos resultados foi realizada de duas formas. A primeira delas pode ser imaginada como algo mais micro, pois trata-se da verificação através da comparação das métricas obtidas por cada experimento, com a intenção de buscar alguma conclusão sobre as diferenças entre eles, além de verificar se o objetivo principal da pesquisa foi alcançado. Por outro lado, há o segundo modo de verificação, passível de ser chamado de modo macro, pois faz uma discussão de comparação com os trabalhos relacionados, após a inserção do presente trabalho na tabela que compara todos os que foram utilizados como referência. Em conclusão, ambas as formas de verificação mostram que o objetivo principal do atual trabalho (calcular indicadores financeiros a partir de relatórios de resultados empresariais com o auxílio de PLN) foi alcançado.

6.1 Trabalhos futuros

A partir da discussão dos resultados apresentada anteriormente, dos resultados obtidos e das dificuldades encontradas durante o desenvolvimento da atual pesquisa, percebe-se diferentes direções em que o atual estudo poderia ser continuado. A direção inicial seria adicionar busca e cálculo por novos indicadores financeiros. Outro caminho interessante seria aumentar o tamanho do *dataset* com mais relatórios financeiros, das mesmas empresas utilizadas ou de empresas diferentes, já que quanto maior o *dataset*, mais valioso o modelo pode ficar. Além disso, há a opção de restringir o escopo de busca e cálculo para alguma parte específica do relatório financeiro, como por exemplo as tabelas. Dessa forma provavelmente seria mais fácil especializar as técnicas utilizadas com o intuito de conseguir melhores resultados.

Um direcionamento relevante seria tratar as problemáticas encontradas no desenvolvimento da plataforma que executa os testes no modelo, onde as que vale citar são: diferenciação do caractere ponto e vírgula (;) com o caractere vírgula (,) iria ajudar em alguns relatórios financeiros; considerar escala ao apresentar os indicadores financeiros encontrados ou calculados; e, por fim, possibilitar a transformação de arquivos PDFs em *string* quando os mesmos forem cifrados de uma forma que dificulte esse processo. No mesmo âmbito de possibilidades de evolução da plataforma, algumas ideias consideradas interessantes para incrementar a mesma são: criar uma abstração para as classes que representam indicadores financeiros, pois dessa forma estaria mais de acordo com as boas práticas de programação; e também criar uma plataforma online para execução dos experimentos. Um esboço disso seria permitir que um usuário enviasse o arquivo PDF e após selecionar e informar os valores (sabido de antemão por ele) dos indicadores financeiro

que deseja testar a plataforma, então a plataforma online daria as métricas alcançadas. Finalmente, a última indicação de encaminhamento para futuro trabalho seria buscar reduzir o tempo de execução de cada experimento por relatório financeiro quando utiliza o *pytesseract*, que possui o melhor desempenho de acordo com as métricas. Hoje com 3 indicadores financeiros sendo calculados, o tempo em média levado para rodar os testes de cada relatório financeiro é de 2 minutos e meio.

Referências

AN, Bo. *Predicting Human Decision-Making, Ariel Rosenfeld, Sarit Kraus, Morgan & Claypool Publishers (2018)*. [S.l.]: Elsevier, 2018.

BEAUXIS-AUSSALET, EMAL; HARDMAN, L. Visualization of confusion matrix for non-expert users (poster). IEEE, 2014.

CAVALCANTE, Rodolfo C. *et al.* Computational intelligence and financial markets: A survey and future directions. *Expert Systems with Applications*, v. 55, p. 194–211, 2016. ISSN 0957-4174. Disponível em: <https://www.sciencedirect.com/science/article/pii/S095741741630029X>.

CHINCHOR, Nancy; SUNDHEIM, Beth M. Muc-5 evaluation metrics. In: *Fifth Message Understanding Conference (MUC-5): Proceedings of a Conference Held in Baltimore, Maryland, August 25-27, 1993*. [S.l.: s.n.], 1993.

COLLIER, Robin. An historical overview of natural language processing systems that learn. *Artificial Intelligence Review*, Springer, v. 8, n. 1, p. 17–54, 1994.

DAUDERT, Tobias; AHMADI, Sina. Cofif: A corpus of financial reports in french language. In: *Proceedings of the First Workshop on Financial Technology and Natural Language Processing*. [S.l.: s.n.], 2019. p. 21–26.

EL-HAJ, Mahmoud *et al.* Detecting document structure in a very large corpus of uk financial reports. European Language Resources Association (ELRA), 2014.

FELDMAN, Ronen; UNGAR, Lyle. *Sentiment Mining from User Generated Content*. 2014. <https://www.cis.upenn.edu/~ungar/AAAI/AAAI14.pdf>. Accessed: 2021–08-26.

FREITAS, Cinthia OA *et al.* Confusion matrix disagreement for multiple classifiers. In: SPRINGER. *Iberoamerican Congress on Pattern Recognition*. [S.l.], 2007. p. 387–396.

GUDIVADA, Venkat N.; RAO, Dhana; RAGHAVAN, Vijay V. Chapter 9 - big data driven natural language processing research and applications. In: GOVINDARAJU, Venu; RAGHAVAN, Vijay V.; RAO, C.R. (Ed.). *Big Data Analytics*. Elsevier, 2015, (Handbook of Statistics, v. 33). p. 203–238. Disponível em: <https://www.sciencedirect.com/science/article/pii/B9780444634924000095>.

ICHSANI, Sakina; SUHARDI, Agatha Rinta. The effect of return on equity (roe) and return on investment (roi) on trading volume. *Procedia-Social and Behavioral Sciences*, Elsevier, v. 211, p. 896–902, 2015.

JONES, K Sparck. Natural language processing: a historical review. *University of Cambridge*, Citeseer, p. 2–10, 2001.

LIN, Ming-Chih *et al.* Stock price movement prediction using representative prototypes of financial reports. *ACM Transactions on Management Information Systems (TMIS)*, ACM New York, NY, USA, v. 2, n. 3, p. 1–18, 2011.

LIU, Ling; ÖZSU, M Tamer. *Encyclopedia of database systems*. [S.l.]: Springer New York, NY, USA:, 2009. v. 6.

LORICA, B; LOUKIDES, M. *What is Artificial Intelligence?* 1st. ed. [S.l.]: O'Reilly Media, Inc, 2016.

LOUKIDES, Mike. *What is data science?* 1st. ed. [S.l.]: O'Reilly Media, Inc., 2011.

LUGER, George F. *Artificial intelligence: structures and strategies for complex problem solving*. 6th. ed. [S.l.]: Pearson education, 2008.

MARTÍNEZ-PLUMED, Fernando *et al.* Crisp-dm twenty years later: From data mining processes to data science trajectories. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, 2019.

MOTE, Kevin. *Natural Language Processing - A Survey*. 2012.

ORENGO, Viviane Moreira; HUYCK, Christian R. A stemming algorithmm for the portuguese language. In: *spire*. [S.l.: s.n.], 2001. v. 8, p. 186–193.

OTTER, Daniel W; MEDINA, Julian R; KALITA, Jugal K. A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, IEEE, v. 32, n. 2, p. 604–624, 2020.

PASTOR, Lubos; VERONESI, Pietro. Learning in financial markets. *Annu. Rev. Financ. Econ.*, Annual Reviews, v. 1, n. 1, p. 361–381, 2009.

PILBEAM, Keith. *Finance & financial markets*. [S.l.]: Macmillan International Higher Education, 2018.

RUSSELL, Stuart J; NORVIG, Peter. *Artificial intelligence: a modern approach*. 3rd. ed. [S.l.]: Prentice Hall Press, 2009.

SAMMUT, Claude; WEBB, Geoffrey I. *Encyclopedia of machine learning*. [S.l.]: Springer Science & Business Media, 2011.

SANTOS, Ronnie ES *et al.* Técnicas de processamento de linguagem natural aplicadas ao processo de mineração de textos: resultados preliminares de um mapeamento sistemático. *Revista de Sistemas e Computação-RSC*, v. 4, n. 2, 2015.

SARKAR, Dipanjan (DJ). *A Practitioner's Guide to Natural Language Processing (Part I) — Processing & Understanding Text*. 2018. <https://towardsdatascience.com/a-practitioners-guide-to-natural-language-processing-part-i-processing-understanding-text-9f4abfd13e72>. Acessado: 2021-08-26.

SASAKI, Yutaka *et al.* The truth of the f-measure. 2007. *URL: https://www. cs. odu. edu/˜ mukka/cs795sum09dm/Lecturenotes/Day3/F-measure-YS-26Oct07. pdf [accessed 2021-08-26]*, 2007.

SINGH, Jasmeet; GUPTA, Vishal. A systematic review of text stemming techniques. *Artificial Intelligence Review*, Springer, v. 48, n. 2, p. 157–217, 2017.

SUDHAMS, Tarun; SARIPALLI, Varun Vamsi. Understanding financial reports using natural language processing. 2019.

WET, JHVH De; TOIT, Elsa Du. Return on equity: A popular, but flawed measure of corporate financial performance. *South African Journal of Business Management*, AOSIS, v. 38, n. 1, p. 59–69, 2007.

WIJERATNE, Yudhanjaya; SILVA, Nisansa de; SHANMUGARAJAH, Yashothara. Natural language processing for government: Problems and potential. *International Development Research Centre (Canada)*, 2019.

XUE, Nianwen; BIRD, Edward *et al.* Natural language processing with python. *Natural Language Engineering*, Cambridge University Press, v. 17, n. 3, p. 419, 2011.

ZELGALVE, Irina Berzkalne-Elvira. Return on equity and company characteristics: An empirical study of industries in latvia. *The 8th International Days of Statistics and Economics, Prague*, 2014.

ZHANG, Dell; WANG, Jun; ZHAO, Xiaoxue. Estimating the uncertainty of average f1 scores. In: *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*. [S.l.: s.n.], 2015. p. 317–320.

ZHANG, Yao. *Using financial reports to predict stock market trends with machine learning techniques*. Tese (Doutorado) — University of Oxford, 2015.

Apêndices

APÊNDICE A – Artigo sobre o trabalho

Cálculo de Indicadores Financeiros com Auxílio do Processamento de Linguagem Natural

Lucas João Martins[1]

[1] Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brasil

lucasjoao.lj@gmail.com

Abstract. *Natural language processing can be considered a domain that connects several areas, some of which belong to the field of artificial intelligence. Meanwhile, the financial market has been constantly growing in recent years. The present work involves joining natural language processing to the financial market with the intention of helping in the execution of a task - knowing a company's financial indicators. The association of these two fields is made in the work by proposing a model that aims to calculate financial indicators from business reports with the aid of natural language processing.*

Resumo. *O processamento de linguagem natural pode ser considerado um domínio que abrange diversas áreas, sendo algumas delas pertencentes ao campo da inteligência artificial. Enquanto isso, o mercado financeiro está em constante crescimento nos últimos anos. O presente trabalho busca juntar o processamento de linguagem natural ao mercado financeiro com a intenção de ajudar na execução de uma tarefa - saber indicadores financeiros de uma empresa. A associação desses dois campos é feita no trabalho ao propor um modelo que tem como objetivo calcular indicadores financeiros a partir de relatórios empresariais com o auxílio do processamento de linguagem natural.*

1. Introdução

Os avanços tecnológicos e a disponibilidade cada vez maior de diferentes tipos de dados mostram que o campo de processamento de linguagem natural (PLN) possui diversas possibilidades de estudo, com uma atestada relevância na academia e na indústria. Já o mercado financeiro está em constante evolução, com mudanças muitas vezes proporcionadas pela disponibilidade de novas tecnologias e que impactam a forma com que empresas e consumidores se relacionam com a área. Sendo assim, este trabalho utilizará técnicas de PLN para calcular indicadores financeiros com base em informações (textos não estruturados) provenientes de relatórios de resultados empresariais, os quais representam a saúde financeira da empresa e por isso ajudam na tomada de decisão no mercado financeiro ao fazer parte de uma análise fundamentalista da instituição.

2. Fundamentação teórica

Como pode ser encontrado em Liu e Özsu (2009), *stemming* é o processo no qual uma palavra possui seu final ou outros afixos removidos ou modificados para que diferentes formas daquelas palavras que diferem de um modo não relevante possam ser consideradas iguais. Singh e Gupta (2017) dão um exemplo disso, as palavras em inglês *maintaining*, *maintained*, *maintenance* são transformadas na palavra raiz *maintain* através do *stemming*. Liu e Özsu (2009) definem que um programa de computador que aplica essa transformação na palavra é chamado de *stemmer* ou de algoritmo de *stemming*. Além disso, o resultado de um algoritmo de *stemming* é conhecido como *stem*. Singh e Gupta (2017) consideram o *stemming* com uma entre as numerosas técnicas de pré-processamento que são úteis nas áreas de recuperação de informação e processamento de linguagem natural.

Zelgalve (2014) fala que o retorno sobre patrimônio líquido diz a porcentagem de lucro que uma empresa faz para cada unidade de dinheiro do patrimônio líquido da empresa. Enquanto Ichsani e Suhardi (2015) afirmam que o ROE é a razão utilizada para mensurar o sucesso de uma empresa em gerar lucro para os seus acionistas. Wet e Toit (2007) já afirmavam que o ROE é um do mais antigo e talvez mais utilizado, entre analistas financeiros e acionistas, indicador para mensurar o desempenho de uma empresa - informação que corroboram após apresentar essa afirmação feita por outros autores da área.

3. Modelo que calcula indicadores financeiros

3.1. *Dataset*

O *dataset* é composto por arquivos no formato PDF que representam as demonstrações de resultados financeiros trimestrais divulgados por empresas que estão listadas na B3, a bolsa de valores brasileira. Apesar da similaridade no conteúdo dos relatórios de diferentes empresas, esses arquivos, no geral, não apresentam um padrão comum de *layout* e informações disponíveis.

Tabela 1. Relatórios financeiros escolhidos para compor o *dataset*

Empresa	Período relatório
Ambev	2019 - 1T (primeiro trimestre)
ENGIE	2019 - 2T (segundo trimestre)
ENGIE	2020 - 2T (segundo trimestre)
Fleury	2019 - 3T (terceiro trimestre)
Fleury	2020 - 2T (segundo trimestre)
Gerdau	2015 - 3T (terceiro trimestre)
Gerdau	2017 - 1T (primeiro trimestre)
Gerdau	2018 - 4T (quarto trimestre)
Gerdau	2019 - 2T (segundo trimestre)
WEG	2010 - 2T (segundo trimestre)
WEG	2015 - 1T (primeiro trimestre)
WEG	2017 - 2T (segundo trimestre)
WEG	2019 - 2T (segundo trimestre)

3.2. Extração do texto

A extração do texto corresponde ao primeiro item do fluxo de trabalho. No código[1] o encarregado dessa tarefa é a classe *pdf_extract.py*, ela possui três métodos, dois públicos e um privado. Um método público para cada biblioteca de extração de texto utilizada no trabalho: *PyPDF2* e *pytesseract*. Enquanto que o método privado foi necessário para encapsular uma pequena lógica que determina o diretório do arquivo no formato PDF que terá seu texto extraído.

3.3. Pré-processamento

Já o pré-processamento corresponde ao segundo item do fluxo de trabalho. Ele realiza as seguintes atividades: *string* do texto dividida em sentenças, as sentenças do texto são *tokenizadas* e ocorre a remoção de *stopwords*. Todas essas etapas se encontram na classe *preprocessor.py* e são realizadas pelo método *execute* com a ajuda da biblioteca *nltk*.

O método execute recebe o texto em *string* do arquivo em formato PDF coletado na etapa anterior e retorna uma espécie de matriz (lista de listas). Nessa estrutura, o número de linhas representa o número de sentenças no texto e o número de colunas será a quantidade de tokens da maior sentença presente no texto.

3.4. Técnica: *stemming*

O *stemming* é uma técnica de processamento de linguagem natural utilizada no presente trabalho. Essa etapa corresponde ao terceiro item do fluxo de trabalho. No código, a classe *stemming.py* é a responsável pela realização dessa fase com a ajuda da biblioteca *nltk* e através dos métodos *stem_word* e *stem_text_matrix*.

Na instanciação de um objeto da classe *stemming* define-se que o algoritmo utilizado para a realização do *stemming* é o *RSLP Stemmer*, onde *RSLP* é a sigla para Removedor de Sufixos da Língua Portuguesa. Esse algoritmo foi apresentado no artigo *A Stemming Algorithm for the Portuguese Language*, de 2001, escrito por Moreira Orengo e Christian Huyck, e, possui, segundo o artigo, melhores resultados do que o stemmer *Snowball* escrito por Martin Porter na sua versão em português. Por essa razão e também por ser o algoritmo recomendado pelo *nltk* para o português que se escolheu o *RSLP*.

3.5. Indicadores financeiros

A parte referente aos indicadores financeiros não corresponde a nenhuma etapa no fluxo de trabalho, já que ela serve principalmente para definir o domínio de interesse da busca que será realizada durante a filtragem na execução. No código, esse domínio é especificado no diretório *src/indicator*, onde cada indicador trabalhado possui uma classe em um arquivo. Logo, há três arquivos: *lucro_liquido.py*, *patrimonio_liquido.py* e *roe.py*.

[1] https://github.com/lucasjoao/tcc

3.6. Execução

O item com nome de execução do presente trabalho corresponde ao código que é invocado diretamente pelos experimentos. Esse código utiliza de tudo que já foi apresentado até então para buscar em uma forma específica o indicador financeiro solicitado no relatório com os resultados financeiros da empresa. A execução é dividida em duas grandes fases: a filtragem e as buscas possíveis.

A fase de filtragem, uma parte da execução, corresponde ao quarto item do fluxo de trabalho. Ela corresponde às seguintes etapas do processo: sentenças são filtradas por domínio de interesse, remoção de possíveis falso-positivos e verificação da ordem dos itens do domínio de interesse. No código, a classe *filters.py* é quem realiza essa etapa por meio dos métodos *candidate_sentences* e *is_searcher_words_in_sequence*.

Já a etapa de buscas possíveis, a outra parte da execução, corresponde ao quinto e último item do fluxo de trabalho. Corresponde às seguintes etapas nas duas ramificações: 1. Busca por valor monetário + remoção de possíveis falso-positivos + remoção de resultados duplicados; 2. Busca por valor numérico + remoção de resultados duplicados. A classe *searcher.py* é a responsável no código por realizar essas atividades e faz isso com a ajuda dos métodos *monetary_value* e *after_target_set_number_value*. Assim, é possível concluir que há duas buscas possíveis no presente trabalho, uma busca por valores monetários e outra busca por valores numéricos.

4. Avaliação do modelo

4.1. Métricas utilizadas para avaliação

Para possibilitar uma melhor avaliação do modelo, algumas métricas serão obtidas a partir da execução dos experimentos. Dessa forma, é possível mensurar a qualidade dos resultados obtidos e também fica mais fácil de comparar quantitativamente um experimento com o outro. As seguintes métricas serão discutidas: matriz de confusão; acurácia; *recall*; precisão; *F1-score*; resultados que consideram unicidade; tempo.

4.2. Experimentos

Os experimentos no presente trabalho foram realizados através de testes unitários. Cada conjunto de testes representa uma definição de configuração para a plataforma, ou seja, uma possível experimentação de determinada especificação que possivelmente impacta nos resultados esperados. Ao serem realizadas as experimentações, algumas métricas foram coletadas e também verificou-se a corretude do resultado obtido através de comparação com uma tabela verdade dos resultados.

Tabela 2. Especificação dos experimentos realizados

Nome experimento	Configuração relevante utilizada
Extração com *PyPDF2* #0	*out of the box* da versão utilizada
Extração com *pytesseract* #0	*pytesseract* com parâmetro $tessedit\_do\_invert = 0$
Extração com *pytesseract* #1	*pytesseract* com parâmetros $tessedit\_do\_invert = 0$, $psm = 10$ e $oem = 2$
Extração com *pytesseract* #2	*pytesseract* com parâmetros $tessedit\_do\_invert = 0$ e $psm = 6$
Extração com *pytesseract* #3	*pytesseract* com parâmetros $tessedit\_do\_invert = 0$ e $psm = 5$

4.3. Análise dos resultados

É possível concluir que os resultados por parte dos experimentos podem ser considerados medianos. Ao fazer uma interpretação sem considerar os valores de *F1-score* zerados, ambos casos especiais, os experimentos tiveram um resultado entre 0,31 e 0,56, ou seja, não recomenda-se confiar completamente na plataforma para realizar a busca de algum indicador financeiro. Ao decorrer do desenvolvimento do trabalho percebeu-se alguns fatores que levaram a esse tipo de resultado: a falta de padronização dos relatórios financeiros, e, o alto número de tabelas nos relatórios financeiros.

Tabela 3. Resultado - *F1-score*

Nome experimento	***F1-score***
Extração com PyPDF2 #0	0,31
Extração com pytesseract #0	0,46
Extração com pytesseract #1	0,00
Extração com pytesseract #2	0,56
Extração com pytesseract #3	0,00

Todas as métricas comprovam que o melhor experimento foi o **Extração com *pytesseract* #2**, que considera cada página dos relatórios financeiros como um único bloco de texto, já que ele possui os melhores resultados em todas as avaliações. Por outro lado, a diferença de valores que faz seu desempenho ser melhor ao ser comparado com o segundo lugar dos experimentos, **Extração com *pytesseract* #0** - a versão out of box do *pytesseract*, é baixa, pois por exemplo, trata-se somente de 0,1 a mais de *F1-score*. Esse fato gera uma linha de raciocínio interessante: a configuração adequada das bibliotecas utilizadas podem melhorar o desempenho da plataforma, mas para o atual escopo do trabalho, suas versões out of box possuem um desempenho razoável, e, dependendo do tipo de configuração realizada, os resultados podem piorar drasticamente (vide **Extração com *pytesseract* #1** e **Extração com *pytesseract* #3**). Portanto, a configuração tem que ser realizada com cautela.

Além disso, o *pytesseract,* quando configurado adequadamente, possui resultados melhores do que o *PyPDF2*, mas há um preço relativo ao tempo de processamento pago para conseguir métricas superiores. O tempo de execução dos experimentos que deram bons resultados com *pytesseract* é cerca de 90 vezes maior do que a execução com *PyPDF2*, que leva menos de 1 minuto para acontecer.

5. Conclusão

É importante perceber após uma averiguação do objetivo definido inicialmente que o mesmo foi cumprido. No decorrer da execução o escopo do estudo foi restrito ao cálculo de um indicador devido a percepção adquirida de que para se calcular um indicador financeiro teria que se encontrar as variáveis que permitem o cálculo no próprio relatório. Sendo assim, a pesquisa mirou em buscar indicadores financeiros e também em calcular um indicador financeiro a partir do resultado das buscas. É importante perceber que uma vantagem de também se realizar a busca, é que em alguns casos (devido a falta de padrão entre os relatórios), o indicador financeiro que deseja ser calculado pode estar presente no documento e ser encontrado ao invés de ser calculado a partir de outros.

Referências

LIU, Ling; ÖZSU, M Tamer. Encyclopedia of database systems. [S.l.]: Springer New York, NY, USA:, 2009. v. 6.

SINGH, Jasmeet; GUPTA, Vishal. A systematic review of text stemming techniques. Artificial Intelligence Review, Springer, v. 48, n. 2, p. 157–217, 2017.

ZELGALVE, Irina Berzkalne-Elvira. Return on equity and company characteristics: An empirical study of industries in latvia. The 8th International Days of Statistics and Economics, Prague, 2014.

ICHSANI, Sakina; SUHARDI, Agatha Rinta. The effect of return on equity (roe) and return on investment (roi) on trading volume. Procedia-Social and Behavioral Sciences, Elsevier, v. 211, p. 896–902, 2015.

WET, JHVH De; TOIT, Elsa Du. Return on equity: A popular, but flawed measure of corporate financial performance. South African Journal of Business Management, AOSIS, v. 38, n. 1, p. 59–69, 2007.

APÊNDICE B – Código desenvolvido

repositorio/data/data.py

```
# NÃO ESTÁ COMPLETO E NEM GARANTO CERTEZA EM TODOS OS VALORES

LUCRO_LIQUIDO = {
                    'ambev_2019_1T.pdf': 2749.1,
                    'engie_2019_2T.pdf': 385.4,
                    'engie_2020_2T.pdf': 765.8,
                    'fleury_2019_3T.pdf': 94.8,
                    'fleury_2020_2T.pdf': -73.3,
                    'gerdau20153T': -193,
                    'gerdau_2017_1T.pdf': 824.0,
                    'gerdau_2018_4T.pdf': 389.0,
                    'gerdau_2019_2T.pdf': 373.0,
                    'weg_2010_2T.pdf': 116.1,
                    'weg_2015_1T.pdf': 245.9,
                    'weg_2017_2T.pdf': 272.2,
                    'weg_2019_2T.pdf': 389.0
                }

PATRIMONIO_LIQUIDO = {
                        'ambev_2019_1T.pdf': 60490.0,
                        'engie_2019_2T.pdf': 7194673.0,
                        'engie_2020_2T.pdf': 7434743,
                        'fleury_2019_3T.pdf': 1750.1,
                        'fleury_2020_2T.pdf': 1549453.0,
                        'gerdau_2015_3T.pdf': 36012381.0,
                        'gerdau_2017_1T.pdf': 24916345.0,
                        'gerdau_2018_4T.pdf': 25938571.0,
                        'gerdau_2019_2T.pdf': 26568445.0,
                        'weg_2010_2T.pdf': 2445405.0,
                        'weg_2015_1T.pdf': 5302661.0,
                        'weg_2017_2T.pdf': 6348046.0,
                        'weg_2019_2T.pdf': 8017
                       }

ROE = {
       'ambev_2019_1T.pdf': 4.566043974210613,
```

```
        'engie_2019_2T.pdf': 0.005356741022142354,
        'engie_2020_2T.pdf': 0.010300288792766609,
        'fleury_2019_3T.pdf': 5.416833323810068,
        'fleury_2020_2T.pdf': -0.004730701737968173,
        'gerdau20153T': -0.53,
        'gerdau_2017_1T.pdf': 3.33,
        'gerdau20184T': 1.49,
        'gerdau_2019_2T.pdf': 1.403921080063210,
        'weg_2010_2T.pdf': 0.004747679832175038,  # diff from
           fundamentus.com.br
        'weg_2015_1T.pdf': 4.63,
        'weg_2017_2T.pdf': 0.0042879336413126174,
        'weg_2019_2T.pdf': 4.85
        }
```

repositorio/main.py

```
import nltk
from src.plataform import preprocessor as pp
from src.technique import stemming as st
from src.plataform import filters as f
from src.plataform import searcher as s

nltk.download('rslp')
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('words')

preprocessor = pp.preprocessor()
stemmer = st.stemming()
filter = f.filters()
searcher = s.searcher()
target_sets = [frozenset([stemmer.stem_word('abacate')])]

text_example = 'Abacate 10 ' \
               'Tangerina 20 ' \
               'Abacate maduro 30'

text_example_preprocessed = preprocessor.execute(text_example)
print(text_example_preprocessed)

text_example_stemmed =
```

```
    stemmer.stem_text_matrix(text_example_preprocessed)
print(text_example_stemmed)

text_example_filtered =
    filter.candidate_sentences(text_example_stemmed, target_sets)
print(text_example_filtered)

text_example_filtered_in_seq =
    filter.is_searcher_words_in_sequence(text_example_filtered,
    target_sets)
print(text_example_filtered_in_seq)

example_monetary_value =
    searcher.monetary_value(text_example_filtered_in_seq)
print(example_monetary_value)

example_after_target_set =
    searcher.after_target_set_number_value(text_example_filtered_in_seq,
    target_sets)
print(example_after_target_set)
```

repositorio/makefile

```
main:
  @poetry run python src/main.py

test-all:
  @poetry run python  -m unittest

test-helper:
  @poetry run python  -m unittest discover ./tests/helper/

test-plataform:
  @poetry run python  -m unittest discover ./tests/plataform/

test-report:
  @poetry run python  -m unittest discover ./tests/reports/

test-pypdf2:
  @poetry run python  -m unittest discover
     ./tests/reports/pypdf2/

```

```
test-pytesseract-default:
  @poetry run python  -m unittest discover
     ./tests/reports/pytesseract/

test-pytesseract-config01:
  @poetry run python  -m unittest discover
     ./tests/reports/pytesseract01/

test-pytesseract-config02:
  @poetry run python  -m unittest discover
     ./tests/reports/pytesseract02/

test-pytesseract-config03:
  @poetry run python  -m unittest discover
     ./tests/reports/pytesseract03/

linter:
  @poetry run flake8 src tests data --count --max-complexity=10
     --max-line-length=127 --statistics
```

repositorio/pyproject.toml

```
[tool.poetry]
name = "tcc"
version = "0.1.0"
description = ""
authors = ["lucas <lucasjoao.lj@gmail.com>"]
license = "MIT"

[tool.poetry.dependencies]
python = "^3.7"
PyPDF2 = "^1.26"
nltk = "^3.5"
numpy = "^1.19"
pytesseract = "^0.3.6"
pdf2image = "^1.14"

[tool.poetry.dev-dependencies]
autopep8 = "^1.4"
flake8 = "^3.7"
[build-system]
requires = ["poetry>=0.12"]
```

```
build-backend = "poetry.masonry.api"
```

repositorio/src/helper/list_helper.py

```
class list_helper:

    @staticmethod
    def remove_duplicates_list_of_lists(list_of_lists):
        return list(set(map(tuple, list_of_lists)))

    @staticmethod
    def remove_duplicates_list_of_dicts(list_of_dicts):
        # thanks
           https://stackoverflow.com/questions/9427163/remove-duplicate-dic
        return [dict(t) for t in {tuple(d.items()) for d in
           list_of_dicts}]
```

repositorio/src/helper/number_helper.py

```
import math
import re


class number_helper:

    @staticmethod
    def is_number(string):
        commas_number = string.count(',')
        if commas_number > 1:
            return (False, math.nan)
        else:
            return
               number_helper.__is_number_with_at_least_one_dot(string)

    @staticmethod
    def __is_number_with_at_least_one_dot(string):
        string = string.replace(',', '.')
        if string.count('.') <= 1:
            try:
                number = float(string)
                return (True, number)
            except ValueError:
                return (False, math.nan)
```

```
24          else:
25              return
                    number_helper.__is_number_with_more_dots(string)
26
27      @staticmethod
28      def __is_number_with_more_dots(string):
29          try:
30              if re.fullmatch(r'.*\.\d{3}', string) is not None:
31                  number = float(string.replace('.', ''))
32                  return (True, number)
33              else:
34                  raise ValueError
35          except ValueError:
36              return
                    number_helper.__is_number_with_more_dots_and_decimal(string)
37
38      @staticmethod
39      def __is_number_with_more_dots_and_decimal(string):
40          try:
41              if re.fullmatch(r'.*\.\d{1,2}', string) is not None:
42                  dots_number = string.count('.')
43                  number = float(string.replace('.', '',
                        dots_number - 1))
44                  return (True, number)
45              else:
46                  raise ValueError
47          except ValueError:
48              return (False, math.nan)
```

repositorio/src/helper/print_helper.py

```
1  class print_helper:
2
3      @staticmethod
4      def sentence_viewer(sentences):
5          """
6          Args:
7              sentences é uma matriz de sentenças por tokens
8          """
9          for sentence in sentences:
10             print(" ".join(sentence))
11             print("\n")
```

```
12
13      @staticmethod
14      def print_line():
15          print(80 * '-')
```

repositorio/src/helper/result_helper.py

```
1  from src.helper import number_helper as nh
2
3
4  class result_helper:
5
6      __undefined = 'undefined'
7
8      @staticmethod
9      def create_result_item(number, possible_size):
10         normalized_possible_size = possible_size
11         was_casted, _ = nh.number_helper.is_number(possible_size)
12
13         if was_casted:
14             normalized_possible_size = result_helper.__undefined
                   # not None here solve some problems with types
15
16         return {'number': number, 'possible_size':
              normalized_possible_size}
17
18     @staticmethod
19     def clean_search_result(dirty_result):
20         clean_result = []
21         for dict_result in dirty_result:
22             if dict_result['possible_size'] !=
                  result_helper.__undefined:
23                 clean_result.append(dict_result)
24         return clean_result
25
26     @staticmethod
27     def get_numbers_as_list(result_item_list):
28         numbers_result = []
29         for result_item in result_item_list:
30             for key, number in result_item.items():
31                 if key == 'number':
32                     numbers_result.append(number)
```

```
33          return numbers_result
```

repositorio/src/indicator/lucro_liquido.py

```
1  from src.technique import stemming as s
2
3
4  class lucro_liquido:
5
6      def __init__(self):
7          self.stemmer = s.stemming()
8
9      def get_target_sets(self):
10         return [frozenset([self.stemmer.stem_word('lucro'),
               self.stemmer.stem_word('líquido')])]
11
12     def get_filter_sets(self):
13         return [frozenset([self.stemmer.stem_word('imposto'),
               self.stemmer.stem_word('renda')]),
14                 frozenset([self.stemmer.stem_word('receita'),
                       self.stemmer.stem_word('operacional'),
15                             self.stemmer.stem_word('líquida')]),
16                 frozenset([self.stemmer.stem_word('wege3')]),
17                 frozenset([self.stemmer.stem_word('ativos'),
                       self.stemmer.stem_word('fixos')]),
18                 frozenset([self.stemmer.stem_word('investimentos')]),
19                 frozenset([self.stemmer.stem_word('variação')]),
20                 frozenset([self.stemmer.stem_word('dividendos')]),
21                 frozenset([self.stemmer.stem_word('dívida')]),
22                 frozenset([self.stemmer.stem_word('imposto')]),
23                 frozenset([self.stemmer.stem_word('recuperação')])]
```

repositorio/src/indicator/patrimonio_liquido.py

```
1  from src.technique import stemming as s
2
3
4  class patrimonio_liquido:
5
6      def __init__(self):
7          self.stemmer = s.stemming()
8
9      def get_target_sets(self):
```

```
10          return [frozenset([self.stemmer.stem_word('patrimônio'),
                self.stemmer.stem_word('líquido')])]
11
12      def get_filter_sets(self):
13          return [frozenset([])]
```

repositorio/src/indicator/roe.py

```
1  from src.technique import stemming as s
2
3
4  class roe:
5
6      def __init__(self):
7          self.stemmer = s.stemming()
8
9      def get_target_sets(self):
10         return [frozenset(['roe']),
11                 frozenset([self.stemmer.stem_word('retorno'),
                       self.stemmer.stem_word('sobre'),
12                             self.stemmer.stem_word('patrimônio'),
                                  self.stemmer.stem_word('líquido')])]
13
14     def get_filter_sets(self):
15         return [frozenset([])]
16
17     def calculate(self, lucro_liquido, patrimonio_liquido):
18         # FIXME: considerar escala na hora de calcular
19         return (lucro_liquido / patrimonio_liquido) * 100
20
21     def calculate_iterating(self, lucro_liquido_result_list,
           patrimonio_liquido_result_list):
22         result = []
23         for lucro_liquido in lucro_liquido_result_list:
24             for patrimonio_liquido in
                   patrimonio_liquido_result_list:
25                 result.append(self.calculate(lucro_liquido['number'],
                       patrimonio_liquido['number']))
26         return result
```

repositorio/src/manager.py

```
1  import nltk
```

```
2  from src.indicator import roe as roe
3  from src.indicator import lucro_liquido as ll
4  from src.indicator import patrimonio_liquido as pl
5  from src.plataform import pdf_extract as pe
6  from src.plataform import preprocessor as pp
7  from src.plataform import filters as f
8  from src.plataform import searcher as se
9  from src.helper import result_helper as rh
10 from src.helper import list_helper as lh
11 from src.technique import stemming as st
12
13
14 class manager:
15
16     def __init__(self, reports_filename,
       text_extract_lib='pypdf2', custom_config_extract_lib=' '):
17         nltk.download('rslp')
18         nltk.download('punkt')
19         nltk.download('stopwords')
20         nltk.download('words')
21
22         preprocessor = pp.preprocessor()
23         stemming = st.stemming()
24
25         self.lucro_liquido = ll.lucro_liquido()
26         self.patrimonio_liquido = pl.patrimonio_liquido()
27         self.roe = roe.roe()
28
29         self.filters = f.filters()
30         self.searcher = se.searcher()
31
32         self.reports_stemming = {}
33         for report_filename in reports_filename:
34             pdf_text = self.__get_pdf_text(report_filename,
                text_extract_lib, custom_config_extract_lib)
35             preprocessed_text = preprocessor.execute(pdf_text)
36             self.reports_stemming[report_filename] =
                stemming.stem_text_matrix(preprocessed_text)
37
38     def __get_pdf_text(self, report_filename, text_extract_lib,
       custom_config_extract_lib):
39         pdf_text = ''
```

```
40          if text_extract_lib == 'pypdf2':
41              pdf_text =
                    pe.pdf_extract.get_text_pypdf2(report_filename)
42          elif text_extract_lib == 'pytesseract':
43              pdf_text =
                    pe.pdf_extract.get_text_pytesseract(report_filename,
                    custom_config_extract_lib)
44          return pdf_text
45
46      def __common_process(self, indicator):
47          target_sets = indicator.get_target_sets()
48
49          result = {}
50          for filename, stem_text_matrix in
                self.reports_stemming.items():
51              candidate_sentences =
                    self.filters.candidate_sentences(stem_text_matrix,
                    target_sets)
52              false_candidate_sentences =
                    self.filters.candidate_sentences(candidate_sentences,
                    indicator.get_filter_sets())
53              candidate_sentences = [sentence for sentence in
                    candidate_sentences if sentence not in
                    false_candidate_sentences]
54              candidate_sentences =
                    self.filters.is_searcher_words_in_sequence(candidate_sentenc
                    target_sets)
55              result[filename] = candidate_sentences
56          return result
57
58      def __common_process_monetary(self, indicator):
59          reports_candidate_sentences =
                self.__common_process(indicator)
60
61          result = {}
62          for filename, candidate_sentences in
                reports_candidate_sentences.items():
63              dirty_result =
                    self.searcher.monetary_value(candidate_sentences)
64              result_with_duplicate =
                    rh.result_helper.clean_search_result(dirty_result)
65              result[filename] =
```

```
                lh.list_helper.remove_duplicates_list_of_dicts(result_with_dupli
66          return result
67
68      def __common_process_number(self, indicator):
69          reports_candidate_sentences =
               self.__common_process(indicator)
70
71          target_sets = indicator.get_target_sets()
72
73          result = {}
74          for filename, candidate_sentences in
               reports_candidate_sentences.items():
75              result_with_duplicate =
                   self.searcher.after_target_set_number_value(candidate_sentences,
                   target_sets)
76              result[filename] =
                   lh.list_helper.remove_duplicates_list_of_dicts(result_with_dupli
77          return result
78
79      def run_lucro_liquido_monetary(self):
80          return self.__common_process_monetary(self.lucro_liquido)
81
82      def run_lucro_liquido_number(self):
83          return self.__common_process_number(self.lucro_liquido)
84
85      def run_patrimonio_liquido_monetary(self):
86          return
               self.__common_process_monetary(self.patrimonio_liquido)
87
88      def run_patrimonio_liquido_number(self):
89          return
               self.__common_process_number(self.patrimonio_liquido)
90
91      def run_roe_monetary(self):
92          return self.__common_process_monetary(self.roe)
93
94      def run_roe_number(self):
95          return self.__common_process_number(self.roe)
96
97      def run_calculate_roe(self):
98          lucro_liquido_number = self.run_lucro_liquido_number()
99          lucro_liquido_monetary =
```

```
            self.run_lucro_liquido_monetary()
100         patrimonio_liquido_number =
               self.run_patrimonio_liquido_number()
101         patrimonio_liquido_monetary =
               self.run_patrimonio_liquido_monetary()
102
103         result = {}
104         for filename in self.reports_stemming.keys():
105             result_file = []
106             result_file +=
                   self.roe.calculate_iterating(lucro_liquido_number[filename]
                   patrimonio_liquido_number[filename])
107             result_file +=
                   self.roe.calculate_iterating(lucro_liquido_number[filename]
                   patrimonio_liquido_monetary[filename])
108             result_file +=
                   self.roe.calculate_iterating(lucro_liquido_monetary[filenam
                   patrimonio_liquido_number[filename])
109             result_file +=
                   self.roe.calculate_iterating(lucro_liquido_monetary[filenam
110                                                          patrimonio_liquido
111             result[filename] = result_file
112         return result
```

repositorio/src/plataform/data_dir_scan.py

```
1  import os
2
3
4  class data_dir_scan:
5
6      __src = 'src'
7      __tests = 'tests'
8      __data = 'data'
9
10     @staticmethod
11     def get_data_directory():
12         current_directory = os.getcwd()
13         directory_splitted = current_directory.split('/')
14
15         if data_dir_scan.__data == directory_splitted[-1]:  #
               last element from list
```

```
            return './'

        if 'tcc' == directory_splitted[-1]:
            return data_dir_scan.__data + '/'

        directory_founded = None
        if data_dir_scan.__src in directory_splitted:
            directory_founded = data_dir_scan.__src
        elif data_dir_scan.__tests in current_directory:
            directory_founded = data_dir_scan.__tests

        if directory_founded is None:
            raise Exception('Diretório inválido!')

        position = directory_splitted.index(directory_founded)
        levels_below = len(directory_splitted) - position
        return ('../' * levels_below) + data_dir_scan.__data +
           '/'
```

repositorio/src/plataform/filters.py

```
from src.helper import list_helper as lh


class filters:

    def candidate_sentences(self, text, target_sets):
        candidates = []
        for sentence in text:
            for target_set in target_sets:
                empty_target_set = len(target_set) == 0
                if not empty_target_set and
                   target_set.issubset(frozenset(sentence)):
                    candidates.append(sentence)
        return candidates

    def is_searcher_words_in_sequence(self, candidates,
       target_sets):
        candidates_filtered = []
        for sentence in candidates:
            for target_set in target_sets:
                first_searcher_element, *_ = target_set
```

```
                target_set_size = len(target_set)
                positions = [i for i, token in
                   enumerate(sentence) if token ==
                   first_searcher_element]

                for position in positions:
                    hits = 1
                    for i in range(1, target_set_size):
                        try:
                            is_hit = sentence[position + i] in
                               target_set or sentence[position -
                               i] in target_set
                        except IndexError:
                            is_hit = False
                        if is_hit:
                            hits += 1

                    if target_set_size == hits:
                        candidates_filtered.append(sentence)
        return
           lh.list_helper.remove_duplicates_list_of_lists(candidates_filter
```

repositorio/src/plataform/pdf_extract.py

```
import PyPDF2
import pytesseract

from pdf2image import convert_from_path
from src.plataform import data_dir_scan as dds


class pdf_extract:

    @staticmethod
    def get_text_pypdf2(filename):
        pdf_reader =
           PyPDF2.PdfFileReader(pdf_extract.__path_from_filename(filename))

        pdf_text = ''
        for i in range(pdf_reader.numPages):
            pdf_page = pdf_reader.getPage(i)
            pdf_text += pdf_page.extractText()
```

```
18
19          return pdf_text
20
21      @staticmethod
22      def __path_from_filename(filename):
23          return dds.data_dir_scan.get_data_directory() + filename
24
25      @staticmethod
26      def get_text_pytesseract(filename, custom_config_extact_lib):
27          pages =
               convert_from_path(pdf_extract.__path_from_filename(filename),
               500)
28
29          config_to_speed_up = '-c tessedit_do_invert=0'
30          config = config_to_speed_up + ' ' +
               custom_config_extact_lib
31
32          pdf_text = ''
33          for page in pages:
34              pdf_text = pdf_text + ' ' +
                   pytesseract.image_to_string(page, lang='por',
                   config=config)
35
36          return pdf_text
```

repositorio/src/plataform/preprocessor.py

```
1  from nltk.tokenize import word_tokenize, sent_tokenize
2  from nltk.corpus import stopwords
3
4
5  class preprocessor:
6
7      __pt_br = 'portuguese'
8
9      def execute(self, text):
10         text_sentences = sent_tokenize(text, self.__pt_br)
11         text_sentences_in_tokens = [word_tokenize(sentence) for
               sentence in text_sentences]
12
13         stopwords_pt_br = stopwords.words(self.__pt_br)
14         preprocess_result = []
```

```
15
16          for sentence in text_sentences_in_tokens:
17              sentence_result = [token for token in sentence if
                    token not in stopwords_pt_br]
18              preprocess_result.append(sentence_result)
19
20          return preprocess_result
```

repositorio/src/plataform/searcher.py

```
1  from src.helper import number_helper as nh
2  from src.helper import result_helper as rh
3
4
5  class searcher:
6
7      def monetary_value(self, candidate_sentences):
8          invalid_positions = frozenset([0, 1])
9
10         results = []
11         for sentence in candidate_sentences:
12             position = 0
13             for token in sentence:
14                 was_casted, number =
                       nh.number_helper.is_number(token)
15                 if was_casted and position not in
                       invalid_positions:
16                     if sentence[position - 2] == 'r' and
                           sentence[position - 1] == '$':
17                         results.append(self.__safe_create_result_item(numbe
                               sentence, position))
18
19                 position += 1
20
21         return results
22
23     def after_target_set_number_value(self, candidate_sentences,
           target_sets):
24         results = []
25         for target_set in target_sets:
26             target_set_size = len(target_set)
27             for sentence in candidate_sentences:
```

```
28                  curr_position = 0
29                  for token in sentence:
30                      possible_result = True
31
32                      was_casted, number =
                           nh.number_helper.is_number(token)
33                      if was_casted:
34
35                          for i in range(1, target_set_size + 1):
36                              lookup_position = curr_position - i
37                              if lookup_position < 0 or
                                   sentence[lookup_position] not in
                                   target_set:
38                                  possible_result = False
39                                  break
40
41                          if possible_result:
42                              results.append(self.__safe_create_result_item(numbe
                                   sentence, curr_position))
43
44                      curr_position += 1
45
46          return results
47
48      def __safe_create_result_item(self, number, sentence,
           position):
49          result_item = {}
50          try:
51              result_item =
                   rh.result_helper.create_result_item(number,
                   sentence[position + 1])
52          except IndexError:
53              result_item =
                   rh.result_helper.create_result_item(number,
                   'undefined')
54          return result_item
```

repositorio/src/technique/stemming.py

```
1 from nltk.stem import RSLPStemmer
2
3
```

```
4  class stemming:
5
6      def __init__(self):
7          self.stemmer = RSLPStemmer()
8
9      def stem_word(self, word):
10         return self.stemmer.stem(word)
11
12     def stem_text_matrix(self, text_matrix):
13         """
14         Args
15             text_matrix é o resultado da etapa de
                   pré-processamento
16         """
17         result = []
18         for sentence in text_matrix:
19             sentence_result = [self.stem_word(token) for token
                   in sentence]
20             result.append(sentence_result)
21
22         return result
```

repositorio/tests/helper/test_list_helper.py

```
1  import unittest
2
3  from src.helper import list_helper as lh
4
5
6  class TestsListHelper(unittest.TestCase):
7
8      def test_remove_duplicates_lists_ok(self):
9          list_of_lists = [['Abacate', 'Abacate', 'Granola'],
               ['Abacate', 'Granola'], ['Abacate', 'Granola']]
10         result =
               lh.list_helper.remove_duplicates_list_of_lists(list_of_lists)
11
12         self.assertEqual(len(result), 2)
13         self.assertIn(('Abacate', 'Abacate', 'Granola'), result)
14         self.assertIn(('Abacate', 'Granola'), result)
15
16     def test_remove_duplicates_lists_not_remove(self):
```

```
17          list_of_lists = [['Abacate', 'Abacate', 'Granola'],
                ['Abacate', 'Granola']]
18          result =
                lh.list_helper.remove_duplicates_list_of_lists(list_of_lists)
19
20          self.assertEqual(len(result), len(list_of_lists))
21          self.assertIn(('Abacate', 'Abacate', 'Granola'), result)
22          self.assertIn(('Abacate', 'Granola'), result)
23
24      def test_remove_duplicates_list_empty(self):
25          result =
                lh.list_helper.remove_duplicates_list_of_lists([])
26          self.assertEqual(len(result), 0)
27
28      def test_remove_duplicates_dict_ok(self):
29          list_of_dicts = [{'key0': 1}, {'key1': 2, 'key2': 3},
                {'key0': 1}]
30          result =
                lh.list_helper.remove_duplicates_list_of_dicts(list_of_dicts)
31
32          self.assertEqual(len(result), 2)
33          self.assertIn({'key0': 1}, result)
34          self.assertIn({'key1': 2, 'key2': 3}, result)
35
36      def test_remove_duplicates_dict_not_remove(self):
37          list_of_dicts = [{'key0': 1}, {'key1': 2, 'key2': 3}]
38          result =
                lh.list_helper.remove_duplicates_list_of_dicts(list_of_dicts)
39
40          self.assertEqual(len(result), len(list_of_dicts))
41          self.assertIn({'key0': 1}, result)
42          self.assertIn({'key1': 2, 'key2': 3}, result)
43
44      def test_remove_duplicates_dict_empty(self):
45          result =
                lh.list_helper.remove_duplicates_list_of_dicts([])
46          self.assertEqual(len(result), 0)
47
48
49  if __name__ == '__main__':
50      unittest.main()
```

repositorio/tests/helper/test_number_helper.py

```
1  import unittest
2  import math
3  from src.helper import number_helper as nh
4
5
6  class TestsNumberHelper(unittest.TestCase):
7
8      def test_without_dot(self):
9          is_number, number = nh.number_helper.is_number('100')
10
11         self.assertTrue(is_number)
12         self.assertEqual(number, 100)
13
14     def test_one_dot_and_one_comma_decimal(self):
15         is_number, number =
               nh.number_helper.is_number('1.234,50')
16
17         self.assertTrue(is_number)
18         self.assertEqual(number, 1234.50)
19
20     def test_more_than_one_comma(self):
21         is_number, number =
               nh.number_helper.is_number('1,234,50')
22
23         self.assertFalse(is_number)
24         self.assertTrue(math.isnan(number))
25
26     def test_one_dot_two_decimals(self):
27         is_number, number = nh.number_helper.is_number('1234.12')
28
29         self.assertTrue(is_number)
30         self.assertEqual(number, 1234.12)
31
32     def test_one_dot_one_decimal(self):
33         is_number, number = nh.number_helper.is_number('1234.0')
34
35         self.assertTrue(is_number)
36         self.assertEqual(number, 1234.0)
37
38     def test_word_nan(self):
39         is_number, number = nh.number_helper.is_number('abacate')
```

```
40
41          self.assertFalse(is_number)
42          self.assertTrue(math.isnan(number))
43
44      def test_more_one_dot(self):
45          is_number, number =
               nh.number_helper.is_number('1.234.567')
46
47          self.assertTrue(is_number)
48          self.assertEqual(number, 1234567.0)
49
50      def test_more_one_dot_and_decimal(self):
51          is_number, number =
               nh.number_helper.is_number('1.234.567.89')
52
53          self.assertTrue(is_number)
54          self.assertEqual(number, 1234567.89)
55
56      def test_more_one_dot_and_decimal_nan(self):
57          is_number, number =
               nh.number_helper.is_number('1.234.567.890000')
58
59          self.assertFalse(is_number)
60          self.assertTrue(math.isnan(number))
61
62      def test_dot_and_comma_one_decimal(self):
63          is_number, number = nh.number_helper.is_number('2.587,6')
64
65          self.assertTrue(is_number)
66          self.assertEqual(number, 2587.6)
67
68      def test_dot_and_comma_without_decimal(self):
69          is_number, number = nh.number_helper.is_number('2.587,')
70
71          self.assertFalse(is_number)
72          self.assertTrue(math.isnan(number))
73
74
75  if __name__ == '__main__':
76      unittest.main()
```

repositorio/tests/helper/test_result_helper.py

```
import unittest
from src.helper import result_helper as rh


class TestsResultHelper(unittest.TestCase):

    def test_create_result_item_with_undefined(self):
        result = rh.result_helper.create_result_item(25, '10')

        expected = {'number': 25, 'possible_size': 'undefined'}

        self.assertEqual(result, expected)

    def test_create_result_item_without_undefined(self):
        result = rh.result_helper.create_result_item(25,
           'milhões')

        expected = {'number': 25, 'possible_size': 'milhões'}

        self.assertEqual(result, expected)

    def test_clean_search_result_success(self):
        test_input = [{'number': 10, 'possible_size':
           'undefined'},
                      {'number': 30, 'possible_size': 40},
                      {'possible_size': 50}]
        result = rh.result_helper.clean_search_result(test_input)

        expected = [{'number': 30, 'possible_size': 40},
           {'possible_size': 50}]
        self.assertEqual(result, expected)

    def test_clean_search_result_without_undefined(self):
        test_input = [{'number': 10, 'possible_size': 20},
                      {'number': 30, 'possible_size': 40},
                      {'possible_size': 50}]
        result = rh.result_helper.clean_search_result(test_input)

        expected = [{'number': 10, 'possible_size': 20},
                    {'number': 30, 'possible_size': 40},
                    {'possible_size': 50}]
```

```
39          self.assertEqual(result, expected)
40
41      def test_clean_search_result_empty(self):
42          result = rh.result_helper.clean_search_result([])
43
44          expected = []
45          self.assertEqual(result, expected)
46
47      def test_get_numbers_as_list_witht_number_key(self):
48          test_input = [{'number': 10, 'other_key': 20},
               {'number': 30, 'other_key': 40}, {'other_key': 50}]
49          result = rh.result_helper.get_numbers_as_list(test_input)
50
51          expected = [10, 30]
52          self.assertEqual(result, expected)
53
54      def test_get_numbers_as_list_without_number_key(self):
55          test_input = [{'numb': 10, 'other_key': 20},
               {'other_key': 40}, {'other_key': 50}]
56          result = rh.result_helper.get_numbers_as_list(test_input)
57
58          expected = []
59          self.assertEqual(result, expected)
60
61      def test_get_numbers_as_list_empty(self):
62          result = rh.result_helper.get_numbers_as_list([])
63
64          expected = []
65          self.assertEqual(result, expected)
66
67
68  if __name__ == '__main__':
69      unittest.main()
```

repositorio/tests/plataform/test_data_dir_scan.py

```
1
2  import unittest
3  from unittest import mock
4  from src.plataform import data_dir_scan as dds
5
6
```

```
7  class TestsDataDirScan(unittest.TestCase):
8
9      @mock.patch('os.getcwd')
10     def test_get_data_directory_from_root(self, mock_getcwd):
11         mock_getcwd.return_value = '/home/tcc'
12         self.assertEqual(dds.data_dir_scan.get_data_directory(),
               'data/')
13         mock_getcwd.assert_called_once()
14
15     @mock.patch('os.getcwd')
16     def test_get_data_directory_from_stranger_place(self,
           mock_getcwd):
17         mock_getcwd.return_value = '/home'
18         self.assertRaises(Exception,
               dds.data_dir_scan.get_data_directory)
19         mock_getcwd.assert_called_once()
20
21     @mock.patch('os.getcwd')
22     def test_get_data_directory_from_tests(self, mock_getcwd):
23         mock_getcwd.return_value = '/home/tcc/tests'
24         self.assertEqual(dds.data_dir_scan.get_data_directory(),
               '../data/')
25         mock_getcwd.assert_called_once()
26
27     @mock.patch('os.getcwd')
28     def
           test_get_data_directory_from_src_with_levels_belows(self,
           mock_getcwd):
29         mock_getcwd.return_value = '/home/tcc/src/plataform'
30         self.assertEqual(dds.data_dir_scan.get_data_directory(),
               '../../data/')
31         mock_getcwd.assert_called_once()
32
33     @mock.patch('os.getcwd')
34     def test_get_data_directory_from_data(self, mock_getcwd):
35         mock_getcwd.return_value = '/home/tcc/data'
36         self.assertEqual(dds.data_dir_scan.get_data_directory(),
               './')
37         mock_getcwd.assert_called_once()
38
39
40 if __name__ == '__main__':
```

```
41      unittest.main()
```

repositorio/tests/plataform/test_filters.py

```
1  import unittest
2
3  from src.plataform import filters as f
4
5
6  class TestsFilters(unittest.TestCase):
7
8      filters = None
9
10     @classmethod
11     def setUpClass(cls):
12         cls.filters = f.filters()
13
14     @classmethod
15     def tearDownClass(cls):
16         cls.filters = None
17
18     def test_candidate_sentences_all_empty(self):
19         result = self.filters.candidate_sentences([['']],
               [frozenset([])])
20         self.assertEqual(len(result), 0)
21
22     def test_candidate_sentences_not_subset(self):
23         result = self.filters.candidate_sentences([['Ameixa'],
               ['Ameixa'], ['Abacate']], [frozenset(['teste',
               'unitário'])])
24         self.assertEqual(len(result), 0)
25
26     def test_candidate_sentences_subset(self):
27         list_of_expected = ['Ameixa', 'muito', 'saborosa']
28
29         result =
               self.filters.candidate_sentences([list_of_expected,
               ['Ameixa'], ['Abacate']], [frozenset(['saborosa'])])
30         expected = [list_of_expected]
31
32         self.assertEqual(len(result), 1)
33         self.assertEqual(expected, result)
```

```
34
35      def test_is_searcher_words_in_sequence_all_empty(self):
36          result =
               self.filters.is_searcher_words_in_sequence([[]],
               [['vazio']])
37          self.assertEqual(len(result), 0)
38
39      def test_is_searcher_words_in_sequence_empty_position(self):
40          candidate_sentences = [['eu', 'gost', 'abacat', 'tard',
               '.'],
41                                 ['-', 'ach', 'bom', 'gost',
                                      'abacax', 'barat', 'r', '$',
                                      '.'],
42                                 ['gost', 'tangerin', 'cust', 'r',
                                      '$', '.']]
43          result =
               self.filters.is_searcher_words_in_sequence(candidate_sentences,
               [['avela']])
44          self.assertEqual(len(result), 0)
45
46      def
           test_is_searcher_words_in_sequence_is_hit_by_except(self):
47          candidate_sentences = [['eu'],
48                                 ['-', 'ach', 'bom', 'gost',
                                      'abacax', 'barat', 'r', '$',
                                      '.'],
49                                 ['gost', 'tangerin', 'cust', 'r',
                                      '$', '.']]
50          result =
               self.filters.is_searcher_words_in_sequence(candidate_sentences,
               [['eu', 'abacat']])
51          self.assertEqual(len(result), 0)
52
53      def test_is_searcher_words_in_sequence_is_hit_false(self):
54          candidate_sentences = [['eu', 'gost', 'abacat', 'tard',
               '.'],
55                                 ['-', 'ach', 'bom', 'gost',
                                      'abacax', 'barat', 'r', '$',
                                      '.'],
56                                 ['gost', 'tangerin', 'cust', 'r',
                                      '$', '.']]
57          result =
```

```
            self.filters.is_searcher_words_in_sequence(candidate_sentences,
            [['barat', 'abacat']])
58          self.assertEqual(len(result), 0)
59
60      def test_is_searcher_words_in_sequence_success(self):
61          candidate_sentences = [['eu', 'gost', 'abacat', 'tard',
            '.'],
62                                 ['-', 'ach', 'bom', 'gost',
                                    'abacax', 'barat', 'r', '$',
                                    '.'],
63                                 ['gost', 'tangerin', 'cust', 'r',
                                    '$', '.']]
64          result =
            self.filters.is_searcher_words_in_sequence(candidate_sentences,
            [['gost', 'abacat']])
65
66          self.assertEqual(len(result), 1)
67          self.assertEqual(result, [('eu', 'gost', 'abacat',
            'tard', '.')])
68
69
70  if __name__ == '__main__':
71      unittest.main()
```

repositorio/tests/plataform/test_searcher.py

```
1   import unittest
2   from src.plataform import searcher as s
3
4
5   class TestsSearcher(unittest.TestCase):
6
7       searcher = None
8
9       @classmethod
10      def setUpClass(cls):
11          cls.searcher = s.searcher()
12
13      @classmethod
14      def tearDownClass(cls):
15          cls.searcher = None
16
```

```
17      def test_monetary_value_empty_candidate_sentences(self):
18          result = self.searcher.monetary_value([])
19          self.assertEqual(len(result), 0)
20
21      def test_monetary_value_empty_sentences(self):
22          result = self.searcher.monetary_value([[], [], []])
23          self.assertEqual(len(result), 0)
24
25      def test_monetary_value_without_numbers(self):
26          candidate_sentences = [['eu', 'gost', 'abacat', 'tard',
               '.'],
27                                 ['-', 'ach', 'bom', 'gost',
                                      'abacax', 'barat', 'r', '$',
                                      '.'],
28                                 ['gost', 'tangerin', 'cust', 'r',
                                      '$', '.']]
29          result =
               self.searcher.monetary_value(candidate_sentences)
30          self.assertEqual(len(result), 0)
31
32      def test_monetary_value_numbers_with_invalid_positions(self):
33          candidate_sentences = [['3', '-', 'ach', 'bom', 'gost',
               'abacax']]
34          result =
               self.searcher.monetary_value(candidate_sentences)
35          self.assertEqual(len(result), 0)
36
37      def test_monetary_value_number_not_monetary(self):
38          candidate_sentences = [['eu', 'gost', 'abacat', '6',
               'tard', '.'],
39                                 ['3', '-', 'ach', 'bom', 'gost',
                                      'abacax', 'barat', '2.50', '.'],
40                                 ['gost', 'tangerin', 'cust',
                                      '1.00', '.']]
41          result =
               self.searcher.monetary_value(candidate_sentences)
42          self.assertEqual(len(result), 0)
43
44      def test_monetary_value_success(self):
45          candidate_sentences = [['eu', 'gost', 'abacat', '6',
               'tard', '.'],
46                                 ['3', '-', 'ach', 'bom', 'gost',
```

```
                                    'abacax', 'barat', 'r', '$',
                                    '2.50', '.'],
47                                ['gost', 'tangerin', 'cust', 'r',
                                    '$', '1.00', '.']]
48          result =
                self.searcher.monetary_value(candidate_sentences)
49          self.assertEqual(len(result), 2)
50          self.assertEqual([{'number': 2.5, 'possible_size': '.'},
                {'number': 1.0, 'possible_size': '.'}], result)
51
52      def test_after_target_set_empty_inputs(self):
53          result =
                self.searcher.after_target_set_number_value([[]], [])
54          self.assertEqual(len(result), 0)
55
56      def test_after_target_set_success(self):
57          candidate_sentences = [['eu', 'gost', 'abacat', '6',
                'tard', '.'],
58                                 ['3', '-', 'ach', 'bom', 'gost',
                                    'abacax', 'barat', '2.50', '.'],
59                                 ['gost', 'tangerin', 'e',
                                    'abacat', '1.00', '.']]
60          result =
                self.searcher.after_target_set_number_value(candidate_sentences,
                [['abacat']])
61          self.assertEqual(len(result), 2)
62          self.assertEqual([{'number': 6, 'possible_size':
                'tard'}, {'number': 1.0, 'possible_size': '.'}],
                result)
63
64      def test_after_target_set_invalid_position(self):
65          candidate_sentences = [['6']]
66          result =
                self.searcher.after_target_set_number_value(candidate_sentences,
                [['abacat']])
67          self.assertEqual(len(result), 0)
68
69      def test_after_target_set_without_number(self):
70          candidate_sentences = [['eu', 'gost', 'abacat', 'tard',
                '.'],
71                                 ['-', 'ach', 'bom', 'gost',
                                    'abacax', 'barat', '.'],
```

```
72                                  ['gost', 'tangerin', 'e',
                                       'abacat', '.']]
73          result =
               self.searcher.after_target_set_number_value(candidate_sentences
               [['abacat']])
74          self.assertEqual(len(result), 0)
75
76      def test_after_target_set_not_in_target_set(self):
77          candidate_sentences = [['eu', 'gost', 'abacat', '6',
               'tard', '.'],
78                                 ['3', '-', 'ach', 'bom', 'gost',
                                       'abacax', 'barat', '2.50', '.'],
79                                 ['gost', 'tangerin', 'e',
                                       'abacat', '1.00', '.']]
80          result =
               self.searcher.after_target_set_number_value(candidate_sentences
               [['lofi']])
81          self.assertEqual(len(result), 0)
82
83
84  if __name__ == '__main__':
85      unittest.main()
```

repositorio/tests/reports/pypdf2/pypdf2_test_ambev20191T.py

```
1   import unittest
2   from src import manager as m
3   from src.helper import result_helper as rh
4   from data import data as data
5
6
7   class TestsAmbev20191T(unittest.TestCase):
8
9       filename = None
10      manager_pypdf2 = None
11
12      @classmethod
13      def setUpClass(cls):
14          cls.filename = 'ambev_2019_1T.pdf'
15          cls.manager_pypdf2 = m.manager([cls.filename], 'pypdf2')
16
17      @classmethod
```

```
18      def tearDownClass(cls):
19          cls.filename = None
20          cls.manager_pypdf2 = None
21
22      def test_lucro_liquido_monetary(self):
23          lucro_liquido_monetary_pypdf2 =
               self.manager_pypdf2.run_lucro_liquido_monetary()
24          result_pypdf2 =
               lucro_liquido_monetary_pypdf2[self.filename]
25          numbers_from_result_pypdf2 =
               rh.result_helper.get_numbers_as_list(result_pypdf2)
26
27          self.assertEqual(len(result_pypdf2), 2, 'lucro líquido
               (R$): tamanho resultado (pypdf2)')
28          self.assertNotIn(data.LUCRO_LIQUIDO[self.filename],
               numbers_from_result_pypdf2,
29                           'lucro líquido (R$): valor (pypdf2)')
30
31      def test_lucro_liquido_number(self):
32          lucro_liquido_number_pypdf2 =
               self.manager_pypdf2.run_lucro_liquido_number()
33          result_pypdf2 =
               lucro_liquido_number_pypdf2[self.filename]
34
35          self.assertEqual(len(result_pypdf2), 0, 'lucro líquido
               (número após conjunto de busca): tamanho resultado')
36
37      def test_patrimonio_liquido_monetary(self):
38          patrimonio_liquido_monetary =
               self.manager_pypdf2.run_patrimonio_liquido_monetary()
39          result = patrimonio_liquido_monetary[self.filename]
40
41          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
42
43      def test_patrimonio_liquido_number(self):
44          patrimonio_liquido_number_pypdf2 =
               self.manager_pypdf2.run_patrimonio_liquido_number()
45          result_pypdf2 =
               patrimonio_liquido_number_pypdf2[self.filename]
46
47          self.assertEqual(len(result_pypdf2), 0,
```

```
48                          'patrimônio líquido (número após
                               conjunto de busca): tamanho resultado
                               (pypdf2)')
49
50      def test_roe_monetary(self):
51          roe_monetary = self.manager_pypdf2.run_roe_monetary()
52          result = roe_monetary[self.filename]
53
54          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
                resultado')
55
56      def test_roe_number(self):
57          roe_number = self.manager_pypdf2.run_roe_number()
58          result = roe_number[self.filename]
59
60          self.assertEqual(len(result), 0, 'ROE (número após
                conjunto de busca): tamanho resultado')
61
62      def test_roe_calculate(self):
63          calculate_roe_pypdf2 =
                self.manager_pypdf2.run_calculate_roe()
64          result_pypdf2 = calculate_roe_pypdf2[self.filename]
65
66          self.assertEqual(len(result_pypdf2), 0, 'ROE por
                cálculo: tamanho resultado (pypdf2)')
67
68
69  if __name__ == '__main__':
70      unittest.main()
```

repositorio/tests/reports/pypdf2/pypdf2_test_engie20192T.py

```
1   import unittest
2   from src import manager as m
3   from src.helper import result_helper as rh
4   from data import data as data
5
6
7   class TestsEngie20192T(unittest.TestCase):
8
9       filename = None
10      manager_pypdf2 = None
```

```
11
12      @classmethod
13      def setUpClass(cls):
14          cls.filename = 'engie_2019_2T.pdf'
15          cls.manager_pypdf2 = m.manager([cls.filename], 'pypdf2')
16
17      @classmethod
18      def tearDownClass(cls):
19          cls.filename = None
20          cls.manager_pypdf2 = None
21
22      def test_lucro_liquido_monetary(self):
23          lucro_liquido_monetary_pypdf2 =
               self.manager_pypdf2.run_lucro_liquido_monetary()
24          result_pypdf2 =
               lucro_liquido_monetary_pypdf2[self.filename]
25          numbers_from_result_pypdf2 =
               rh.result_helper.get_numbers_as_list(result_pypdf2)
26
27          self.assertEqual(len(result_pypdf2), 3, 'lucro líquido
               (R$): tamanho resultado (pypdf2)')
28          self.assertIn(data.LUCRO_LIQUIDO[self.filename],
               numbers_from_result_pypdf2, 'lucro líquido (R$):
               valor')
29
30      def test_lucro_liquido_number(self):
31          lucro_liquido_number_pypdf2 =
               self.manager_pypdf2.run_lucro_liquido_number()
32          result_pypdf2 =
               lucro_liquido_number_pypdf2[self.filename]
33
34          self.assertEqual(len(result_pypdf2), 0, 'lucro líquido
               (R$): tamanho resultado (pypdf2)')
35
36      def test_patrimonio_liquido_monetary(self):
37          patrimonio_liquido_monetary =
               self.manager_pypdf2.run_patrimonio_liquido_monetary()
38          result = patrimonio_liquido_monetary[self.filename]
39
40          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
41
```

```
42      def test_patrimonio_liquido_number(self):
43          patrimonio_liquido_number =
               self.manager_pypdf2.run_patrimonio_liquido_number()
44          result = patrimonio_liquido_number[self.filename]
45
46          self.assertEqual(len(result), 0, 'patrimônio líquido
               (número após conjunto de busca): tamanho resultado')
47
48      def test_roe_monetary(self):
49          roe_monetary = self.manager_pypdf2.run_roe_monetary()
50          result = roe_monetary[self.filename]
51
52          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
53
54      def test_roe_number(self):
55          roe_number = self.manager_pypdf2.run_roe_number()
56          result = roe_number[self.filename]
57
58          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
59
60      def test_roe_calculate(self):
61          roe_calculate = self.manager_pypdf2.run_calculate_roe()
62          result = roe_calculate[self.filename]
63
64          self.assertEqual(len(result), 0, 'ROE por cálculo:
               tamanho resultado')
65
66
67  if __name__ == '__main__':
68      unittest.main()
```

repositorio/tests/reports/pypdf2/pypdf2_test_engie20202T.py

```
1  import unittest
2  from src import manager as m
3  from src.helper import result_helper as rh
4  from data import data as data
5
6
7  class TestsEngie20202T(unittest.TestCase):
```

```
8
9       filename = None
10      manager_pypdf2 = None
11
12      @classmethod
13      def setUpClass(cls):
14          cls.filename = 'engie_2020_2T.pdf'
15          cls.manager_pypdf2 = m.manager([cls.filename], 'pypdf2')
16
17      @classmethod
18      def tearDownClass(cls):
19          cls.filename = None
20          cls.manager_pypdf2 = None
21
22      def test_lucro_liquido_monetary(self):
23          lucro_liquido_monetary_pypdf2 =
               self.manager_pypdf2.run_lucro_liquido_monetary()
24          result_pypdf2 =
               lucro_liquido_monetary_pypdf2[self.filename]
25          numbers_from_result_pypdf2 =
               rh.result_helper.get_numbers_as_list(result_pypdf2)
26
27          self.assertEqual(len(result_pypdf2), 6, 'lucro líquido
               (R$): tamanho resultado (pypdf2)')
28          self.assertIn(data.LUCRO_LIQUIDO[self.filename],
               numbers_from_result_pypdf2, 'lucro líquido (R$): valor
               (pypdf2)')
29
30      def test_lucro_liquido_number(self):
31          lucro_liquido_number_pypdf2 =
               self.manager_pypdf2.run_lucro_liquido_number()
32          result_pypdf2 =
               lucro_liquido_number_pypdf2[self.filename]
33
34          self.assertEqual(len(result_pypdf2), 0, 'lucro líquido
               (número após conjunto de busca): tamanho resultado')
35
36      def test_patrimonio_liquido_monetary(self):
37          patrimonio_liquido_monetary =
               self.manager_pypdf2.run_patrimonio_liquido_monetary()
38          result = patrimonio_liquido_monetary[self.filename]
39
```

```
40          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
41
42      def test_patrimonio_liquido_number(self):
43          patrimonio_liquido_number_pypdf2 =
               self.manager_pypdf2.run_patrimonio_liquido_number()
44          result_pypdf2 =
               patrimonio_liquido_number_pypdf2[self.filename]
45
46          self.assertEqual(len(result_pypdf2), 0,
47                           'patrimônio líquido (número após
                              conjunto de busca): tamanho resultado
                              (pypdf2)')
48
49      def test_roe_monetary(self):
50          roe_monetary = self.manager_pypdf2.run_roe_monetary()
51          result = roe_monetary[self.filename]
52
53          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
54
55      def test_roe_number(self):
56          roe_number = self.manager_pypdf2.run_roe_number()
57          result = roe_number[self.filename]
58
59          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
60
61      def test_roe_calculate(self):
62          roe_calculate_pypdf2 =
               self.manager_pypdf2.run_calculate_roe()
63          result_pypdf2 = roe_calculate_pypdf2[self.filename]
64
65          self.assertEqual(len(result_pypdf2), 0, 'ROE por
               cálculo: tamanho resultado (pypdf2)')
66
67
68  if __name__ == '__main__':
69      unittest.main()
```

repositorio/tests/reports/pypdf2/pypdf2_test_fleury20193T.py

```
1  import unittest
2  from src import manager as m
3
4
5  class TestsFleury20193T(unittest.TestCase):
6
7      filename = None
8      manager_pypdf2 = None
9
10     @classmethod
11     def setUpClass(cls):
12         cls.filename = 'fleury_2019_3T.pdf'
13         cls.manager_pypdf2 = m.manager([cls.filename], 'pypdf2')
14
15     @classmethod
16     def tearDownClass(cls):
17         cls.filename = None
18         cls.manager_pypdf2 = None
19
20     def test_lucro_liquido_monetary(self):
21         lucro_liquido_monetary =
               self.manager_pypdf2.run_lucro_liquido_monetary()
22         result = lucro_liquido_monetary[self.filename]
23
24         self.assertEqual(len(result), 0, 'lucro líquido (R$):
               tamanho resultado')
25
26     def test_lucro_liquido_number(self):
27         lucro_liquido_number_pypdf2 =
               self.manager_pypdf2.run_lucro_liquido_number()
28         result_pypdf2 =
               lucro_liquido_number_pypdf2[self.filename]
29
30         self.assertEqual(len(result_pypdf2), 0, 'lucro líquido
               (número após conjunto de busca): tamanho resultado')
31
32     def test_patrimonio_liquido_monetary(self):
33         patrimonio_liquido_monetary =
               self.manager_pypdf2.run_patrimonio_liquido_monetary()
34         result = patrimonio_liquido_monetary[self.filename]
35
36         self.assertEqual(len(result), 0, 'patrimônio líquido
```

```
            (R$): tamanho resultado')
37
38      def test_patrimonio_liquido_number(self):
39          patrimonio_liquido_number =
               self.manager_pypdf2.run_patrimonio_liquido_number()
40          result = patrimonio_liquido_number[self.filename]
41
42          self.assertEqual(len(result), 0, 'patrimônio líquido
               (número após conjunto de busca): tamanho resultado')
43
44      def test_roe_monetary(self):
45          roe_monetary = self.manager_pypdf2.run_roe_monetary()
46          result = roe_monetary[self.filename]
47
48          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
49
50      def test_roe_number(self):
51          roe_number = self.manager_pypdf2.run_roe_number()
52          result = roe_number[self.filename]
53
54          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
55
56      def test_roe_calculate(self):
57          roe_calculate = self.manager_pypdf2.run_calculate_roe()
58          result = roe_calculate[self.filename]
59
60          self.assertEqual(len(result), 0, 'ROE por cálculo:
               tamanho resultado')
61
62
63  if __name__ == '__main__':
64      unittest.main()
```

repositorio/tests/reports/pypdf2/pypdf2_test_fleury20202T.py

```
1  import unittest
2  from src import manager as m
3  from src.helper import result_helper as rh
4  from data import data as data
5
```

```
6
7  class TestsFleury20202T(unittest.TestCase):
8
9      filename = None
10     manager_pypdf2 = None
11
12     @classmethod
13     def setUpClass(cls):
14         cls.filename = 'fleury_2020_2T.pdf'
15         cls.manager_pypdf2 = m.manager([cls.filename], 'pypdf2')
16
17     @classmethod
18     def tearDownClass(cls):
19         cls.filename = None
20         cls.manager_pypdf2 = None
21
22     def test_lucro_liquido_monetary(self):
23         lucro_liquido_monetary_pypdf2 =
               self.manager_pypdf2.run_lucro_liquido_monetary()
24         result_pypdf2 =
               lucro_liquido_monetary_pypdf2[self.filename]
25         numbers_from_result_pypdf2 =
               rh.result_helper.get_numbers_as_list(result_pypdf2)
26
27         self.assertEqual(len(result_pypdf2), 1, 'lucro líquido
               (R$): tamanho resultado (pypdf2)')
28         self.assertIn(data.LUCRO_LIQUIDO[self.filename],
               numbers_from_result_pypdf2, 'lucro líquido (R$):
               valor')
29
30     def test_lucro_liquido_number(self):
31         lucro_liquido_number =
               self.manager_pypdf2.run_lucro_liquido_number()
32         result = lucro_liquido_number[self.filename]
33
34         self.assertEqual(len(result), 0, 'lucro líquido (número
               após conjunto de busca): tamanho resultado')
35
36     def test_patrimonio_liquido_monetary(self):
37         patrimonio_liquido_monetary =
               self.manager_pypdf2.run_patrimonio_liquido_monetary()
38         result = patrimonio_liquido_monetary[self.filename]
```

```
39
40          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
41
42      def test_patrimonio_liquido_number(self):
43          patrimonio_liquido_number_pypdf2 =
               self.manager_pypdf2.run_patrimonio_liquido_number()
44          result_pypdf2 =
               patrimonio_liquido_number_pypdf2[self.filename]
45          numbers_from_result_pypdf2 =
               rh.result_helper.get_numbers_as_list(result_pypdf2)
46
47          self.assertEqual(len(result_pypdf2), 2,
48                           'patrimônio líquido (número após
                              conjunto de busca): tamanho resultado
                              (pypdf2)')
49          self.assertIn(data.PATRIMONIO_LIQUIDO[self.filename],
               numbers_from_result_pypdf2,
50                        'patrimônio líquido (número após conjunto
                             de busca): valor (pypdf2)')
51
52      def test_roe_monetary(self):
53          roe_monetary = self.manager_pypdf2.run_roe_monetary()
54          result = roe_monetary[self.filename]
55
56          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
57
58      def test_roe_number(self):
59          roe_number = self.manager_pypdf2.run_roe_number()
60          result = roe_number[self.filename]
61
62          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
63
64      def test_roe_calculate(self):
65          roe_calculate_pypdf2 =
               self.manager_pypdf2.run_calculate_roe()
66          result_pypdf2 = roe_calculate_pypdf2[self.filename]
67
68          self.assertEqual(len(result_pypdf2), 2, 'ROE por
               cálculo: tamanho resultado (pypdf2)')
```

```
69          self.assertIn(data.ROE[self.filename], result_pypdf2,
               'ROE por cálculo: valor (pypdf2)')
70
71
72  if __name__ == '__main__':
73      unittest.main()
```

repositorio/tests/reports/pypdf2/pypdf2_test_gerdau20153T.py

```
1   import unittest
2   from src import manager as m
3   from src.helper import result_helper as rh
4   from data import data as data
5
6
7   class TestsGerdau20153T(unittest.TestCase):
8
9       filename = None
10      manager_pypdf2 = None
11
12      @classmethod
13      def setUpClass(cls):
14          cls.filename = 'gerdau_2015_3T.pdf'
15          cls.manager_pypdf2 = m.manager([cls.filename], 'pypdf2')
16
17      @classmethod
18      def tearDownClass(cls):
19          cls.filename = None
20          cls.manager_pypdf2 = None
21
22      def test_lucro_liquido_monetary(self):
23          lucro_liquido_monetary =
               self.manager_pypdf2.run_lucro_liquido_monetary()
24          result = lucro_liquido_monetary[self.filename]
25
26          self.assertEqual(len(result), 0, 'lucro líquido (R$):
               tamanho resultado')
27
28      def test_lucro_liquido_number(self):
29          lucro_liquido_number =
               self.manager_pypdf2.run_lucro_liquido_number()
30          result = lucro_liquido_number[self.filename]
```

```
31
32          self.assertEqual(len(result), 0, 'lucro líquido (número
                após conjunto de busca): tamanho resultado')
33
34      def test_patrimonio_liquido_monetary(self):
35          patrimonio_liquido_monetary =
                self.manager_pypdf2.run_patrimonio_liquido_monetary()
36          result = patrimonio_liquido_monetary[self.filename]
37
38          self.assertEqual(len(result), 0, 'patrimônio líquido
                (R$): tamanho resultado')
39
40      def test_patrimonio_liquido_number(self):
41          patrimonio_liquido_number_pypdf2 =
                self.manager_pypdf2.run_patrimonio_liquido_number()
42          result_pypdf2 =
                patrimonio_liquido_number_pypdf2[self.filename]
43          numbers_from_result =
                rh.result_helper.get_numbers_as_list(result_pypdf2)
44
45          self.assertEqual(len(result_pypdf2), 2, 'patrimônio
                líquido (número após conjunto de busca): tamanho
                resultado')
46          self.assertNotIn(data.PATRIMONIO_LIQUIDO[self.filename],
                numbers_from_result,
47                           'patrimônio líquido (número após
                                conjunto de busca): valor')
48
49      def test_roe_monetary(self):
50          roe_monetary = self.manager_pypdf2.run_roe_monetary()
51          result = roe_monetary[self.filename]
52
53          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
                resultado')
54
55      def test_roe_number(self):
56          roe_number = self.manager_pypdf2.run_roe_number()
57          result = roe_number[self.filename]
58
59          self.assertEqual(len(result), 0, 'ROE (número após
                conjunto de busca): tamanho resultado')
60
```

```
61      def test_roe_calculate(self):
62          roe_calculate = self.manager_pypdf2.run_calculate_roe()
63          result = roe_calculate[self.filename]
64
65          self.assertEqual(len(result), 0, 'ROE por cálculo:
                tamanho resultado')
66
67
68  if __name__ == '__main__':
69      unittest.main()
```

repositorio/tests/reports/pypdf2/pypdf2_test_gerdau20171T.py

```
1   import unittest
2   from src import manager as m
3   from src.helper import result_helper as rh
4   from data import data as data
5
6
7   class TestsGerdau20171T(unittest.TestCase):
8
9       filename = None
10      manager_pypdf2 = None
11
12      @classmethod
13      def setUpClass(cls):
14          cls.filename = 'gerdau_2017_1T.pdf'
15          cls.manager_pypdf2 = m.manager([cls.filename], 'pypdf2')
16
17      @classmethod
18      def tearDownClass(cls):
19          cls.filename = None
20          cls.manager_pypdf2 = None
21
22      def test_lucro_liquido_monetary(self):
23          lucro_liquido_monetary =
                self.manager_pypdf2.run_lucro_liquido_monetary()
24          result = lucro_liquido_monetary[self.filename]
25
26          self.assertEqual(len(result), 0, 'lucro líquido (R$):
                tamanho resultado')
27
```

```
28      def test_lucro_liquido_number(self):
29          lucro_liquido_number_pypdf2 =
               self.manager_pypdf2.run_lucro_liquido_number()
30          result_pypdf2 =
               lucro_liquido_number_pypdf2[self.filename]
31
32          self.assertEqual(len(result_pypdf2), 1, 'lucro líquido
               (número após conjunto de busca): tamanho resultado')
33          self.assertEqual(result_pypdf2[0]['number'],
               data.LUCRO_LIQUIDO[self.filename],
34                           'lucro líquido (número após conjunto de
                                busca): valor (pypdf2)')
35
36      def test_patrimonio_liquido_monetary(self):
37          patrimonio_liquido_monetary =
               self.manager_pypdf2.run_patrimonio_liquido_monetary()
38          result = patrimonio_liquido_monetary[self.filename]
39
40          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
41
42      def test_patrimonio_liquido_number(self):
43          patrimonio_liquido_number_pypdf2 =
               self.manager_pypdf2.run_patrimonio_liquido_number()
44          result_pypdf2 =
               patrimonio_liquido_number_pypdf2[self.filename]
45          numbers_from_result =
               rh.result_helper.get_numbers_as_list(result_pypdf2)
46
47          self.assertEqual(len(result_pypdf2), 2, 'patrimônio
               líquido (número após conjunto de busca): tamanho
               resultado')
48          self.assertNotIn(data.PATRIMONIO_LIQUIDO[self.filename],
               numbers_from_result,
49                           'patrimônio líquido (número após
                                conjunto de busca): valor')
50
51      def test_roe_monetary(self):
52          roe_monetary = self.manager_pypdf2.run_roe_monetary()
53          result = roe_monetary[self.filename]
54
55          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
```

```
            resultado')
56
57      def test_roe_number(self):
58          roe_number = self.manager_pypdf2.run_roe_number()
59          result = roe_number[self.filename]
60
61          self.assertEqual(len(result), 0, 'ROE (número após
                conjunto de busca): tamanho resultado')
62
63      def test_roe_calculate(self):
64          calculate_roe_pypdf2 =
                self.manager_pypdf2.run_calculate_roe()
65          result_pypdf2 = calculate_roe_pypdf2[self.filename]
66
67          self.assertEqual(len(result_pypdf2), 2, 'ROE por
                cálculo: tamanho resultado (pypdf2)')
68          self.assertNotIn(data.ROE[self.filename], result_pypdf2,
                'ROE por cálculo: valor')
69
70
71  if __name__ == '__main__':
72      unittest.main()
```

repositorio/tests/reports/pypdf2/pypdf2_test_gerdau20184T.py

```
1   import unittest
2   from src import manager as m
3   from src.helper import result_helper as rh
4   from data import data as data
5
6
7   class TestsGerdau20184T(unittest.TestCase):
8
9       filename = None
10      manager_pypdf2 = None
11
12      @classmethod
13      def setUpClass(cls):
14          cls.filename = 'gerdau_2018_4T.pdf'
15          cls.manager_pypdf2 = m.manager([cls.filename], 'pypdf2')
16
17      @classmethod
```

```
18      def tearDownClass(cls):
19          cls.filename = None
20          cls.manager_pypdf2 = None
21
22      def test_lucro_liquido_monetary(self):
23          lucro_liquido_monetary =
               self.manager_pypdf2.run_lucro_liquido_monetary()
24          result = lucro_liquido_monetary[self.filename]
25
26          self.assertEqual(len(result), 0, 'lucro líquido (R$):
               tamanho resultado')
27
28      def test_lucro_liquido_number(self):
29          lucro_liquido_number_pypdf2 =
               self.manager_pypdf2.run_lucro_liquido_number()
30          result_pypdf2 =
               lucro_liquido_number_pypdf2[self.filename]
31          numbers_from_result_pypdf2 =
               rh.result_helper.get_numbers_as_list(result_pypdf2)
32
33          self.assertEqual(len(result_pypdf2), 1, 'lucro líquido
               (número após conjunto de busca): tamanho resultado')
34          self.assertNotIn(data.LUCRO_LIQUIDO[self.filename],
               numbers_from_result_pypdf2,
35                           'lucro líquido (número após conjunto de
                              busca): valor (pypdf2)')
36
37      def test_patrimonio_liquido_monetary(self):
38          patrimonio_liquido_monetary =
               self.manager_pypdf2.run_patrimonio_liquido_monetary()
39          result = patrimonio_liquido_monetary[self.filename]
40
41          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
42
43      def test_patrimonio_liquido_number(self):
44          patrimonio_liquido_number_pypdf2 =
               self.manager_pypdf2.run_patrimonio_liquido_number()
45          result_pypdf2 =
               patrimonio_liquido_number_pypdf2[self.filename]
46
47          self.assertEqual(len(result_pypdf2), 0,
```

```
48                          'patrimônio líquido (número após
                               conjunto de busca): tamanho resultado
                               (pypdf2)')
49
50      def test_roe_monetary(self):
51          roe_monetary = self.manager_pypdf2.run_roe_monetary()
52          result = roe_monetary[self.filename]
53
54          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
                resultado')
55
56      def test_roe_number(self):
57          roe_number = self.manager_pypdf2.run_roe_number()
58          result = roe_number[self.filename]
59
60          self.assertEqual(len(result), 0, 'ROE (número após
                conjunto de busca): tamanho resultado')
61
62      def test_roe_calculate(self):
63          calculate_roe_pypdf2 =
                self.manager_pypdf2.run_calculate_roe()
64          result_pypdf2 = calculate_roe_pypdf2[self.filename]
65
66          self.assertEqual(len(result_pypdf2), 0, 'ROE por
                cálculo: tamanho resultado (pypdf2)')
67
68
69  if __name__ == '__main__':
70      unittest.main()
```

repositorio/tests/reports/pypdf2/pypdf2_test_gerdau20192T.py

```
1   import unittest
2   from src import manager as m
3   from src.helper import result_helper as rh
4   from data import data as data
5
6
7   class TestsGerdau20192T(unittest.TestCase):
8
9       filename = None
10      manager_pypdf2 = None
```

```
11
12      @classmethod
13      def setUpClass(cls):
14          cls.filename = 'gerdau_2019_2T.pdf'
15          cls.manager_pypdf2 = m.manager([cls.filename], 'pypdf2')
16
17      @classmethod
18      def tearDownClass(cls):
19          cls.filename = None
20          cls.manager_pypdf2 = None
21
22      def test_lucro_liquido_monetary(self):
23          lucro_liquido_monetary =
               self.manager_pypdf2.run_lucro_liquido_monetary()
24          result = lucro_liquido_monetary[self.filename]
25
26          self.assertEqual(len(result), 0, 'lucro líquido (R$):
               tamanho resultado')
27
28      def test_lucro_liquido_number(self):
29          lucro_liquido_number_pypdf2 =
               self.manager_pypdf2.run_lucro_liquido_number()
30          result_pypdf2 =
               lucro_liquido_number_pypdf2[self.filename]
31          numbers_from_result_pypdf2 =
               rh.result_helper.get_numbers_as_list(result_pypdf2)
32
33          self.assertEqual(len(result_pypdf2), 1, 'lucro líquido
               (número após conjunto de busca): tamanho resultado')
34          self.assertNotIn(data.LUCRO_LIQUIDO[self.filename],
               numbers_from_result_pypdf2,
35                           'lucro líquido (número após conjunto de
                              busca): valor (pypdf2)')
36
37      def test_patrimonio_liquido_monetary(self):
38          patrimonio_liquido_monetary =
               self.manager_pypdf2.run_patrimonio_liquido_monetary()
39          result = patrimonio_liquido_monetary[self.filename]
40
41          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
42
```

```
43      def test_patrimonio_liquido_number(self):
44          patrimonio_liquido_number_pypdf2 =
               self.manager_pypdf2.run_patrimonio_liquido_number()
45          result_pypdf2 =
               patrimonio_liquido_number_pypdf2[self.filename]
46
47          self.assertEqual(len(result_pypdf2), 0,
48                           'patrimônio líquido (número após
                                conjunto de busca): tamanho resultado
                                (pypdf2)')
49
50      def test_roe_monetary(self):
51          roe_monetary = self.manager_pypdf2.run_roe_monetary()
52          result = roe_monetary[self.filename]
53
54          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
55
56      def test_roe_number(self):
57          roe_number = self.manager_pypdf2.run_roe_number()
58          result = roe_number[self.filename]
59
60          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
61
62      def test_roe_calculate(self):
63          calculate_roe_pypdf2 =
               self.manager_pypdf2.run_calculate_roe()
64          result_pypdf2 = calculate_roe_pypdf2[self.filename]
65
66          self.assertEqual(len(result_pypdf2), 0, 'ROE por
               cálculo: tamanho resultado (pypdf2)')
67
68
69  if __name__ == '__main__':
70      unittest.main()
```

repositorio/tests/reports/pypdf2/pypdf2_test_weg20102T.py

```
1  import unittest
2  from src import manager as m
3  from src.helper import result_helper as rh
```

```
4  from data import data as data
5
6
7  class TestsWeg20102T(unittest.TestCase):
8
9      filename = None
10     manager_pypdf2 = None
11
12     @classmethod
13     def setUpClass(cls):
14         cls.filename = 'weg_2010_2T.pdf'
15         cls.manager_pypdf2 = m.manager([cls.filename], 'pypdf2')
16
17     @classmethod
18     def tearDownClass(cls):
19         cls.filename = None
20         cls.manager_pypdf2 = None
21
22     def test_lucro_liquido_monetary(self):
23         lucro_liquido_monetary_pypdf2 =
               self.manager_pypdf2.run_lucro_liquido_monetary()
24         result_pypdf2 =
               lucro_liquido_monetary_pypdf2[self.filename]
25         numbers_from_result_pypdf2 =
               rh.result_helper.get_numbers_as_list(result_pypdf2)
26
27         self.assertEqual(len(result_pypdf2), 1, 'lucro líquido
               (R$): tamanho resultado (pypdf2)')
28         self.assertIn(data.LUCRO_LIQUIDO[self.filename],
               numbers_from_result_pypdf2, 'lucro líquido (R$):
               valor')
29
30     def test_lucro_liquido_number(self):
31         lucro_liquido_number =
               self.manager_pypdf2.run_lucro_liquido_number()
32         result = lucro_liquido_number[self.filename]
33
34         self.assertEqual(len(result), 0, 'lucro líquido (número
               após conjunto de busca): tamanho resultado')
35
36     def test_patrimonio_liquido_monetary(self):
37         patrimonio_liquido_monetary =
```

```
            self.manager_pypdf2.run_patrimonio_liquido_monetary()
38          result = patrimonio_liquido_monetary[self.filename]
39
40          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
41
42      def test_patrimonio_liquido_number(self):
43          patrimonio_liquido_number_pypdf2 =
               self.manager_pypdf2.run_patrimonio_liquido_number()
44          result_pypdf2 =
               patrimonio_liquido_number_pypdf2[self.filename]
45          numbers_from_result_pypdf2 =
               rh.result_helper.get_numbers_as_list(result_pypdf2)
46
47          self.assertEqual(len(result_pypdf2), 1,
48                           'patrimônio líquido (número após
                              conjunto de busca): tamanho resultado
                              (pypdf2)')
49          self.assertIn(data.PATRIMONIO_LIQUIDO[self.filename],
               numbers_from_result_pypdf2,
50                        'patrimônio líquido (número após conjunto
                             de busca): valor (pypdf2)')
51
52      def test_roe_monetary(self):
53          roe_monetary = self.manager_pypdf2.run_roe_monetary()
54          result = roe_monetary[self.filename]
55
56          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
57
58      def test_roe_number(self):
59          roe_number = self.manager_pypdf2.run_roe_number()
60          result = roe_number[self.filename]
61
62          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
63
64      def test_roe_calculate(self):
65          roe_calculate_pypdf2 =
               self.manager_pypdf2.run_calculate_roe()
66          result_pypdf2 = roe_calculate_pypdf2[self.filename]
67
```

```
68          self.assertEqual(len(result_pypdf2), 1, 'ROE por
               cálculo: tamanho resultado (pypdf2)')
69          self.assertIn(data.ROE[self.filename], result_pypdf2,
               'ROE por cálculo: valor (pypdf2)')
70
71
72  if __name__ == '__main__':
73      unittest.main()
```

repositorio/tests/reports/pypdf2/pypdf2_test_weg20151T.py

```
1   import unittest
2   from src import manager as m
3   from src.helper import result_helper as rh
4   from data import data as data
5
6
7   class TestsWeg20151T(unittest.TestCase):
8
9       filename = None
10      manager_pypdf2 = None
11
12      @classmethod
13      def setUpClass(cls):
14          cls.filename = 'weg_2015_1T.pdf'
15          cls.manager_pypdf2 = m.manager([cls.filename], 'pypdf2')
16
17      @classmethod
18      def tearDownClass(cls):
19          cls.filename = None
20          cls.manager_pypdf2 = None
21
22      def test_lucro_liquido_monetary(self):
23          lucro_liquido_monetary_pypdf2 =
               self.manager_pypdf2.run_lucro_liquido_monetary()
24          result_pypdf2 =
               lucro_liquido_monetary_pypdf2[self.filename]
25          numbers_from_result_pypdf2 =
               rh.result_helper.get_numbers_as_list(result_pypdf2)
26
27          self.assertEqual(len(result_pypdf2), 1, 'lucro líquido
               (R$): tamanho resultado (pypdf2)')
```

```
28          self.assertIn(data.LUCRO_LIQUIDO[self.filename],
                numbers_from_result_pypdf2, 'lucro líquido (R$):
                valor')
29
30      def test_lucro_liquido_number(self):
31          lucro_liquido_number =
                self.manager_pypdf2.run_lucro_liquido_number()
32          result = lucro_liquido_number[self.filename]
33
34          self.assertEqual(len(result), 0, 'lucro líquido (número
                após conjunto de busca): tamanho resultado')
35
36      def test_patrimonio_liquido_monetary(self):
37          patrimonio_liquido_monetary =
                self.manager_pypdf2.run_patrimonio_liquido_monetary()
38          result = patrimonio_liquido_monetary[self.filename]
39
40          self.assertEqual(len(result), 0, 'patrimônio líquido
                (R$): tamanho resultado')
41
42      def test_patrimonio_liquido_number(self):
43          patrimonio_liquido_number_pypdf2 =
                self.manager_pypdf2.run_patrimonio_liquido_number()
44          result_pypdf2 =
                patrimonio_liquido_number_pypdf2[self.filename]
45
46          self.assertEqual(len(result_pypdf2), 0,
47                           'patrimônio líquido (número após
                                conjunto de busca): tamanho resultado
                                (pypdf2)')
48
49      def test_roe_monetary(self):
50          roe_monetary = self.manager_pypdf2.run_roe_monetary()
51          result = roe_monetary[self.filename]
52
53          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
                resultado')
54
55      def test_roe_number(self):
56          roe_number = self.manager_pypdf2.run_roe_number()
57          result = roe_number[self.filename]
58
```

```
59          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
60
61      def test_roe_calculate(self):
62          calculate_roe_pypdf2 =
               self.manager_pypdf2.run_calculate_roe()
63          result_pypdf2 = calculate_roe_pypdf2[self.filename]
64
65          self.assertEqual(len(result_pypdf2), 0, 'ROE por
               cálculo: tamanho resultado (pypdf2)')
66
67
68  if __name__ == '__main__':
69      unittest.main()
```

repositorio/tests/reports/pypdf2/pypdf2_test_weg20172T.py

```
1   import unittest
2   from src import manager as m
3
4
5   class TestsWeg20172T(unittest.TestCase):
6
7       filename = None
8       manager_pypdf2 = None
9
10      @classmethod
11      def setUpClass(cls):
12          cls.filename = 'weg_2017_2T.pdf'
13          cls.manager_pypdf2 = m.manager([cls.filename], 'pypdf2')
14
15      @classmethod
16      def tearDownClass(cls):
17          cls.filename = None
18          cls.manager_pypdf2 = None
19
20      def test_lucro_liquido_monetary(self):
21          lucro_liquido_monetary_pypdf2 =
               self.manager_pypdf2.run_lucro_liquido_monetary()
22          result_pypdf2 =
               lucro_liquido_monetary_pypdf2[self.filename]
23
```

```
24          self.assertEqual(len(result_pypdf2), 0, 'lucro líquido
               (R$): tamanho resultado (pypdf2)')
25
26      def test_lucro_liquido_number(self):
27          lucro_liquido_number_pypdf2 =
               self.manager_pypdf2.run_lucro_liquido_number()
28          result_pypdf2 =
               lucro_liquido_number_pypdf2[self.filename]
29
30          self.assertEqual(len(result_pypdf2), 0, 'lucro líquido
               (número após conjunto de busca): tamanho resultado')
31
32      def test_patrimonio_liquido_monetary(self):
33          patrimonio_liquido_monetary =
               self.manager_pypdf2.run_patrimonio_liquido_monetary()
34          result = patrimonio_liquido_monetary[self.filename]
35
36          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
37
38      def test_patrimonio_liquido_number(self):
39          patrimonio_liquido_number_pypdf2 =
               self.manager_pypdf2.run_patrimonio_liquido_number()
40          result_pypdf2 =
               patrimonio_liquido_number_pypdf2[self.filename]
41
42          self.assertEqual(len(result_pypdf2), 0,
43                           'patrimônio líquido (número após
                              conjunto de busca): tamanho resultado
                              (pypdf2)')
44
45      def test_roe_monetary(self):
46          roe_monetary = self.manager_pypdf2.run_roe_monetary()
47          result = roe_monetary[self.filename]
48
49          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
50
51      def test_roe_number(self):
52          roe_number = self.manager_pypdf2.run_roe_number()
53          result = roe_number[self.filename]
54
```

```
55          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
56
57      def test_roe_calculate(self):
58          roe_calculate_pypdf2 =
               self.manager_pypdf2.run_calculate_roe()
59          result_pypdf2 = roe_calculate_pypdf2[self.filename]
60
61          self.assertEqual(len(result_pypdf2), 0, 'ROE por
               cálculo: tamanho resultado (pypdf2)')
62
63
64  if __name__ == '__main__':
65      unittest.main()
```

repositorio/tests/reports/pypdf2/pypdf2_test_weg20192T.py

```
1   import unittest
2   from src import manager as m
3   from src.helper import result_helper as rh
4   from data import data as data
5
6
7   class TestsWeg20192T(unittest.TestCase):
8
9       filename = None
10      manager_pypdf2 = None
11
12      @classmethod
13      def setUpClass(cls):
14          cls.filename = 'weg_2019_2T.pdf'
15          cls.manager_pypdf2 = m.manager([cls.filename], 'pypdf2')
16
17      @classmethod
18      def tearDownClass(cls):
19          cls.filename = None
20          cls.manager_pypdf2 = None
21
22      def test_lucro_liquido_monetary(self):
23          lucro_liquido_monetary_pypdf2 =
               self.manager_pypdf2.run_lucro_liquido_monetary()
24          result_pypdf2 =
```

```
            lucro_liquido_monetary_pypdf2[self.filename]
25          numbers_from_result_pypdf2 =
            rh.result_helper.get_numbers_as_list(result_pypdf2)
26
27          self.assertEqual(len(result_pypdf2), 1, 'lucro líquido
            (R$): tamanho resultado (pypdf2)')
28          self.assertIn(data.LUCRO_LIQUIDO[self.filename],
            numbers_from_result_pypdf2,
29                        'lucro líquido (R$): valor (pypdf2)')
30
31      def test_lucro_liquido_number(self):
32          lucro_liquido_number_pypdf2 =
            self.manager_pypdf2.run_lucro_liquido_number()
33          result_pypdf2 =
            lucro_liquido_number_pypdf2[self.filename]
34          numbers_from_result_pypdf2 =
            rh.result_helper.get_numbers_as_list(result_pypdf2)
35
36          self.assertEqual(len(result_pypdf2), 1,
37                           'lucro líquido (número após conjunto de
                              busca): tamanho resultado (pypdf2)')
38          self.assertNotIn(data.LUCRO_LIQUIDO[self.filename],
            numbers_from_result_pypdf2,
39                           'lucro líquido (número após conjunto de
                              busca): valor (pypdf2)')
40
41      def test_patrimonio_liquido_monetary(self):
42          patrimonio_liquido_monetary =
            self.manager_pypdf2.run_patrimonio_liquido_monetary()
43          result = patrimonio_liquido_monetary[self.filename]
44
45          self.assertEqual(len(result), 0, 'patrimônio líquido
            (R$): tamanho resultado')
46
47      def test_patrimonio_liquido_number(self):
48          patrimonio_liquido_number_pypdf2 =
            self.manager_pypdf2.run_patrimonio_liquido_number()
49          result_pypdf2 =
            patrimonio_liquido_number_pypdf2[self.filename]
50
51          self.assertEqual(len(result_pypdf2), 0,
52                           'patrimônio líquido (número após
```

```
                                conjunto de busca): tamanho resultado
                                (pypdf2)')
53
54      def test_roe_monetary(self):
55          roe_monetary = self.manager_pypdf2.run_roe_monetary()
56          result = roe_monetary[self.filename]
57
58          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
                resultado')
59
60      def test_roe_number(self):
61          roe_number = self.manager_pypdf2.run_roe_number()
62          result = roe_number[self.filename]
63
64          self.assertEqual(len(result), 0, 'ROE (número após
                conjunto de busca): tamanho resultado')
65
66      def test_roe_calculate(self):
67          roe_calculate_pypdf2 =
                self.manager_pypdf2.run_calculate_roe()
68          result_pypdf2 = roe_calculate_pypdf2[self.filename]
69
70          self.assertEqual(len(result_pypdf2), 0, 'ROE por
                cálculo: tamanho resultado (pypdf2)')
71
72
73  if __name__ == '__main__':
74      unittest.main()
```

repositorio/tests/reports/pytesseract/py0_test_ambev20191T.py

```
1   import unittest
2   from src import manager as m
3   from src.helper import result_helper as rh
4   from data import data as data
5
6
7   class TestsAmbev20191T(unittest.TestCase):
8
9       filename = None
10      manager_pytesseract = None
11
```

```
12      @classmethod
13      def setUpClass(cls):
14          cls.filename = 'ambev_2019_1T.pdf'
15          cls.manager_pytesseract = m.manager([cls.filename],
                'pytesseract')
16
17      @classmethod
18      def tearDownClass(cls):
19          cls.filename = None
20          cls.manager_pytesseract = None
21
22      def test_lucro_liquido_monetary(self):
23          lucro_liquido_monetary_pytesseract =
                self.manager_pytesseract.run_lucro_liquido_monetary()
24          result_pytesseract =
                lucro_liquido_monetary_pytesseract[self.filename]
25          numbers_from_result_pytesseract =
                rh.result_helper.get_numbers_as_list(result_pytesseract)
26
27          self.assertEqual(len(result_pytesseract), 4, 'lucro
                líquido (R$): tamanho resultado (pytesseract)')
28          self.assertNotIn(data.LUCRO_LIQUIDO[self.filename],
                numbers_from_result_pytesseract,
29                           'lucro líquido (R$): valor
                                (pytesseract)')
30
31      def test_lucro_liquido_number(self):
32          lucro_liquido_number_pytesseract =
                self.manager_pytesseract.run_lucro_liquido_number()
33          result_pytesseract =
                lucro_liquido_number_pytesseract[self.filename]
34          numbers_from_result =
                rh.result_helper.get_numbers_as_list(result_pytesseract)
35
36          self.assertEqual(len(result_pytesseract), 1, 'lucro
                líquido (número após conjunto de busca): tamanho
                resultado')
37          self.assertNotIn(data.LUCRO_LIQUIDO[self.filename],
                numbers_from_result,
38                           'lucro líquido (número após conjunto de
                                busca): valor')
39
```

```
40      def test_patrimonio_liquido_monetary(self):
41          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self
42
43          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
44
45      def test_patrimonio_liquido_number(self):
46          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
47          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
48          number_from_result =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
49
50          self.assertEqual(len(result_pytesseract), 2,
51                           'patrimônio líquido (número após
                              conjunto de busca): tamanho
                              resultado')
52          self.assertNotIn(data.PATRIMONIO_LIQUIDO[self.filename],
               number_from_result,
53                           'patrimônio líquido (número após
                              conjunto de busca): valor')
54
55      def test_roe_monetary(self):
56          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
57
58          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
59
60      def test_roe_number(self):
61          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
62
63          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
64
65      def test_roe_calculate(self):
66          calculate_roe_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
67          result_pytesseract =
```

```
            calculate_roe_pytesseract[self.filename]
68
69          self.assertEqual(len(result_pytesseract), 10, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
70          self.assertNotIn(data.ROE[self.filename],
               result_pytesseract, 'ROE por cálculo: valor')
71
72
73  if __name__ == '__main__':
74      unittest.main()
```

repositorio/tests/reports/pytesseract/py0_test_engie20192T.py

```
1   import unittest
2   from src import manager as m
3   from src.helper import result_helper as rh
4   from data import data as data
5
6
7   class TestsEngie20192T(unittest.TestCase):
8
9       filename = None
10      manager_pytesseract = None
11
12      @classmethod
13      def setUpClass(cls):
14          cls.filename = 'engie_2019_2T.pdf'
15          cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract')
16
17      @classmethod
18      def tearDownClass(cls):
19          cls.filename = None
20          cls.manager_pytesseract = None
21
22      def test_lucro_liquido_monetary(self):
23          lucro_liquido_monetary_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_monetary()
24          result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
25          numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
```

```
26
27          self.assertEqual(len(result_pytesseract), 6, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
28          self.assertIn(data.LUCRO_LIQUIDO[self.filename],
29                        numbers_from_result_pytesseract,
30                        'lucro líquido (R$): valor (pytesseract)')
31
32      def test_lucro_liquido_number(self):
33          lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
34          result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
35          numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
36
37          self.assertEqual(len(result_pytesseract), 2, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
38          self.assertIn(data.LUCRO_LIQUIDO[self.filename],
39                        numbers_from_result_pytesseract,
40                        'lucro líquido (R$): valor (pytesseract)')
41
42      def test_patrimonio_liquido_monetary(self):
43          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self
44
45          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
46
47      def test_patrimonio_liquido_number(self):
48          result =
               self.manager_pytesseract.run_patrimonio_liquido_number()[self.f
49
50          self.assertEqual(len(result), 0, 'patrimônio líquido
               (número após conjunto de busca): tamanho resultado')
51
52      def test_roe_monetary(self):
53          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
54
55          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
56
```

```
57      def test_roe_number(self):
58          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
59
60          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
61
62      def test_roe_calculate(self):
63          result =
               self.manager_pytesseract.run_calculate_roe()[self.filename]
64
65          self.assertEqual(len(result), 0, 'ROE por cálculo:
               tamanho resultado')
66
67
68  if __name__ == '__main__':
69      unittest.main()
```

repositorio/tests/reports/pytesseract/py0_test_engie20202T.py

```
1   import unittest
2   from src import manager as m
3   from src.helper import result_helper as rh
4   from data import data as data
5
6
7   class TestsEngie20202T(unittest.TestCase):
8
9       filename = None
10      manager_pytesseract = None
11
12      @classmethod
13      def setUpClass(cls):
14          cls.filename = 'engie_2020_2T.pdf'
15          cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract')
16
17      @classmethod
18      def tearDownClass(cls):
19          cls.filename = None
20          cls.manager_pytesseract = None
21
```

```
22      def test_lucro_liquido_monetary(self):
23          lucro_liquido_monetary_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_monetary()
24          result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
25          numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
26
27          self.assertEqual(len(result_pytesseract), 5, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
28          self.assertIn(data.LUCRO_LIQUIDO[self.filename],
29                        numbers_from_result_pytesseract,
30                        'lucro líquido (R$): valor (pytesseract)')
31
32      def test_lucro_liquido_number(self):
33          lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
34          result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
35
36          self.assertEqual(len(result_pytesseract), 0, 'lucro
               líquido (número após conjunto de busca): tamanho
               resultado')
37
38      def test_patrimonio_liquido_monetary(self):
39          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self
40
41          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
42
43      def test_patrimonio_liquido_number(self):
44          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
45          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
46
47          self.assertEqual(len(result_pytesseract), 1,
48                           'patrimônio líquido (número após
                              conjunto de busca): tamanho resultado
                              (pytesseract)')
49          self.assertEqual(result_pytesseract[0]['number'],
```

```
            data.PATRIMONIO_LIQUIDO[self.filename],
50                          'patrimônio líquido (número após
                               conjunto de busca): valor
                               (pytesseract)')
51
52      def test_roe_monetary(self):
53          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
54
55          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
56
57      def test_roe_number(self):
58          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
59
60          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
61
62      def test_roe_calculate(self):
63          roe_calculate_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
64          result_pytesseract =
               roe_calculate_pytesseract[self.filename]
65
66          self.assertEqual(len(result_pytesseract), 5, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
67          self.assertIn(data.ROE[self.filename],
               result_pytesseract, 'ROE por cálculo: valor
               (pytesseract)')
68
69
70  if __name__ == '__main__':
71      unittest.main()
```

repositorio/tests/reports/pytesseract/py0_test_fleury20193T.py

```
1  import unittest
2  from src import manager as m
3  from src.helper import result_helper as rh
4  from data import data as data
5
```

```
6
7  class TestsFleury20193T(unittest.TestCase):
8
9      filename = None
10     manager_pytesseract = None
11
12     @classmethod
13     def setUpClass(cls):
14         cls.filename = 'fleury_2019_3T.pdf'
15         cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract')
16
17     @classmethod
18     def tearDownClass(cls):
19         cls.filename = None
20         cls.manager_pytesseract = None
21
22     def test_lucro_liquido_monetary(self):
23         result =
               self.manager_pytesseract.run_lucro_liquido_monetary()[self.file
24
25         self.assertEqual(len(result), 0, 'lucro líquido (R$):
               tamanho resultado')
26
27     def test_lucro_liquido_number(self):
28         lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
29         result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
30         numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
31
32         self.assertEqual(len(result_pytesseract), 1,
33                          'lucro líquido (número após conjunto de
                                busca): tamanho resultado
                                (pytesseract)')
34         self.assertIn(data.LUCRO_LIQUIDO[self.filename],
               numbers_from_result_pytesseract,
35                       'lucro líquido (número após conjunto de
                             busca): valor (pytesseract)')
36
37     def test_patrimonio_liquido_monetary(self):
```

```
38          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
39
40          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
41
42      def test_patrimonio_liquido_number(self):
43          result =
               self.manager_pytesseract.run_patrimonio_liquido_number()[self.filena
44          number_from_result =
               rh.result_helper.get_numbers_as_list(result)
45
46          self.assertEqual(len(result), 3, 'patrimônio líquido
               (número após conjunto de busca): tamanho resultado')
47          self.assertIn(data.PATRIMONIO_LIQUIDO[self.filename],
               number_from_result,
48                        'patrimônio líquido (número após conjunto
                           de busca): valor')
49
50      def test_roe_monetary(self):
51          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
52
53          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
54
55      def test_roe_number(self):
56          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
57
58          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
59
60      def test_roe_calculate(self):
61          result =
               self.manager_pytesseract.run_calculate_roe()[self.filename]
62
63          self.assertEqual(len(result), 3, 'ROE por cálculo:
               tamanho resultado')
64          self.assertIn(data.ROE[self.filename], result, 'ROE por
               cálculo: valor')
65
```

```
66
67 if __name__ == '__main__':
68     unittest.main()
```

repositorio/tests/reports/pytesseract/py0_test_fleury20202T.py

```
1  import unittest
2  from src import manager as m
3
4
5  class TestsFleury20202T(unittest.TestCase):
6
7      filename = None
8      manager_pytesseract = None
9
10     @classmethod
11     def setUpClass(cls):
12         cls.filename = 'fleury_2020_2T.pdf'
13         cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract')
14
15     @classmethod
16     def tearDownClass(cls):
17         cls.filename = None
18         cls.manager_pytesseract = None
19
20     def test_lucro_liquido_monetary(self):
21         lucro_liquido_monetary_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_monetary()
22         result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
23
24         self.assertEqual(len(result_pytesseract), 0, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
25
26     def test_lucro_liquido_number(self):
27         result =
               self.manager_pytesseract.run_lucro_liquido_number()[self.filenar
28
29         self.assertEqual(len(result), 0, 'lucro líquido (número
               após conjunto de busca): tamanho resultado')
30
```

```
31      def test_patrimonio_liquido_monetary(self):
32          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
33
34          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
35
36      def test_patrimonio_liquido_number(self):
37          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
38          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
39
40          self.assertEqual(len(result_pytesseract), 0,
41                           'patrimônio líquido (número após
                              conjunto de busca): tamanho resultado
                              (pytesseract)')
42
43      def test_roe_monetary(self):
44          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
45
46          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
47
48      def test_roe_number(self):
49          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
50
51          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
52
53      def test_roe_calculate(self):
54          roe_calculate_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
55          result_pytesseract =
               roe_calculate_pytesseract[self.filename]
56
57          self.assertEqual(len(result_pytesseract), 0, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
58
59
```

```
60 if __name__ == '__main__':
61     unittest.main()
```

repositorio/tests/reports/pytesseract/py0_test_gerdau20153T.py

```
1  import unittest
2  from src import manager as m
3  from src.helper import result_helper as rh
4  from data import data as data
5
6
7  class TestsGerdau20153T(unittest.TestCase):
8
9      filename = None
10     manager_pytesseract = None
11
12     @classmethod
13     def setUpClass(cls):
14         cls.filename = 'gerdau_2015_3T.pdf'
15         cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract')
16
17     @classmethod
18     def tearDownClass(cls):
19         cls.filename = None
20         cls.manager_pytesseract = None
21
22     def test_lucro_liquido_monetary(self):
23         result =
               self.manager_pytesseract.run_lucro_liquido_monetary()[self.file
24
25         self.assertEqual(len(result), 0, 'lucro líquido (R$):
               tamanho resultado')
26
27     def test_lucro_liquido_number(self):
28         result =
               self.manager_pytesseract.run_lucro_liquido_number()[self.filena
29
30         self.assertEqual(len(result), 0, 'lucro líquido (número
               após conjunto de busca): tamanho resultado')
31
32     def test_patrimonio_liquido_monetary(self):
```

```
33          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
34
35          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
36
37      def test_patrimonio_liquido_number(self):
38          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
39          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
40          numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
41
42          self.assertEqual(len(result_pytesseract), 2,
43                           'patrimônio líquido (número após
                              conjunto de busca): tamanho resultado
                              (pytesseract)')
44          self.assertIn(data.PATRIMONIO_LIQUIDO[self.filename],
               numbers_from_result_pytesseract,
45                        'patrimônio líquido (número após conjunto
                             de busca): valor (pytesseract)')
46
47      def test_roe_monetary(self):
48          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
49
50          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
51
52      def test_roe_number(self):
53          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
54
55          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
56
57      def test_roe_calculate(self):
58          result =
               self.manager_pytesseract.run_calculate_roe()[self.filename]
59
60          self.assertEqual(len(result), 0, 'ROE por cálculo:
```

```
            tamanho resultado')


if __name__ == '__main__':
    unittest.main()
```

repositorio/tests/reports/pytesseract/py0_test_gerdau20171T.py

```
import unittest
from src import manager as m
from src.helper import result_helper as rh
from data import data as data


class TestsGerdau20171T(unittest.TestCase):

    filename = None
    manager_pytesseract = None

    @classmethod
    def setUpClass(cls):
        cls.filename = 'gerdau_2017_1T.pdf'
        cls.manager_pytesseract = m.manager([cls.filename],
            'pytesseract')

    @classmethod
    def tearDownClass(cls):
        cls.filename = None
        cls.manager_pytesseract = None

    def test_lucro_liquido_monetary(self):
        result =
            self.manager_pytesseract.run_lucro_liquido_monetary()[self.file

        self.assertEqual(len(result), 0, 'lucro líquido (R$):
            tamanho resultado')

    def test_lucro_liquido_number(self):
        lucro_liquido_number_pytesseract =
            self.manager_pytesseract.run_lucro_liquido_number()
        result_pytesseract =
            lucro_liquido_number_pytesseract[self.filename]
```

```
30
31          self.assertEqual(len(result_pytesseract), 0,
32                           'lucro líquido (número após conjunto de
                                busca): tamanho resultado
                                (pytesseract)')
33
34      def test_patrimonio_liquido_monetary(self):
35          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
36
37          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
38
39      def test_patrimonio_liquido_number(self):
40          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
41          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
42          numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
43
44          self.assertEqual(len(result_pytesseract), 2,
45                           'patrimônio líquido (número após
                                conjunto de busca): tamanho resultado
                                (pytesseract)')
46          self.assertIn(data.PATRIMONIO_LIQUIDO[self.filename],
               numbers_from_result_pytesseract,
47                        'patrimônio líquido (número após conjunto
                             de busca): valor (pytesseract)')
48
49      def test_roe_monetary(self):
50          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
51
52          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
53
54      def test_roe_number(self):
55          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
56
57          self.assertEqual(len(result), 0, 'ROE (número após
```

```
            conjunto de busca): tamanho resultado')

    def test_roe_calculate(self):
        calculate_roe_pytesseract =
            self.manager_pytesseract.run_calculate_roe()
        result_pytesseract =
            calculate_roe_pytesseract[self.filename]

        self.assertEqual(len(result_pytesseract), 0, 'ROE por
            cálculo: tamanho resultado (pytesseract)')


if __name__ == '__main__':
    unittest.main()
```

repositorio/tests/reports/pytesseract/py0_test_gerdau20184T.py

```
import unittest
from src import manager as m


class TestsGerdau20184T(unittest.TestCase):

    filename = None
    manager_pytesseract = None

    @classmethod
    def setUpClass(cls):
        cls.filename = 'gerdau_2018_4T.pdf'
        cls.manager_pytesseract = m.manager([cls.filename],
            'pytesseract')

    @classmethod
    def tearDownClass(cls):
        cls.filename = None
        cls.manager_pytesseract = None

    def test_lucro_liquido_monetary(self):
        result =
            self.manager_pytesseract.run_lucro_liquido_monetary()[self.filen

        self.assertEqual(len(result), 0, 'lucro líquido (R$):
```

```
            tamanho resultado')
24
25      def test_lucro_liquido_number(self):
26          lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
27          result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
28
29          self.assertEqual(len(result_pytesseract), 0,
30                           'lucro líquido (número após conjunto de
                                busca): tamanho resultado
                                (pytesseract)')
31
32      def test_patrimonio_liquido_monetary(self):
33          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
34
35          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
36
37      def test_patrimonio_liquido_number(self):
38          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
39          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
40
41          self.assertEqual(len(result_pytesseract), 0,
42                           'patrimônio líquido (número após
                                conjunto de busca): tamanho resultado
                                (pytesseract)')
43
44      def test_roe_monetary(self):
45          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
46
47          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
48
49      def test_roe_number(self):
50          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
51
```

```
52          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
53
54      def test_roe_calculate(self):
55          calculate_roe_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
56          result_pytesseract =
               calculate_roe_pytesseract[self.filename]
57
58          self.assertEqual(len(result_pytesseract), 0, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
59
60
61  if __name__ == '__main__':
62      unittest.main()
```

repositorio/tests/reports/pytesseract/py0_test_gerdau20192T.py

```
1   import unittest
2   from src import manager as m
3   from src.helper import result_helper as rh
4   from data import data as data
5
6
7   class TestsGerdau20192T(unittest.TestCase):
8
9       filename = None
10      manager_pytesseract = None
11
12      @classmethod
13      def setUpClass(cls):
14          cls.filename = 'gerdau_2019_2T.pdf'
15          cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract')
16
17      @classmethod
18      def tearDownClass(cls):
19          cls.filename = None
20          cls.manager_pytesseract = None
21
22      def test_lucro_liquido_monetary(self):
23          result =
```

```
            self.manager_pytesseract.run_lucro_liquido_monetary()[self.filename]
24
25          self.assertEqual(len(result), 0, 'lucro líquido (R$):
                tamanho resultado')
26
27      def test_lucro_liquido_number(self):
28          lucro_liquido_number_pytesseract =
                self.manager_pytesseract.run_lucro_liquido_number()
29          result_pytesseract =
                lucro_liquido_number_pytesseract[self.filename]
30          numbers_from_result_pytesseract =
                rh.result_helper.get_numbers_as_list(result_pytesseract)
31
32          self.assertEqual(len(result_pytesseract), 1,
33                           'lucro líquido (número após conjunto de
                                busca): tamanho resultado
                                (pytesseract)')
34          self.assertNotIn(data.LUCRO_LIQUIDO[self.filename],
                numbers_from_result_pytesseract,
35                           'lucro líquido (número após conjunto de
                                busca): valor (pytesseract)')
36
37      def test_patrimonio_liquido_monetary(self):
38          result =
                self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
39
40          self.assertEqual(len(result), 0, 'patrimônio líquido
                (R$): tamanho resultado')
41
42      def test_patrimonio_liquido_number(self):
43          patrimonio_liquido_number_pytesseract =
                self.manager_pytesseract.run_patrimonio_liquido_number()
44          result_pytesseract =
                patrimonio_liquido_number_pytesseract[self.filename]
45
46          self.assertEqual(len(result_pytesseract), 0,
47                           'patrimônio líquido (número após
                                conjunto de busca): tamanho resultado
                                (pytesseract)')
48
49      def test_roe_monetary(self):
50          result =
```

```
            self.manager_pytesseract.run_roe_monetary()[self.filename]
51
52          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
                resultado')
53
54      def test_roe_number(self):
55          result =
                self.manager_pytesseract.run_roe_number()[self.filename]
56
57          self.assertEqual(len(result), 0, 'ROE (número após
                conjunto de busca): tamanho resultado')
58
59      def test_roe_calculate(self):
60          calculate_roe_pytesseract =
                self.manager_pytesseract.run_calculate_roe()
61          result_pytesseract =
                calculate_roe_pytesseract[self.filename]
62
63          self.assertEqual(len(result_pytesseract), 0, 'ROE por
                cálculo: tamanho resultado (pytesseract)')
64
65
66  if __name__ == '__main__':
67      unittest.main()
```

repositorio/tests/reports/pytesseract/py0_test_weg20102T.py

```
1   import unittest
2   from src import manager as m
3   from src.helper import result_helper as rh
4   from data import data as data
5
6
7   class TestsWeg20102T(unittest.TestCase):
8
9       filename = None
10      manager_pytesseract = None
11
12      @classmethod
13      def setUpClass(cls):
14          cls.filename = 'weg_2010_2T.pdf'
15          cls.manager_pytesseract = m.manager([cls.filename],
```

```
            'pytesseract')
16
17      @classmethod
18      def tearDownClass(cls):
19          cls.filename = None
20          cls.manager_pytesseract = None
21
22      def test_lucro_liquido_monetary(self):
23          lucro_liquido_monetary_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_monetary()
24          result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
25          numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
26
27          self.assertEqual(len(result_pytesseract), 1, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
28          self.assertIn(data.LUCRO_LIQUIDO[self.filename],
               numbers_from_result_pytesseract,
29                        'lucro líquido (R$): valor (pytesseract)')
30
31      def test_lucro_liquido_number(self):
32          result =
               self.manager_pytesseract.run_lucro_liquido_number()[self.filename]
33
34          self.assertEqual(len(result), 0, 'lucro líquido (número
               após conjunto de busca): tamanho resultado')
35
36      def test_patrimonio_liquido_monetary(self):
37          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
38
39          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
40
41      def test_patrimonio_liquido_number(self):
42          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
43          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
44
45          self.assertEqual(len(result_pytesseract), 0,
```

```
46                          'patrimônio líquido (número após
                               conjunto de busca): tamanho resultado
                               (pytesseract)')
47
48      def test_roe_monetary(self):
49          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
50
51          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
52
53      def test_roe_number(self):
54          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
55
56          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
57
58      def test_roe_calculate(self):
59          roe_calculate_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
60          result_pytesseract =
               roe_calculate_pytesseract[self.filename]
61
62          self.assertEqual(len(result_pytesseract), 0, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
63
64
65  if __name__ == '__main__':
66      unittest.main()
```

repositorio/tests/reports/pytesseract/py0_test_weg20151T.py

```
1  import unittest
2  from src import manager as m
3  from src.helper import result_helper as rh
4  from data import data as data
5
6
7  class TestsWeg20151T(unittest.TestCase):
8
9      filename = None
```

```
10      manager_pytesseract = None
11
12      @classmethod
13      def setUpClass(cls):
14          cls.filename = 'weg_2015_1T.pdf'
15          cls.manager_pytesseract = m.manager([cls.filename],
                'pytesseract')
16
17      @classmethod
18      def tearDownClass(cls):
19          cls.filename = None
20          cls.manager_pytesseract = None
21
22      def test_lucro_liquido_monetary(self):
23          lucro_liquido_monetary_pytesseract =
                self.manager_pytesseract.run_lucro_liquido_monetary()
24          result_pytesseract =
                lucro_liquido_monetary_pytesseract[self.filename]
25          numbers_from_result_pytesseract =
                rh.result_helper.get_numbers_as_list(result_pytesseract)
26
27          self.assertEqual(len(result_pytesseract), 1, 'lucro
                líquido (R$): tamanho resultado (pytesseract)')
28          self.assertIn(data.LUCRO_LIQUIDO[self.filename],
                numbers_from_result_pytesseract,
29                        'lucro líquido (R$): valor (pytesseract)')
30
31      def test_lucro_liquido_number(self):
32          result =
                self.manager_pytesseract.run_lucro_liquido_number()[self.filename]
33
34          self.assertEqual(len(result), 0, 'lucro líquido (número
                após conjunto de busca): tamanho resultado')
35
36      def test_patrimonio_liquido_monetary(self):
37          result =
                self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
38
39          self.assertEqual(len(result), 0, 'patrimônio líquido
                (R$): tamanho resultado')
40
41      def test_patrimonio_liquido_number(self):
```

```
42          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
43          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
44          numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
45
46          self.assertEqual(len(result_pytesseract), 1,
47                           'patrimônio líquido (número após
                                conjunto de busca): tamanho resultado
                                (pytesseract)')
48          self.assertNotIn(data.PATRIMONIO_LIQUIDO[self.filename],
               numbers_from_result_pytesseract,
49                           'patrimônio líquido (número após
                                conjunto de busca): valor
                                (pytesseract)')
50
51      def test_roe_monetary(self):
52          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
53
54          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
55
56      def test_roe_number(self):
57          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
58
59          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
60
61      def test_roe_calculate(self):
62          calculate_roe_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
63          result_pytesseract =
               calculate_roe_pytesseract[self.filename]
64
65          self.assertEqual(len(result_pytesseract), 1, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
66          self.assertNotIn(data.ROE[self.filename],
               result_pytesseract,
67                           'ROE por cálculo: tamanho resultado
```

```
                                (pytesseract)')
68
69
70 if __name__ == '__main__':
71     unittest.main()
```

repositorio/tests/reports/pytesseract/py0_test_weg20172T.py

```
1  import unittest
2  from src import manager as m
3  from src.helper import result_helper as rh
4  from data import data as data
5
6
7  class TestsWeg20172T(unittest.TestCase):
8
9      filename = None
10     manager_pytesseract = None
11
12     @classmethod
13     def setUpClass(cls):
14         cls.filename = 'weg_2017_2T.pdf'
15         cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract')
16
17     @classmethod
18     def tearDownClass(cls):
19         cls.filename = None
20         cls.manager_pytesseract = None
21
22     def test_lucro_liquido_monetary(self):
23         lucro_liquido_monetary_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_monetary()
24         result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
25         numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
26
27         self.assertEqual(len(result_pytesseract), 1, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
28         self.assertIn(data.LUCRO_LIQUIDO[self.filename],
               numbers_from_result_pytesseract,
```

```
29                      'lucro líquido (R$): valor (pytesseract)')
30
31      def test_lucro_liquido_number(self):
32          lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
33          result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
34          numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
35
36          self.assertEqual(len(result_pytesseract), 1,
37                           'lucro líquido (número após conjunto de
                                busca): tamanho resultado
                                (pytesseract)')
38          self.assertNotIn(data.LUCRO_LIQUIDO[self.filename],
               numbers_from_result_pytesseract,
39                           'lucro líquido (número após conjunto de
                                busca): valor (pytesseract)')
40
41      def test_patrimonio_liquido_monetary(self):
42          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self
43
44          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
45
46      def test_patrimonio_liquido_number(self):
47          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
48          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
49          numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
50
51          self.assertEqual(len(result_pytesseract), 1,
52                           'patrimônio líquido (número após
                                conjunto de busca): tamanho resultado
                                (pytesseract)')
53          self.assertIn(data.PATRIMONIO_LIQUIDO[self.filename],
               numbers_from_result_pytesseract,
54                        'patrimônio líquido (número após conjunto
                             de busca): valor (pytesseract)')
```

```
55
56      def test_roe_monetary(self):
57          result =
                self.manager_pytesseract.run_roe_monetary()[self.filename]
58
59          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
                resultado')
60
61      def test_roe_number(self):
62          result =
                self.manager_pytesseract.run_roe_number()[self.filename]
63
64          self.assertEqual(len(result), 0, 'ROE (número após
                conjunto de busca): tamanho resultado')
65
66      def test_roe_calculate(self):
67          roe_calculate_pytesseract =
                self.manager_pytesseract.run_calculate_roe()
68          result_pytesseract =
                roe_calculate_pytesseract[self.filename]
69
70          self.assertEqual(len(result_pytesseract), 2, 'ROE por
                cálculo: tamanho resultado (pytesseract)')
71          self.assertIn(data.ROE[self.filename],
                result_pytesseract, 'ROE por cálculo: valor')
72
73
74  if __name__ == '__main__':
75      unittest.main()
```

repositorio/tests/reports/pytesseract/py0_test_weg20192T.py

```
1   import unittest
2   from src import manager as m
3   from src.helper import result_helper as rh
4   from data import data as data
5
6
7   class TestsWeg20192T(unittest.TestCase):
8
9       filename = None
10      manager_pytesseract = None
```

```
11
12      @classmethod
13      def setUpClass(cls):
14          cls.filename = 'weg_2019_2T.pdf'
15          cls.manager_pytesseract = m.manager([cls.filename],
                'pytesseract')
16
17      @classmethod
18      def tearDownClass(cls):
19          cls.filename = None
20          cls.manager_pytesseract = None
21
22      def test_lucro_liquido_monetary(self):
23          lucro_liquido_monetary_pytesseract =
                self.manager_pytesseract.run_lucro_liquido_monetary()
24          result_pytesseract =
                lucro_liquido_monetary_pytesseract[self.filename]
25          numbers_from_result_pytesseract =
                rh.result_helper.get_numbers_as_list(result_pytesseract)
26
27          self.assertEqual(len(result_pytesseract), 1, 'lucro
                líquido (R$): tamanho resultado (pytesseract)')
28          self.assertIn(data.LUCRO_LIQUIDO[self.filename],
                numbers_from_result_pytesseract,
29                        'lucro líquido (R$): valor (pytesseract)')
30
31      def test_lucro_liquido_number(self):
32          lucro_liquido_number_pytesseract =
                self.manager_pytesseract.run_lucro_liquido_number()
33          result_pytesseract =
                lucro_liquido_number_pytesseract[self.filename]
34
35          self.assertEqual(len(result_pytesseract), 0,
36                           'lucro líquido (número após conjunto de
                                busca): tamanho resultado
                                (pytesseract)')
37
38      def test_patrimonio_liquido_monetary(self):
39          result =
                self.manager_pytesseract.run_patrimonio_liquido_monetary()[self
40
41          self.assertEqual(len(result), 0, 'patrimônio líquido
```

```
            (R$): tamanho resultado')
42
43      def test_patrimonio_liquido_number(self):
44          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
45          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
46          numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
47
48          self.assertEqual(len(result_pytesseract), 1,
49                           'patrimônio líquido (número após
                                conjunto de busca): tamanho resultado
                                (pytesseract)')
50          self.assertNotIn(data.PATRIMONIO_LIQUIDO[self.filename],
               numbers_from_result_pytesseract,
51                           'patrimônio líquido (número após
                                conjunto de busca): valor
                                (pytesseract)')
52
53      def test_roe_monetary(self):
54          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
55
56          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
57
58      def test_roe_number(self):
59          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
60
61          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
62
63      def test_roe_calculate(self):
64          roe_calculate_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
65          result_pytesseract =
               roe_calculate_pytesseract[self.filename]
66
67          self.assertEqual(len(result_pytesseract), 1, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
```

```
68          self.assertNotIn(data.ROE[self.filename],
               result_pytesseract, 'ROE por cálculo: valor')
69
70
71  if __name__ == '__main__':
72      unittest.main()
```

repositorio/tests/reports/pytesseract01/py1_test_ambev20191T.py

```
1   import unittest
2   from src import manager as m
3
4
5   class TestsAmbev20191T(unittest.TestCase):
6
7       filename = None
8       manager_pytesseract = None
9
10      @classmethod
11      def setUpClass(cls):
12          cls.filename = 'ambev_2019_1T.pdf'
13          cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 10 --oem 2')
14
15      @classmethod
16      def tearDownClass(cls):
17          cls.filename = None
18          cls.manager_pytesseract = None
19
20      def test_lucro_liquido_monetary(self):
21          lucro_liquido_monetary_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_monetary()
22          result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
23
24          self.assertEqual(len(result_pytesseract), 0, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
25
26      def test_lucro_liquido_number(self):
27          lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
28          result_pytesseract =
```

```
            lucro_liquido_number_pytesseract[self.filename]
29
30          self.assertEqual(len(result_pytesseract), 0, 'lucro
               líquido (número após conjunto de busca): tamanho
               resultado')
31
32      def test_patrimonio_liquido_monetary(self):
33          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
34
35          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
36
37      def test_patrimonio_liquido_number(self):
38          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
39          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
40
41          self.assertEqual(len(result_pytesseract), 0,
42                           'patrimônio líquido (número após
                              conjunto de busca): tamanho
                              resultado')
43
44      def test_roe_monetary(self):
45          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
46
47          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
48
49      def test_roe_number(self):
50          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
51
52          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
53
54      def test_roe_calculate(self):
55          calculate_roe_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
56          result_pytesseract =
```

```
            calculate_roe_pytesseract[self.filename]
57
58          self.assertEqual(len(result_pytesseract), 0, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
59
60
61  if __name__ == '__main__':
62      unittest.main()
```

repositorio/tests/reports/pytesseract01/py1_test_engie20192T.py

```
1   import unittest
2   from src import manager as m
3
4
5   class TestsEngie20192T(unittest.TestCase):
6
7       filename = None
8       manager_pytesseract = None
9
10      @classmethod
11      def setUpClass(cls):
12          cls.filename = 'engie_2019_2T.pdf'
13          cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 10 --oem 2')
14
15      @classmethod
16      def tearDownClass(cls):
17          cls.filename = None
18          cls.manager_pytesseract = None
19
20      def test_lucro_liquido_monetary(self):
21          lucro_liquido_monetary_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_monetary()
22          result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
23
24          self.assertEqual(len(result_pytesseract), 0, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
25
26      def test_lucro_liquido_number(self):
27          lucro_liquido_number_pytesseract =
```

```
            self.manager_pytesseract.run_lucro_liquido_number()
28          result_pytesseract =
            lucro_liquido_number_pytesseract[self.filename]
29
30          self.assertEqual(len(result_pytesseract), 0, 'lucro
            líquido (R$): tamanho resultado (pytesseract)')
31
32      def test_patrimonio_liquido_monetary(self):
33          result =
            self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
34
35          self.assertEqual(len(result), 0, 'patrimônio líquido
            (R$): tamanho resultado')
36
37      def test_patrimonio_liquido_number(self):
38          result =
            self.manager_pytesseract.run_patrimonio_liquido_number()[self.filena
39
40          self.assertEqual(len(result), 0, 'patrimônio líquido
            (número após conjunto de busca): tamanho resultado')
41
42      def test_roe_monetary(self):
43          result =
            self.manager_pytesseract.run_roe_monetary()[self.filename]
44
45          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
            resultado')
46
47      def test_roe_number(self):
48          result =
            self.manager_pytesseract.run_roe_number()[self.filename]
49
50          self.assertEqual(len(result), 0, 'ROE (número após
            conjunto de busca): tamanho resultado')
51
52      def test_roe_calculate(self):
53          result =
            self.manager_pytesseract.run_calculate_roe()[self.filename]
54
55          self.assertEqual(len(result), 0, 'ROE por cálculo:
            tamanho resultado')
56
```

```
57
58 if __name__ == '__main__':
59     unittest.main()
```

repositorio/tests/reports/pytesseract01/py1_test_engie20202T.py

```
1  import unittest
2  from src import manager as m
3
4
5  class TestsEngie20202T(unittest.TestCase):
6
7      filename = None
8      manager_pytesseract = None
9
10     @classmethod
11     def setUpClass(cls):
12         cls.filename = 'engie_2020_2T.pdf'
13         cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 10 --oem 2')
14
15     @classmethod
16     def tearDownClass(cls):
17         cls.filename = None
18         cls.manager_pytesseract = None
19
20     def test_lucro_liquido_monetary(self):
21         lucro_liquido_monetary_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_monetary()
22         result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
23
24         self.assertEqual(len(result_pytesseract), 0, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
25
26     def test_lucro_liquido_number(self):
27         lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
28         result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
29
30         self.assertEqual(len(result_pytesseract), 0, 'lucro
```

```
            líquido (número após conjunto de busca): tamanho
            resultado')
31
32      def test_patrimonio_liquido_monetary(self):
33          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
34
35          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
36
37      def test_patrimonio_liquido_number(self):
38          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
39          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
40
41          self.assertEqual(len(result_pytesseract), 0,
42                           'patrimônio líquido (número após
                              conjunto de busca): tamanho resultado
                              (pytesseract)')
43
44      def test_roe_monetary(self):
45          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
46
47          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
48
49      def test_roe_number(self):
50          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
51
52          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
53
54      def test_roe_calculate(self):
55          roe_calculate_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
56          result_pytesseract =
               roe_calculate_pytesseract[self.filename]
57
58          self.assertEqual(len(result_pytesseract), 0, 'ROE por
```

```
            cálculo: tamanho resultado (pytesseract)')
59
60
61  if __name__ == '__main__':
62      unittest.main()
```

repositorio/tests/reports/pytesseract01/py1_test_fleury20193T.py

```
1   import unittest
2   from src import manager as m
3
4
5   class TestsFleury20193T(unittest.TestCase):
6
7       filename = None
8       manager_pytesseract = None
9
10      @classmethod
11      def setUpClass(cls):
12          cls.filename = 'fleury_2019_3T.pdf'
13          cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 10 --oem 2')
14
15      @classmethod
16      def tearDownClass(cls):
17          cls.filename = None
18          cls.manager_pytesseract = None
19
20      def test_lucro_liquido_monetary(self):
21          result =
               self.manager_pytesseract.run_lucro_liquido_monetary()[self.filer
22
23          self.assertEqual(len(result), 0, 'lucro líquido (R$):
               tamanho resultado')
24
25      def test_lucro_liquido_number(self):
26          lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
27          result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
28
29          self.assertEqual(len(result_pytesseract), 0,
```

```
30                          'lucro líquido (número após conjunto de
                               busca): tamanho resultado
                               (pytesseract)')
31
32      def test_patrimonio_liquido_monetary(self):
33          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
34
35          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
36
37      def test_patrimonio_liquido_number(self):
38          result =
               self.manager_pytesseract.run_patrimonio_liquido_number()[self.filena
39
40          self.assertEqual(len(result), 0, 'patrimônio líquido
               (número após conjunto de busca): tamanho resultado')
41
42      def test_roe_monetary(self):
43          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
44
45          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
46
47      def test_roe_number(self):
48          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
49
50          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
51
52      def test_roe_calculate(self):
53          result =
               self.manager_pytesseract.run_calculate_roe()[self.filename]
54
55          self.assertEqual(len(result), 0, 'ROE por cálculo:
               tamanho resultado')
56
57
58  if __name__ == '__main__':
59      unittest.main()
```

repositorio/tests/reports/pytesseract01/py1_test_fleury20202T.py

```
import unittest
from src import manager as m


class TestsFleury20202T(unittest.TestCase):

    filename = None
    manager_pytesseract = None

    @classmethod
    def setUpClass(cls):
        cls.filename = 'fleury_2020_2T.pdf'
        cls.manager_pytesseract = m.manager([cls.filename],
           'pytesseract', '--psm 10 --oem 2')

    @classmethod
    def tearDownClass(cls):
        cls.filename = None
        cls.manager_pytesseract = None

    def test_lucro_liquido_monetary(self):
        lucro_liquido_monetary_pytesseract =
           self.manager_pytesseract.run_lucro_liquido_monetary()
        result_pytesseract =
           lucro_liquido_monetary_pytesseract[self.filename]

        self.assertEqual(len(result_pytesseract), 0, 'lucro
           líquido (R$): tamanho resultado (pytesseract)')

    def test_lucro_liquido_number(self):
        result =
           self.manager_pytesseract.run_lucro_liquido_number()[self.filenar

        self.assertEqual(len(result), 0, 'lucro líquido (número
           após conjunto de busca): tamanho resultado')

    def test_patrimonio_liquido_monetary(self):
        result =
           self.manager_pytesseract.run_patrimonio_liquido_monetary()[self
```

```
33
34          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
35
36      def test_patrimonio_liquido_number(self):
37          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
38          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
39
40          self.assertEqual(len(result_pytesseract), 0,
41                           'patrimônio líquido (número após
                              conjunto de busca): tamanho resultado
                              (pytesseract)')
42
43      def test_roe_monetary(self):
44          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
45
46          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
47
48      def test_roe_number(self):
49          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
50
51          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
52
53      def test_roe_calculate(self):
54          roe_calculate_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
55          result_pytesseract =
               roe_calculate_pytesseract[self.filename]
56
57          self.assertEqual(len(result_pytesseract), 0, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
58
59
60  if __name__ == '__main__':
61      unittest.main()
```

repositorio/tests/reports/pytesseract01/py1_test_gerdau20153T.py

```
1  import unittest
2  from src import manager as m
3
4
5  class TestsGerdau20153T(unittest.TestCase):
6
7      filename = None
8      manager_pytesseract = None
9
10     @classmethod
11     def setUpClass(cls):
12         cls.filename = 'gerdau_2015_3T.pdf'
13         cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 10 --oem 2')
14
15     @classmethod
16     def tearDownClass(cls):
17         cls.filename = None
18         cls.manager_pytesseract = None
19
20     def test_lucro_liquido_monetary(self):
21         result =
               self.manager_pytesseract.run_lucro_liquido_monetary()[self.file
22
23         self.assertEqual(len(result), 0, 'lucro líquido (R$):
               tamanho resultado')
24
25     def test_lucro_liquido_number(self):
26         result =
               self.manager_pytesseract.run_lucro_liquido_number()[self.filena
27
28         self.assertEqual(len(result), 0, 'lucro líquido (número
               após conjunto de busca): tamanho resultado')
29
30     def test_patrimonio_liquido_monetary(self):
31         result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self
32
33         self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
34
```

```
35      def test_patrimonio_liquido_number(self):
36          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
37          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
38
39          self.assertEqual(len(result_pytesseract), 0,
40                           'patrimônio líquido (número após
                                conjunto de busca): tamanho resultado
                                (pytesseract)')
41
42      def test_roe_monetary(self):
43          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
44
45          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
46
47      def test_roe_number(self):
48          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
49
50          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
51
52      def test_roe_calculate(self):
53          result =
               self.manager_pytesseract.run_calculate_roe()[self.filename]
54
55          self.assertEqual(len(result), 0, 'ROE por cálculo:
               tamanho resultado')
56
57
58  if __name__ == '__main__':
59      unittest.main()
```

repositorio/tests/reports/pytesseract01/py1_test_gerdau20171T.py

```
1  import unittest
2  from src import manager as m
3
4
```

```
5  class TestsGerdau20171T(unittest.TestCase):
6
7      filename = None
8      manager_pytesseract = None
9
10     @classmethod
11     def setUpClass(cls):
12         cls.filename = 'gerdau_2017_1T.pdf'
13         cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 10 --oem 2')
14
15     @classmethod
16     def tearDownClass(cls):
17         cls.filename = None
18         cls.manager_pytesseract = None
19
20     def test_lucro_liquido_monetary(self):
21         result =
               self.manager_pytesseract.run_lucro_liquido_monetary()[self.file
22
23         self.assertEqual(len(result), 0, 'lucro líquido (R$):
               tamanho resultado')
24
25     def test_lucro_liquido_number(self):
26         lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
27         result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
28
29         self.assertEqual(len(result_pytesseract), 0,
30                          'lucro líquido (número após conjunto de
                                busca): tamanho resultado
                                (pytesseract)')
31
32     def test_patrimonio_liquido_monetary(self):
33         result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self
34
35         self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
36
37     def test_patrimonio_liquido_number(self):
```

```
38          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
39          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
40
41          self.assertEqual(len(result_pytesseract), 0,
42                           'patrimônio líquido (número após
                                conjunto de busca): tamanho resultado
                                (pytesseract)')
43
44      def test_roe_monetary(self):
45          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
46
47          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
48
49      def test_roe_number(self):
50          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
51
52          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
53
54      def test_roe_calculate(self):
55          calculate_roe_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
56          result_pytesseract =
               calculate_roe_pytesseract[self.filename]
57
58          self.assertEqual(len(result_pytesseract), 0, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
59
60
61  if __name__ == '__main__':
62      unittest.main()
```

repositorio/tests/reports/pytesseract01/py1_test_gerdau20184T.py

```
1  import unittest
2  from src import manager as m
3
```

```
4
5  class TestsGerdau20184T(unittest.TestCase):
6
7      filename = None
8      manager_pytesseract = None
9
10     @classmethod
11     def setUpClass(cls):
12         cls.filename = 'gerdau_2018_4T.pdf'
13         cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 10 --oem 2')
14
15     @classmethod
16     def tearDownClass(cls):
17         cls.filename = None
18         cls.manager_pytesseract = None
19
20     def test_lucro_liquido_monetary(self):
21         result =
               self.manager_pytesseract.run_lucro_liquido_monetary()[self.file
22
23         self.assertEqual(len(result), 0, 'lucro líquido (R$):
               tamanho resultado')
24
25     def test_lucro_liquido_number(self):
26         lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
27         result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
28
29         self.assertEqual(len(result_pytesseract), 0,
30                          'lucro líquido (número após conjunto de
                                busca): tamanho resultado
                                (pytesseract)')
31
32     def test_patrimonio_liquido_monetary(self):
33         result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self
34
35         self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
36
```

```
37      def test_patrimonio_liquido_number(self):
38          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
39          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
40
41          self.assertEqual(len(result_pytesseract), 0,
42                           'patrimônio líquido (número após
                                conjunto de busca): tamanho resultado
                                (pytesseract)')
43
44      def test_roe_monetary(self):
45          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
46
47          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
48
49      def test_roe_number(self):
50          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
51
52          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
53
54      def test_roe_calculate(self):
55          calculate_roe_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
56          result_pytesseract =
               calculate_roe_pytesseract[self.filename]
57
58          self.assertEqual(len(result_pytesseract), 0, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
59
60
61  if __name__ == '__main__':
62      unittest.main()
```

repositorio/tests/reports/pytesseract01/py1_test_gerdau20192T.py

```
1  import unittest
2  from src import manager as m
```

```
3
4
5  class TestsGerdau20192T(unittest.TestCase):
6
7      filename = None
8      manager_pytesseract = None
9
10     @classmethod
11     def setUpClass(cls):
12         cls.filename = 'gerdau_2019_2T.pdf'
13         cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 10 --oem 2')
14
15     @classmethod
16     def tearDownClass(cls):
17         cls.filename = None
18         cls.manager_pytesseract = None
19
20     def test_lucro_liquido_monetary(self):
21         result =
               self.manager_pytesseract.run_lucro_liquido_monetary()[self.filen
22
23         self.assertEqual(len(result), 0, 'lucro líquido (R$):
               tamanho resultado')
24
25     def test_lucro_liquido_number(self):
26         lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
27         result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
28
29         self.assertEqual(len(result_pytesseract), 0,
30                          'lucro líquido (número após conjunto de
                                busca): tamanho resultado
                                (pytesseract)')
31
32     def test_patrimonio_liquido_monetary(self):
33         result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self
34
35         self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
```

```
36
37      def test_patrimonio_liquido_number(self):
38          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
39          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
40
41          self.assertEqual(len(result_pytesseract), 0,
42                           'patrimônio líquido (número após
                                conjunto de busca): tamanho resultado
                                (pytesseract)')
43
44      def test_roe_monetary(self):
45          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
46
47          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
48
49      def test_roe_number(self):
50          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
51
52          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
53
54      def test_roe_calculate(self):
55          calculate_roe_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
56          result_pytesseract =
               calculate_roe_pytesseract[self.filename]
57
58          self.assertEqual(len(result_pytesseract), 0, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
59
60
61  if __name__ == '__main__':
62      unittest.main()
```

repositorio/tests/reports/pytesseract01/py1_test_weg20102T.py

```
1  import unittest
```

```
2  from src import manager as m
3
4
5  class TestsWeg20102T(unittest.TestCase):
6
7      filename = None
8      manager_pytesseract = None
9
10     @classmethod
11     def setUpClass(cls):
12         cls.filename = 'weg_2010_2T.pdf'
13         cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 10 --oem 2')
14
15     @classmethod
16     def tearDownClass(cls):
17         cls.filename = None
18         cls.manager_pytesseract = None
19
20     def test_lucro_liquido_monetary(self):
21         lucro_liquido_monetary_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_monetary()
22         result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
23
24         self.assertEqual(len(result_pytesseract), 0, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
25
26     def test_lucro_liquido_number(self):
27         result =
               self.manager_pytesseract.run_lucro_liquido_number()[self.filenar
28
29         self.assertEqual(len(result), 0, 'lucro líquido (número
               após conjunto de busca): tamanho resultado')
30
31     def test_patrimonio_liquido_monetary(self):
32         result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self
33
34         self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
35
```

```
36      def test_patrimonio_liquido_number(self):
37          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
38          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
39
40          self.assertEqual(len(result_pytesseract), 0,
41                           'patrimônio líquido (número após
                                conjunto de busca): tamanho resultado
                                (pytesseract)')
42
43      def test_roe_monetary(self):
44          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
45
46          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
47
48      def test_roe_number(self):
49          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
50
51          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
52
53      def test_roe_calculate(self):
54          roe_calculate_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
55          result_pytesseract =
               roe_calculate_pytesseract[self.filename]
56
57          self.assertEqual(len(result_pytesseract), 0, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
58
59
60  if __name__ == '__main__':
61      unittest.main()
```

repositorio/tests/reports/pytesseract01/py1_test_weg20151T.py

```
1  import unittest
2  from src import manager as m
```

```
3
4
5  class TestsWeg20151T(unittest.TestCase):
6
7      filename = None
8      manager_pytesseract = None
9
10     @classmethod
11     def setUpClass(cls):
12         cls.filename = 'weg_2015_1T.pdf'
13         cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 10 --oem 2')
14
15     @classmethod
16     def tearDownClass(cls):
17         cls.filename = None
18         cls.manager_pytesseract = None
19
20     def test_lucro_liquido_monetary(self):
21         lucro_liquido_monetary_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_monetary()
22         result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
23
24         self.assertEqual(len(result_pytesseract), 0, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
25
26     def test_lucro_liquido_number(self):
27         result =
               self.manager_pytesseract.run_lucro_liquido_number()[self.filenar
28
29         self.assertEqual(len(result), 0, 'lucro líquido (número
               após conjunto de busca): tamanho resultado')
30
31     def test_patrimonio_liquido_monetary(self):
32         result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self
33
34         self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
35
36     def test_patrimonio_liquido_number(self):
```

```
37          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
38          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
39
40          self.assertEqual(len(result_pytesseract), 0,
41                           'patrimônio líquido (número após
                                conjunto de busca): tamanho resultado
                                (pytesseract)')
42
43      def test_roe_monetary(self):
44          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
45
46          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
47
48      def test_roe_number(self):
49          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
50
51          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
52
53      def test_roe_calculate(self):
54          calculate_roe_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
55          result_pytesseract =
               calculate_roe_pytesseract[self.filename]
56
57          self.assertEqual(len(result_pytesseract), 0, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
58
59
60  if __name__ == '__main__':
61      unittest.main()
```

repositorio/tests/reports/pytesseract01/py1_test_weg20172T.py

```
1  import unittest
2  from src import manager as m
3
```

```
4
5  class TestsWeg20172T(unittest.TestCase):
6
7      filename = None
8      manager_pytesseract = None
9
10     @classmethod
11     def setUpClass(cls):
12         cls.filename = 'weg_2017_2T.pdf'
13         cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 10 --oem 2')
14
15     @classmethod
16     def tearDownClass(cls):
17         cls.filename = None
18         cls.manager_pytesseract = None
19
20     def test_lucro_liquido_monetary(self):
21         lucro_liquido_monetary_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_monetary()
22         result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
23
24         self.assertEqual(len(result_pytesseract), 0, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
25
26     def test_lucro_liquido_number(self):
27         lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
28         result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
29
30         self.assertEqual(len(result_pytesseract), 0,
31                          'lucro líquido (número após conjunto de
                                busca): tamanho resultado
                                (pytesseract)')
32
33     def test_patrimonio_liquido_monetary(self):
34         result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self
35
36         self.assertEqual(len(result), 0, 'patrimônio líquido
```

```
            (R$): tamanho resultado')
37
38      def test_patrimonio_liquido_number(self):
39          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
40          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
41
42          self.assertEqual(len(result_pytesseract), 0,
43                           'patrimônio líquido (número após
                                conjunto de busca): tamanho resultado
                                (pytesseract)')
44
45      def test_roe_monetary(self):
46          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
47
48          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
49
50      def test_roe_number(self):
51          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
52
53          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
54
55      def test_roe_calculate(self):
56          roe_calculate_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
57          result_pytesseract =
               roe_calculate_pytesseract[self.filename]
58
59          self.assertEqual(len(result_pytesseract), 0, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
60
61
62  if __name__ == '__main__':
63      unittest.main()
```

repositorio/tests/reports/pytesseract01/py1_test_weg20192T.py

```
1  import unittest
2  from src import manager as m
3
4
5  class TestsWeg20192T(unittest.TestCase):
6
7      filename = None
8      manager_pytesseract = None
9
10     @classmethod
11     def setUpClass(cls):
12         cls.filename = 'weg_2019_2T.pdf'
13         cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 10 --oem 2')
14
15     @classmethod
16     def tearDownClass(cls):
17         cls.filename = None
18         cls.manager_pytesseract = None
19
20     def test_lucro_liquido_monetary(self):
21         lucro_liquido_monetary_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_monetary()
22         result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
23
24         self.assertEqual(len(result_pytesseract), 0, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
25
26     def test_lucro_liquido_number(self):
27         lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
28         result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
29
30         self.assertEqual(len(result_pytesseract), 0,
31                          'lucro líquido (número após conjunto de
                                busca): tamanho resultado
                                (pytesseract)')
32
33     def test_patrimonio_liquido_monetary(self):
34         result =
```

```
            self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
35
36          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
37
38      def test_patrimonio_liquido_number(self):
39          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
40          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
41
42          self.assertEqual(len(result_pytesseract), 0,
43                           'patrimônio líquido (número após
                              conjunto de busca): tamanho resultado
                              (pytesseract)')
44
45      def test_roe_monetary(self):
46          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
47
48          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
49
50      def test_roe_number(self):
51          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
52
53          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
54
55      def test_roe_calculate(self):
56          roe_calculate_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
57          result_pytesseract =
               roe_calculate_pytesseract[self.filename]
58
59          self.assertEqual(len(result_pytesseract), 0, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
60
61
62  if __name__ == '__main__':
63      unittest.main()
```

repositorio/tests/reports/pytesseract02/py2_test_ambev20191T.py

```
import unittest
from src import manager as m
from src.helper import result_helper as rh
from data import data as data


class TestsAmbev20191T(unittest.TestCase):

    filename = None
    manager_pytesseract = None

    @classmethod
    def setUpClass(cls):
        cls.filename = 'ambev_2019_1T.pdf'
        cls.manager_pytesseract = m.manager([cls.filename],
           'pytesseract', '--psm 6')

    @classmethod
    def tearDownClass(cls):
        cls.filename = None
        cls.manager_pytesseract = None

    def test_lucro_liquido_monetary(self):
        lucro_liquido_monetary_pytesseract =
           self.manager_pytesseract.run_lucro_liquido_monetary()
        result_pytesseract =
           lucro_liquido_monetary_pytesseract[self.filename]
        numbers_from_result_pytesseract =
           rh.result_helper.get_numbers_as_list(result_pytesseract)

        self.assertEqual(len(result_pytesseract), 4, 'lucro
           líquido (R$): tamanho resultado (pytesseract)')
        self.assertNotIn(data.LUCRO_LIQUIDO[self.filename],
           numbers_from_result_pytesseract,
                         'lucro líquido (R$): valor
                            (pytesseract)')

    def test_lucro_liquido_number(self):
        lucro_liquido_number_pytesseract =
```

```
            self.manager_pytesseract.run_lucro_liquido_number()
33          result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
34          numbers_from_result =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
35
36          self.assertEqual(len(result_pytesseract), 1, 'lucro
               líquido (número após conjunto de busca): tamanho
               resultado')
37          self.assertNotIn(data.LUCRO_LIQUIDO[self.filename],
               numbers_from_result,
38                           'lucro líquido (número após conjunto de
                                busca): valor')
39
40      def test_patrimonio_liquido_monetary(self):
41          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
42
43          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
44
45      def test_patrimonio_liquido_number(self):
46          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
47          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
48          number_from_result =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
49
50          self.assertEqual(len(result_pytesseract), 2,
51                           'patrimônio líquido (número após
                                conjunto de busca): tamanho
                                resultado')
52          self.assertNotIn(data.PATRIMONIO_LIQUIDO[self.filename],
               number_from_result,
53                           'patrimônio líquido (número após
                                conjunto de busca): valor')
54
55      def test_roe_monetary(self):
56          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
57
```

```
58          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
59
60      def test_roe_number(self):
61          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
62
63          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
64
65      def test_roe_calculate(self):
66          calculate_roe_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
67          result_pytesseract =
               calculate_roe_pytesseract[self.filename]
68
69          self.assertEqual(len(result_pytesseract), 10, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
70          self.assertNotIn(data.ROE[self.filename],
               result_pytesseract, 'ROE por cálculo: valor')
71
72
73  if __name__ == '__main__':
74      unittest.main()
```

repositorio/tests/reports/pytesseract02/py2_test_engie20192T.py

```
1   import unittest
2   from src import manager as m
3   from src.helper import result_helper as rh
4   from data import data as data
5
6
7   class TestsEngie20192T(unittest.TestCase):
8
9       filename = None
10      manager_pytesseract = None
11
12      @classmethod
13      def setUpClass(cls):
14          cls.filename = 'engie_2019_2T.pdf'
15          cls.manager_pytesseract = m.manager([cls.filename],
```

```
            'pytesseract', '--psm 6')
16
17      @classmethod
18      def tearDownClass(cls):
19          cls.filename = None
20          cls.manager_pytesseract = None
21
22      def test_lucro_liquido_monetary(self):
23          lucro_liquido_monetary_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_monetary()
24          result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
25          numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
26
27          self.assertEqual(len(result_pytesseract), 6, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
28          self.assertIn(data.LUCRO_LIQUIDO[self.filename],
29                        numbers_from_result_pytesseract,
30                        'lucro líquido (R$): valor (pytesseract)')
31
32      def test_lucro_liquido_number(self):
33          lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
34          result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
35          numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
36
37          self.assertEqual(len(result_pytesseract), 2, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
38          self.assertIn(data.LUCRO_LIQUIDO[self.filename],
39                        numbers_from_result_pytesseract,
40                        'lucro líquido (R$): valor (pytesseract)')
41
42      def test_patrimonio_liquido_monetary(self):
43          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
44
45          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
46
```

```
47      def test_patrimonio_liquido_number(self):
48          result =
               self.manager_pytesseract.run_patrimonio_liquido_number()[self.fi
49          numbers_from_result =
               rh.result_helper.get_numbers_as_list(result)
50
51          self.assertEqual(len(result), 1, 'patrimônio líquido
               (número após conjunto de busca): tamanho resultado')
52          self.assertIn(data.PATRIMONIO_LIQUIDO[self.filename],
               numbers_from_result,
53                        'patrimônio líquido (número após conjunto
                             de busca): valor')
54
55      def test_roe_monetary(self):
56          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
57
58          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
59
60      def test_roe_number(self):
61          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
62
63          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
64
65      def test_roe_calculate(self):
66          result =
               self.manager_pytesseract.run_calculate_roe()[self.filename]
67
68          self.assertEqual(len(result), 8, 'ROE por cálculo:
               tamanho resultado')
69          self.assertIn(data.ROE[self.filename], result, 'ROE por
               cálculo: valor')
70
71
72  if __name__ == '__main__':
73      unittest.main()
```

repositorio/tests/reports/pytesseract02/py2_test_engie20202T.py

```
1  import unittest
2  from src import manager as m
3  from src.helper import result_helper as rh
4  from data import data as data
5
6
7  class TestsEngie20202T(unittest.TestCase):
8
9      filename = None
10     manager_pytesseract = None
11
12     @classmethod
13     def setUpClass(cls):
14         cls.filename = 'engie_2020_2T.pdf'
15         cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 6')
16
17     @classmethod
18     def tearDownClass(cls):
19         cls.filename = None
20         cls.manager_pytesseract = None
21
22     def test_lucro_liquido_monetary(self):
23         lucro_liquido_monetary_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_monetary()
24         result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
25         numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
26
27         self.assertEqual(len(result_pytesseract), 6, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
28         self.assertIn(data.LUCRO_LIQUIDO[self.filename],
29                       numbers_from_result_pytesseract,
30                       'lucro líquido (R$): valor (pytesseract)')
31
32     def test_lucro_liquido_number(self):
33         lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
34         result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
35         numbers_from_result_pytesseract =
```

```
            rh.result_helper.get_numbers_as_list(result_pytesseract)
36
37          self.assertEqual(len(result_pytesseract), 2, 'lucro
               líquido (número após conjunto de busca): tamanho
               resultado')
38          self.assertIn(data.LUCRO_LIQUIDO[self.filename],
               numbers_from_result_pytesseract,
39                        'lucro líquido (número após conjunto de
                             busca): valor')
40
41      def test_patrimonio_liquido_monetary(self):
42          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self
43
44          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
45
46      def test_patrimonio_liquido_number(self):
47          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
48          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
49
50          self.assertEqual(len(result_pytesseract), 1,
51                           'patrimônio líquido (número após
                              conjunto de busca): tamanho resultado
                              (pytesseract)')
52          self.assertEqual(result_pytesseract[0]['number'],
               data.PATRIMONIO_LIQUIDO[self.filename],
53                           'patrimônio líquido (número após
                              conjunto de busca): valor
                              (pytesseract)')
54
55      def test_roe_monetary(self):
56          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
57
58          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
59
60      def test_roe_number(self):
61          result =
```

```
            self.manager_pytesseract.run_roe_number()[self.filename]
62
63          self.assertEqual(len(result), 0, 'ROE (número após
                conjunto de busca): tamanho resultado')
64
65      def test_roe_calculate(self):
66          roe_calculate_pytesseract =
                self.manager_pytesseract.run_calculate_roe()
67          result_pytesseract =
                roe_calculate_pytesseract[self.filename]
68
69          self.assertEqual(len(result_pytesseract), 8, 'ROE por
                cálculo: tamanho resultado (pytesseract)')
70          self.assertIn(data.ROE[self.filename],
                result_pytesseract, 'ROE por cálculo: valor
                (pytesseract)')
71
72
73  if __name__ == '__main__':
74      unittest.main()
```

repositorio/tests/reports/pytesseract02/py2_test_fleury20193T.py

```
1   import unittest
2   from src import manager as m
3   from src.helper import result_helper as rh
4   from data import data as data
5
6
7   class TestsFleury20193T(unittest.TestCase):
8
9       filename = None
10      manager_pytesseract = None
11
12      @classmethod
13      def setUpClass(cls):
14          cls.filename = 'fleury_2019_3T.pdf'
15          cls.manager_pytesseract = m.manager([cls.filename],
                'pytesseract', '--psm 6')
16
17      @classmethod
18      def tearDownClass(cls):
```

```
19          cls.filename = None
20          cls.manager_pytesseract = None
21
22      def test_lucro_liquido_monetary(self):
23          result =
               self.manager_pytesseract.run_lucro_liquido_monetary()[self.filen
24
25          self.assertEqual(len(result), 0, 'lucro líquido (R$):
               tamanho resultado')
26
27      def test_lucro_liquido_number(self):
28          lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
29          result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
30          numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
31
32          self.assertEqual(len(result_pytesseract), 1,
33                           'lucro líquido (número após conjunto de
                                busca): tamanho resultado
                                (pytesseract)')
34          self.assertIn(data.LUCRO_LIQUIDO[self.filename],
               numbers_from_result_pytesseract,
35                        'lucro líquido (número após conjunto de
                             busca): valor (pytesseract)')
36
37      def test_patrimonio_liquido_monetary(self):
38          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self
39
40          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
41
42      def test_patrimonio_liquido_number(self):
43          result =
               self.manager_pytesseract.run_patrimonio_liquido_number()[self.fi
44          number_from_result =
               rh.result_helper.get_numbers_as_list(result)
45
46          self.assertEqual(len(result), 3, 'patrimônio líquido
               (número após conjunto de busca): tamanho resultado')
```

```
47          self.assertIn(data.PATRIMONIO_LIQUIDO[self.filename],
                number_from_result,
48                        'patrimônio líquido (número após conjunto
                             de busca): valor')
49
50      def test_roe_monetary(self):
51          result =
                self.manager_pytesseract.run_roe_monetary()[self.filename]
52
53          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
                resultado')
54
55      def test_roe_number(self):
56          result =
                self.manager_pytesseract.run_roe_number()[self.filename]
57
58          self.assertEqual(len(result), 0, 'ROE (número após
                conjunto de busca): tamanho resultado')
59
60      def test_roe_calculate(self):
61          result =
                self.manager_pytesseract.run_calculate_roe()[self.filename]
62
63          self.assertEqual(len(result), 3, 'ROE por cálculo:
                tamanho resultado')
64          self.assertIn(data.ROE[self.filename], result, 'ROE por
                cálculo: valor')
65
66
67  if __name__ == '__main__':
68      unittest.main()
```

repositorio/tests/reports/pytesseract02/py2_test_fleury20202T.py

```
1  import unittest
2  from src import manager as m
3  from src.helper import result_helper as rh
4  from data import data as data
5
6
7  class TestsFleury20202T(unittest.TestCase):
8
```

```
9       filename = None
10      manager_pytesseract = None
11
12      @classmethod
13      def setUpClass(cls):
14          cls.filename = 'fleury_2020_2T.pdf'
15          cls.manager_pytesseract = m.manager([cls.filename],
                'pytesseract', '--psm 6')
16
17      @classmethod
18      def tearDownClass(cls):
19          cls.filename = None
20          cls.manager_pytesseract = None
21
22      def test_lucro_liquido_monetary(self):
23          lucro_liquido_monetary_pytesseract =
                self.manager_pytesseract.run_lucro_liquido_monetary()
24          result_pytesseract =
                lucro_liquido_monetary_pytesseract[self.filename]
25
26          self.assertEqual(len(result_pytesseract), 0, 'lucro
                líquido (R$): tamanho resultado (pytesseract)')
27
28      def test_lucro_liquido_number(self):
29          result =
                self.manager_pytesseract.run_lucro_liquido_number()[self.filenar
30
31          self.assertEqual(len(result), 0, 'lucro líquido (número
                após conjunto de busca): tamanho resultado')
32
33      def test_patrimonio_liquido_monetary(self):
34          result =
                self.manager_pytesseract.run_patrimonio_liquido_monetary()[self
35
36          self.assertEqual(len(result), 0, 'patrimônio líquido
                (R$): tamanho resultado')
37
38      def test_patrimonio_liquido_number(self):
39          patrimonio_liquido_number_pytesseract =
                self.manager_pytesseract.run_patrimonio_liquido_number()
40          result_pytesseract =
                patrimonio_liquido_number_pytesseract[self.filename]
```

```
41          numbers_from_result =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
42
43          self.assertEqual(len(result_pytesseract), 2,
44                           'patrimônio líquido (número após
                                conjunto de busca): tamanho resultado
                                (pytesseract)')
45          self.assertIn(data.PATRIMONIO_LIQUIDO[self.filename],
               numbers_from_result,
46                        'patrimônio líquido (número após conjunto
                             de busca): valor')
47
48      def test_roe_monetary(self):
49          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
50
51          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
52
53      def test_roe_number(self):
54          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
55
56          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
57
58      def test_roe_calculate(self):
59          roe_calculate_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
60          result_pytesseract =
               roe_calculate_pytesseract[self.filename]
61
62          self.assertEqual(len(result_pytesseract), 0, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
63
64
65  if __name__ == '__main__':
66      unittest.main()
```

repositorio/tests/reports/pytesseract02/py2_test_gerdau20153T.py

```
1  import unittest
```

```
2  from src import manager as m
3  from src.helper import result_helper as rh
4  from data import data as data
5
6
7  class TestsGerdau20153T(unittest.TestCase):
8
9      filename = None
10     manager_pytesseract = None
11
12     @classmethod
13     def setUpClass(cls):
14         cls.filename = 'gerdau_2015_3T.pdf'
15         cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 6')
16
17     @classmethod
18     def tearDownClass(cls):
19         cls.filename = None
20         cls.manager_pytesseract = None
21
22     def test_lucro_liquido_monetary(self):
23         result =
               self.manager_pytesseract.run_lucro_liquido_monetary()[self.file
24
25         self.assertEqual(len(result), 0, 'lucro líquido (R$):
               tamanho resultado')
26
27     def test_lucro_liquido_number(self):
28         result =
               self.manager_pytesseract.run_lucro_liquido_number()[self.filena
29
30         self.assertEqual(len(result), 0, 'lucro líquido (número
               após conjunto de busca): tamanho resultado')
31
32     def test_patrimonio_liquido_monetary(self):
33         result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self
34
35         self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
36
```

```
37      def test_patrimonio_liquido_number(self):
38          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
39          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
40          numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
41
42          self.assertEqual(len(result_pytesseract), 2,
43                           'patrimônio líquido (número após
                                conjunto de busca): tamanho resultado
                                (pytesseract)')
44          self.assertIn(data.PATRIMONIO_LIQUIDO[self.filename],
               numbers_from_result_pytesseract,
45                        'patrimônio líquido (número após conjunto
                             de busca): valor (pytesseract)')
46
47      def test_roe_monetary(self):
48          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
49
50          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
51
52      def test_roe_number(self):
53          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
54
55          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
56
57      def test_roe_calculate(self):
58          result =
               self.manager_pytesseract.run_calculate_roe()[self.filename]
59
60          self.assertEqual(len(result), 0, 'ROE por cálculo:
               tamanho resultado')
61
62
63  if __name__ == '__main__':
64      unittest.main()
```

repositorio/tests/reports/pytesseract02/py2_test_gerdau20171T.py

```
1  import unittest
2  from src import manager as m
3  from src.helper import result_helper as rh
4  from data import data as data
5
6
7  class TestsGerdau20171T(unittest.TestCase):
8
9      filename = None
10     manager_pytesseract = None
11
12     @classmethod
13     def setUpClass(cls):
14         cls.filename = 'gerdau_2017_1T.pdf'
15         cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 6')
16
17     @classmethod
18     def tearDownClass(cls):
19         cls.filename = None
20         cls.manager_pytesseract = None
21
22     def test_lucro_liquido_monetary(self):
23         result =
               self.manager_pytesseract.run_lucro_liquido_monetary()[self.filer
24
25         self.assertEqual(len(result), 0, 'lucro líquido (R$):
               tamanho resultado')
26
27     def test_lucro_liquido_number(self):
28         lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
29         result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
30
31         self.assertEqual(len(result_pytesseract), 0,
32                          'lucro líquido (número após conjunto de
                                busca): tamanho resultado
                                (pytesseract)')
33
34     def test_patrimonio_liquido_monetary(self):
```

```
35          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
36
37          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
38
39      def test_patrimonio_liquido_number(self):
40          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
41          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
42          numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
43
44          self.assertEqual(len(result_pytesseract), 2,
45                           'patrimônio líquido (número após
                              conjunto de busca): tamanho resultado
                              (pytesseract)')
46          self.assertIn(data.PATRIMONIO_LIQUIDO[self.filename],
               numbers_from_result_pytesseract,
47                        'patrimônio líquido (número após conjunto
                           de busca): valor (pytesseract)')
48
49      def test_roe_monetary(self):
50          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
51
52          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
53
54      def test_roe_number(self):
55          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
56
57          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
58
59      def test_roe_calculate(self):
60          calculate_roe_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
61          result_pytesseract =
               calculate_roe_pytesseract[self.filename]
```

```
62
63          self.assertEqual(len(result_pytesseract), 0, 'ROE por
                cálculo: tamanho resultado (pytesseract)')
64
65
66  if __name__ == '__main__':
67      unittest.main()
```

repositorio/tests/reports/pytesseract02/py2_test_gerdau20184T.py

```
1   import unittest
2   from src import manager as m
3   from src.helper import result_helper as rh
4   from data import data as data
5
6
7   class TestsGerdau20184T(unittest.TestCase):
8
9       filename = None
10      manager_pytesseract = None
11
12      @classmethod
13      def setUpClass(cls):
14          cls.filename = 'gerdau_2018_4T.pdf'
15          cls.manager_pytesseract = m.manager([cls.filename],
                'pytesseract', '--psm 6')
16
17      @classmethod
18      def tearDownClass(cls):
19          cls.filename = None
20          cls.manager_pytesseract = None
21
22      def test_lucro_liquido_monetary(self):
23          result =
                self.manager_pytesseract.run_lucro_liquido_monetary()[self.filer
24
25          self.assertEqual(len(result), 0, 'lucro líquido (R$):
                tamanho resultado')
26
27      def test_lucro_liquido_number(self):
28          lucro_liquido_number_pytesseract =
                self.manager_pytesseract.run_lucro_liquido_number()
```

```
29          result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
30
31          self.assertEqual(len(result_pytesseract), 0,
32                           'lucro líquido (número após conjunto de
                                busca): tamanho resultado
                                (pytesseract)')
33
34      def test_patrimonio_liquido_monetary(self):
35          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
36
37          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
38
39      def test_patrimonio_liquido_number(self):
40          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
41          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
42          numbers_from_result =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
43
44          self.assertEqual(len(result_pytesseract), 2,
45                           'patrimônio líquido (número após
                                conjunto de busca): tamanho resultado
                                (pytesseract)')
46          self.assertIn(data.PATRIMONIO_LIQUIDO[self.filename],
               numbers_from_result,
47                        'patrimônio líquido (número após conjunto
                             de busca): valor')
48
49      def test_roe_monetary(self):
50          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
51
52          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
53
54      def test_roe_number(self):
55          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
```

```
56
57          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
58
59      def test_roe_calculate(self):
60          calculate_roe_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
61          result_pytesseract =
               calculate_roe_pytesseract[self.filename]
62
63          self.assertEqual(len(result_pytesseract), 0, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
64
65
66  if __name__ == '__main__':
67      unittest.main()
```

repositorio/tests/reports/pytesseract02/py2_test_gerdau20192T.py

```
1   import unittest
2   from src import manager as m
3   from src.helper import result_helper as rh
4   from data import data as data
5
6
7   class TestsGerdau20192T(unittest.TestCase):
8
9       filename = None
10      manager_pytesseract = None
11
12      @classmethod
13      def setUpClass(cls):
14          cls.filename = 'gerdau_2019_2T.pdf'
15          cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 6')
16
17      @classmethod
18      def tearDownClass(cls):
19          cls.filename = None
20          cls.manager_pytesseract = None
21
22      def test_lucro_liquido_monetary(self):
```

```
23          result =
               self.manager_pytesseract.run_lucro_liquido_monetary()[self.filename]
24
25          self.assertEqual(len(result), 0, 'lucro líquido (R$):
               tamanho resultado')
26
27      def test_lucro_liquido_number(self):
28          lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
29          result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
30          numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
31
32          self.assertEqual(len(result_pytesseract), 1,
33                           'lucro líquido (número após conjunto de
                                busca): tamanho resultado
                                (pytesseract)')
34          self.assertNotIn(data.LUCRO_LIQUIDO[self.filename],
               numbers_from_result_pytesseract,
35                           'lucro líquido (número após conjunto de
                                busca): valor (pytesseract)')
36
37      def test_patrimonio_liquido_monetary(self):
38          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
39
40          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
41
42      def test_patrimonio_liquido_number(self):
43          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
44          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
45          numbers_from_result =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
46
47          self.assertEqual(len(result_pytesseract), 2,
48                           'patrimônio líquido (número após
                                conjunto de busca): tamanho resultado
                                (pytesseract)')
```

```
49          self.assertIn(data.PATRIMONIO_LIQUIDO[self.filename],
               numbers_from_result,
50                        'patrimônio líquido (número após conjunto
                             de busca): valor')
51
52      def test_roe_monetary(self):
53          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
54
55          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
56
57      def test_roe_number(self):
58          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
59
60          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
61
62      def test_roe_calculate(self):
63          calculate_roe_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
64          result_pytesseract =
               calculate_roe_pytesseract[self.filename]
65
66          self.assertEqual(len(result_pytesseract), 2, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
67          self.assertNotIn(data.ROE[self.filename],
               result_pytesseract, 'ROE por cálculo: valor')
68
69
70  if __name__ == '__main__':
71      unittest.main()
```

repositorio/tests/reports/pytesseract02/py2_test_weg20102T.py

```
1  import unittest
2  from src import manager as m
3  from src.helper import result_helper as rh
4  from data import data as data
5
6
```

```
7  class TestsWeg20102T(unittest.TestCase):
8
9      filename = None
10     manager_pytesseract = None
11
12     @classmethod
13     def setUpClass(cls):
14         cls.filename = 'weg_2010_2T.pdf'
15         cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 6')
16
17     @classmethod
18     def tearDownClass(cls):
19         cls.filename = None
20         cls.manager_pytesseract = None
21
22     def test_lucro_liquido_monetary(self):
23         lucro_liquido_monetary_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_monetary()
24         result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
25         numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
26
27         self.assertEqual(len(result_pytesseract), 1, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
28         self.assertIn(data.LUCRO_LIQUIDO[self.filename],
               numbers_from_result_pytesseract,
29                       'lucro líquido (R$): valor (pytesseract)')
30
31     def test_lucro_liquido_number(self):
32         result =
               self.manager_pytesseract.run_lucro_liquido_number()[self.filename]
33
34         self.assertEqual(len(result), 0, 'lucro líquido (número
               após conjunto de busca): tamanho resultado')
35
36     def test_patrimonio_liquido_monetary(self):
37         result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
38
39         self.assertEqual(len(result), 0, 'patrimônio líquido
```

```
            (R$): tamanho resultado')
40
41      def test_patrimonio_liquido_number(self):
42          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
43          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
44
45          self.assertEqual(len(result_pytesseract), 0,
46                           'patrimônio líquido (número após
                                conjunto de busca): tamanho resultado
                                (pytesseract)')
47
48      def test_roe_monetary(self):
49          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
50
51          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
52
53      def test_roe_number(self):
54          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
55
56          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
57
58      def test_roe_calculate(self):
59          roe_calculate_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
60          result_pytesseract =
               roe_calculate_pytesseract[self.filename]
61
62          self.assertEqual(len(result_pytesseract), 0, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
63
64
65  if __name__ == '__main__':
66      unittest.main()
```

repositorio/tests/reports/pytesseract02/py2_test_weg20151T.py

```
import unittest
from src import manager as m
from src.helper import result_helper as rh
from data import data as data


class TestsWeg20151T(unittest.TestCase):

    filename = None
    manager_pytesseract = None

    @classmethod
    def setUpClass(cls):
        cls.filename = 'weg_2015_1T.pdf'
        cls.manager_pytesseract = m.manager([cls.filename],
           'pytesseract', '--psm 6')

    @classmethod
    def tearDownClass(cls):
        cls.filename = None
        cls.manager_pytesseract = None

    def test_lucro_liquido_monetary(self):
        lucro_liquido_monetary_pytesseract =
           self.manager_pytesseract.run_lucro_liquido_monetary()
        result_pytesseract =
           lucro_liquido_monetary_pytesseract[self.filename]
        numbers_from_result_pytesseract =
           rh.result_helper.get_numbers_as_list(result_pytesseract)

        self.assertEqual(len(result_pytesseract), 1, 'lucro
           líquido (R$): tamanho resultado (pytesseract)')
        self.assertIn(data.LUCRO_LIQUIDO[self.filename],
           numbers_from_result_pytesseract,
                      'lucro líquido (R$): valor (pytesseract)')

    def test_lucro_liquido_number(self):
        result =
           self.manager_pytesseract.run_lucro_liquido_number()[self.filename]

        self.assertEqual(len(result), 0, 'lucro líquido (número
           após conjunto de busca): tamanho resultado')
```

```
35
36      def test_patrimonio_liquido_monetary(self):
37          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self
38
39          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
40
41      def test_patrimonio_liquido_number(self):
42          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
43          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
44          numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
45
46          self.assertEqual(len(result_pytesseract), 1,
47                           'patrimônio líquido (número após
                              conjunto de busca): tamanho resultado
                              (pytesseract)')
48          self.assertNotIn(data.PATRIMONIO_LIQUIDO[self.filename],
               numbers_from_result_pytesseract,
49                           'patrimônio líquido (número após
                              conjunto de busca): valor
                              (pytesseract)')
50
51      def test_roe_monetary(self):
52          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
53
54          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
55
56      def test_roe_number(self):
57          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
58
59          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
60
61      def test_roe_calculate(self):
62          calculate_roe_pytesseract =
```

```
            self.manager_pytesseract.run_calculate_roe()
63          result_pytesseract =
               calculate_roe_pytesseract[self.filename]
64
65          self.assertEqual(len(result_pytesseract), 1, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
66          self.assertNotIn(data.ROE[self.filename],
               result_pytesseract,
67                           'ROE por cálculo: tamanho resultado
                                (pytesseract)')
68
69
70  if __name__ == '__main__':
71      unittest.main()
```

repositorio/tests/reports/pytesseract02/py2_test_weg20172T.py

```
1   import unittest
2   from src import manager as m
3   from src.helper import result_helper as rh
4   from data import data as data
5
6
7   class TestsWeg20172T(unittest.TestCase):
8
9       filename = None
10      manager_pytesseract = None
11
12      @classmethod
13      def setUpClass(cls):
14          cls.filename = 'weg_2017_2T.pdf'
15          cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 6')
16
17      @classmethod
18      def tearDownClass(cls):
19          cls.filename = None
20          cls.manager_pytesseract = None
21
22      def test_lucro_liquido_monetary(self):
23          lucro_liquido_monetary_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_monetary()
```

```
24          result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
25          numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
26
27          self.assertEqual(len(result_pytesseract), 1, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
28          self.assertIn(data.LUCRO_LIQUIDO[self.filename],
               numbers_from_result_pytesseract,
29                        'lucro líquido (R$): valor (pytesseract)')
30
31      def test_lucro_liquido_number(self):
32          lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
33          result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
34          numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
35
36          self.assertEqual(len(result_pytesseract), 1,
37                           'lucro líquido (número após conjunto de
                              busca): tamanho resultado
                              (pytesseract)')
38          self.assertNotIn(data.LUCRO_LIQUIDO[self.filename],
               numbers_from_result_pytesseract,
39                           'lucro líquido (número após conjunto de
                              busca): valor (pytesseract)')
40
41      def test_patrimonio_liquido_monetary(self):
42          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self
43
44          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
45
46      def test_patrimonio_liquido_number(self):
47          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
48          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
49          numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
```

```
50
51          self.assertEqual(len(result_pytesseract), 1,
52                           'patrimônio líquido (número após
                                conjunto de busca): tamanho resultado
                                (pytesseract)')
53          self.assertNotIn(data.PATRIMONIO_LIQUIDO[self.filename],
                numbers_from_result_pytesseract,
54                           'patrimônio líquido (número após
                                conjunto de busca): valor
                                (pytesseract)')
55
56      def test_roe_monetary(self):
57          result =
                self.manager_pytesseract.run_roe_monetary()[self.filename]
58
59          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
                resultado')
60
61      def test_roe_number(self):
62          result =
                self.manager_pytesseract.run_roe_number()[self.filename]
63
64          self.assertEqual(len(result), 0, 'ROE (número após
                conjunto de busca): tamanho resultado')
65
66      def test_roe_calculate(self):
67          roe_calculate_pytesseract =
                self.manager_pytesseract.run_calculate_roe()
68          result_pytesseract =
                roe_calculate_pytesseract[self.filename]
69
70          self.assertEqual(len(result_pytesseract), 2, 'ROE por
                cálculo: tamanho resultado (pytesseract)')
71          self.assertNotIn(data.ROE[self.filename],
                result_pytesseract, 'ROE por cálculo: valor')
72
73
74  if __name__ == '__main__':
75      unittest.main()
```

repositorio/tests/reports/pytesseract02/py2_test_weg20192T.py

```
1  import unittest
2  from src import manager as m
3  from src.helper import result_helper as rh
4  from data import data as data
5
6
7  class TestsWeg20192T(unittest.TestCase):
8
9      filename = None
10     manager_pytesseract = None
11
12     @classmethod
13     def setUpClass(cls):
14         cls.filename = 'weg_2019_2T.pdf'
15         cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 6')
16
17     @classmethod
18     def tearDownClass(cls):
19         cls.filename = None
20         cls.manager_pytesseract = None
21
22     def test_lucro_liquido_monetary(self):
23         lucro_liquido_monetary_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_monetary()
24         result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
25         numbers_from_result_pytesseract =
               rh.result_helper.get_numbers_as_list(result_pytesseract)
26
27         self.assertEqual(len(result_pytesseract), 1, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
28         self.assertIn(data.LUCRO_LIQUIDO[self.filename],
               numbers_from_result_pytesseract,
29                       'lucro líquido (R$): valor (pytesseract)')
30
31     def test_lucro_liquido_number(self):
32         lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
33         result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
34
```

```
35          self.assertEqual(len(result_pytesseract), 0,
36                           'lucro líquido (número após conjunto de
                                busca): tamanho resultado
                                (pytesseract)')
37
38      def test_patrimonio_liquido_monetary(self):
39          result =
                self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
40
41          self.assertEqual(len(result), 0, 'patrimônio líquido
                (R$): tamanho resultado')
42
43      def test_patrimonio_liquido_number(self):
44          patrimonio_liquido_number_pytesseract =
                self.manager_pytesseract.run_patrimonio_liquido_number()
45          result_pytesseract =
                patrimonio_liquido_number_pytesseract[self.filename]
46          numbers_from_result_pytesseract =
                rh.result_helper.get_numbers_as_list(result_pytesseract)
47
48          self.assertEqual(len(result_pytesseract), 1,
49                           'patrimônio líquido (número após
                                conjunto de busca): tamanho resultado
                                (pytesseract)')
50          self.assertNotIn(data.PATRIMONIO_LIQUIDO[self.filename],
                numbers_from_result_pytesseract,
51                           'patrimônio líquido (número após
                                conjunto de busca): valor
                                (pytesseract)')
52
53      def test_roe_monetary(self):
54          result =
                self.manager_pytesseract.run_roe_monetary()[self.filename]
55
56          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
                resultado')
57
58      def test_roe_number(self):
59          result =
                self.manager_pytesseract.run_roe_number()[self.filename]
60
61          self.assertEqual(len(result), 0, 'ROE (número após
```

```
            conjunto de busca): tamanho resultado')

    def test_roe_calculate(self):
        roe_calculate_pytesseract =
            self.manager_pytesseract.run_calculate_roe()
        result_pytesseract =
            roe_calculate_pytesseract[self.filename]

        self.assertEqual(len(result_pytesseract), 1, 'ROE por
            cálculo: tamanho resultado (pytesseract)')
        self.assertNotIn(data.ROE[self.filename],
            result_pytesseract, 'ROE por cálculo: valor')


if __name__ == '__main__':
    unittest.main()
```

repositorio/tests/reports/pytesseract03/py3_test_ambev20191T.py

```
import unittest
from src import manager as m


class TestsAmbev20191T(unittest.TestCase):

    filename = None
    manager_pytesseract = None

    @classmethod
    def setUpClass(cls):
        cls.filename = 'ambev_2019_1T.pdf'
        cls.manager_pytesseract = m.manager([cls.filename],
            'pytesseract', '--psm 5')

    @classmethod
    def tearDownClass(cls):
        cls.filename = None
        cls.manager_pytesseract = None

    def test_lucro_liquido_monetary(self):
        lucro_liquido_monetary_pytesseract =
            self.manager_pytesseract.run_lucro_liquido_monetary()
```

```
22          result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
23
24          self.assertEqual(len(result_pytesseract), 0, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
25
26      def test_lucro_liquido_number(self):
27          lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
28          result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
29
30          self.assertEqual(len(result_pytesseract), 0, 'lucro
               líquido (número após conjunto de busca): tamanho
               resultado')
31
32      def test_patrimonio_liquido_monetary(self):
33          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
34
35          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
36
37      def test_patrimonio_liquido_number(self):
38          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
39          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
40
41          self.assertEqual(len(result_pytesseract), 0,
42                           'patrimônio líquido (número após
                              conjunto de busca): tamanho
                              resultado')
43
44      def test_roe_monetary(self):
45          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
46
47          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
48
49      def test_roe_number(self):
```

```
50          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
51
52          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
53
54      def test_roe_calculate(self):
55          calculate_roe_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
56          result_pytesseract =
               calculate_roe_pytesseract[self.filename]
57
58          self.assertEqual(len(result_pytesseract), 0, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
59
60
61  if __name__ == '__main__':
62      unittest.main()
```

repositorio/tests/reports/pytesseract03/py3_test_engie20192T.py

```
1   import unittest
2   from src import manager as m
3
4
5   class TestsEngie20192T(unittest.TestCase):
6
7       filename = None
8       manager_pytesseract = None
9
10      @classmethod
11      def setUpClass(cls):
12          cls.filename = 'engie_2019_2T.pdf'
13          cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 5')
14
15      @classmethod
16      def tearDownClass(cls):
17          cls.filename = None
18          cls.manager_pytesseract = None
19
20      def test_lucro_liquido_monetary(self):
```

```
21          lucro_liquido_monetary_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_monetary()
22          result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
23
24          self.assertEqual(len(result_pytesseract), 0, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
25
26      def test_lucro_liquido_number(self):
27          lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
28          result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
29
30          self.assertEqual(len(result_pytesseract), 0, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
31
32      def test_patrimonio_liquido_monetary(self):
33          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
34
35          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
36
37      def test_patrimonio_liquido_number(self):
38          result =
               self.manager_pytesseract.run_patrimonio_liquido_number()[self.filena
39
40          self.assertEqual(len(result), 0, 'patrimônio líquido
               (número após conjunto de busca): tamanho resultado')
41
42      def test_roe_monetary(self):
43          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
44
45          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
46
47      def test_roe_number(self):
48          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
49
```

```
50          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
51
52      def test_roe_calculate(self):
53          result =
               self.manager_pytesseract.run_calculate_roe()[self.filename]
54
55          self.assertEqual(len(result), 0, 'ROE por cálculo:
               tamanho resultado')
56
57
58  if __name__ == '__main__':
59      unittest.main()
```

repositorio/tests/reports/pytesseract03/py3_test_engie20202T.py

```
1   import unittest
2   from src import manager as m
3
4
5   class TestsEngie20202T(unittest.TestCase):
6
7       filename = None
8       manager_pytesseract = None
9
10      @classmethod
11      def setUpClass(cls):
12          cls.filename = 'engie_2020_2T.pdf'
13          cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 5')
14
15      @classmethod
16      def tearDownClass(cls):
17          cls.filename = None
18          cls.manager_pytesseract = None
19
20      def test_lucro_liquido_monetary(self):
21          lucro_liquido_monetary_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_monetary()
22          result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
23
```

```
24          self.assertEqual(len(result_pytesseract), 0, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
25
26      def test_lucro_liquido_number(self):
27          lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
28          result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
29
30          self.assertEqual(len(result_pytesseract), 0, 'lucro
               líquido (número após conjunto de busca): tamanho
               resultado')
31
32      def test_patrimonio_liquido_monetary(self):
33          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
34
35          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
36
37      def test_patrimonio_liquido_number(self):
38          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
39          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
40
41          self.assertEqual(len(result_pytesseract), 0,
42                           'patrimônio líquido (número após
                              conjunto de busca): tamanho resultado
                              (pytesseract)')
43
44      def test_roe_monetary(self):
45          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
46
47          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
48
49      def test_roe_number(self):
50          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
51
```

```
52          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
53
54      def test_roe_calculate(self):
55          roe_calculate_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
56          result_pytesseract =
               roe_calculate_pytesseract[self.filename]
57
58          self.assertEqual(len(result_pytesseract), 0, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
59
60
61  if __name__ == '__main__':
62      unittest.main()
```

repositorio/tests/reports/pytesseract03/py3_test_fleury20193T.py

```
1   import unittest
2   from src import manager as m
3
4
5   class TestsFleury20193T(unittest.TestCase):
6
7       filename = None
8       manager_pytesseract = None
9
10      @classmethod
11      def setUpClass(cls):
12          cls.filename = 'fleury_2019_3T.pdf'
13          cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 5')
14
15      @classmethod
16      def tearDownClass(cls):
17          cls.filename = None
18          cls.manager_pytesseract = None
19
20      def test_lucro_liquido_monetary(self):
21          result =
               self.manager_pytesseract.run_lucro_liquido_monetary()[self.filen
22
```

```
23          self.assertEqual(len(result), 0, 'lucro líquido (R$):
               tamanho resultado')
24
25      def test_lucro_liquido_number(self):
26          lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
27          result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
28
29          self.assertEqual(len(result_pytesseract), 0,
30                           'lucro líquido (número após conjunto de
                                busca): tamanho resultado
                                (pytesseract)')
31
32      def test_patrimonio_liquido_monetary(self):
33          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
34
35          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
36
37      def test_patrimonio_liquido_number(self):
38          result =
               self.manager_pytesseract.run_patrimonio_liquido_number()[self.filena
39
40          self.assertEqual(len(result), 0, 'patrimônio líquido
               (número após conjunto de busca): tamanho resultado')
41
42      def test_roe_monetary(self):
43          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
44
45          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
46
47      def test_roe_number(self):
48          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
49
50          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
51
```

```
52      def test_roe_calculate(self):
53          result =
               self.manager_pytesseract.run_calculate_roe()[self.filename]
54
55          self.assertEqual(len(result), 0, 'ROE por cálculo:
               tamanho resultado')
56
57
58  if __name__ == '__main__':
59      unittest.main()
```

repositorio/tests/reports/pytesseract03/py3_test_fleury20202T.py

```
1   import unittest
2   from src import manager as m
3
4
5   class TestsFleury20202T(unittest.TestCase):
6
7       filename = None
8       manager_pytesseract = None
9
10      @classmethod
11      def setUpClass(cls):
12          cls.filename = 'fleury_2020_2T.pdf'
13          cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 5')
14
15      @classmethod
16      def tearDownClass(cls):
17          cls.filename = None
18          cls.manager_pytesseract = None
19
20      def test_lucro_liquido_monetary(self):
21          lucro_liquido_monetary_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_monetary()
22          result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
23
24          self.assertEqual(len(result_pytesseract), 0, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
25
```

```
26      def test_lucro_liquido_number(self):
27          result =
               self.manager_pytesseract.run_lucro_liquido_number()[self.filename]
28
29          self.assertEqual(len(result), 0, 'lucro líquido (número
               após conjunto de busca): tamanho resultado')
30
31      def test_patrimonio_liquido_monetary(self):
32          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
33
34          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
35
36      def test_patrimonio_liquido_number(self):
37          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
38          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
39
40          self.assertEqual(len(result_pytesseract), 0,
41                           'patrimônio líquido (número após
                              conjunto de busca): tamanho resultado
                              (pytesseract)')
42
43      def test_roe_monetary(self):
44          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
45
46          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
47
48      def test_roe_number(self):
49          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
50
51          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
52
53      def test_roe_calculate(self):
54          roe_calculate_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
```

```
55          result_pytesseract =
               roe_calculate_pytesseract[self.filename]
56
57          self.assertEqual(len(result_pytesseract), 0, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
58
59
60  if __name__ == '__main__':
61      unittest.main()
```

repositorio/tests/reports/pytesseract03/py3_test_gerdau20153T.py

```
1   import unittest
2   from src import manager as m
3
4
5   class TestsGerdau20153T(unittest.TestCase):
6
7       filename = None
8       manager_pytesseract = None
9
10      @classmethod
11      def setUpClass(cls):
12          cls.filename = 'gerdau_2015_3T.pdf'
13          cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 5')
14
15      @classmethod
16      def tearDownClass(cls):
17          cls.filename = None
18          cls.manager_pytesseract = None
19
20      def test_lucro_liquido_monetary(self):
21          result =
               self.manager_pytesseract.run_lucro_liquido_monetary()[self.filer
22
23          self.assertEqual(len(result), 0, 'lucro líquido (R$):
               tamanho resultado')
24
25      def test_lucro_liquido_number(self):
26          result =
               self.manager_pytesseract.run_lucro_liquido_number()[self.filenar
```

```
27
28          self.assertEqual(len(result), 0, 'lucro líquido (número
               após conjunto de busca): tamanho resultado')
29
30      def test_patrimonio_liquido_monetary(self):
31          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
32
33          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
34
35      def test_patrimonio_liquido_number(self):
36          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
37          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
38
39          self.assertEqual(len(result_pytesseract), 0,
40                           'patrimônio líquido (número após
                              conjunto de busca): tamanho resultado
                              (pytesseract)')
41
42      def test_roe_monetary(self):
43          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
44
45          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
46
47      def test_roe_number(self):
48          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
49
50          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
51
52      def test_roe_calculate(self):
53          result =
               self.manager_pytesseract.run_calculate_roe()[self.filename]
54
55          self.assertEqual(len(result), 0, 'ROE por cálculo:
               tamanho resultado')
```

```
56
57
58 if __name__ == '__main__':
59     unittest.main()
```

repositorio/tests/reports/pytesseract03/py3_test_gerdau20171T.py

```
1  import unittest
2  from src import manager as m
3
4
5  class TestsGerdau20171T(unittest.TestCase):
6
7      filename = None
8      manager_pytesseract = None
9
10     @classmethod
11     def setUpClass(cls):
12         cls.filename = 'gerdau_2017_1T.pdf'
13         cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 5')
14
15     @classmethod
16     def tearDownClass(cls):
17         cls.filename = None
18         cls.manager_pytesseract = None
19
20     def test_lucro_liquido_monetary(self):
21         result =
               self.manager_pytesseract.run_lucro_liquido_monetary()[self.file
22
23         self.assertEqual(len(result), 0, 'lucro líquido (R$):
               tamanho resultado')
24
25     def test_lucro_liquido_number(self):
26         lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
27         result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
28
29         self.assertEqual(len(result_pytesseract), 0,
30                          'lucro líquido (número após conjunto de
```

```
                            busca): tamanho resultado
                            (pytesseract)')
31
32      def test_patrimonio_liquido_monetary(self):
33          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
34
35          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
36
37      def test_patrimonio_liquido_number(self):
38          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
39          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
40
41          self.assertEqual(len(result_pytesseract), 0,
42                           'patrimônio líquido (número após
                              conjunto de busca): tamanho resultado
                              (pytesseract)')
43
44      def test_roe_monetary(self):
45          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
46
47          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
48
49      def test_roe_number(self):
50          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
51
52          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
53
54      def test_roe_calculate(self):
55          calculate_roe_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
56          result_pytesseract =
               calculate_roe_pytesseract[self.filename]
57
58          self.assertEqual(len(result_pytesseract), 0, 'ROE por
```

```
            cálculo: tamanho resultado (pytesseract)')


if __name__ == '__main__':
    unittest.main()
```

repositorio/tests/reports/pytesseract03/py3_test_gerdau20184T.py

```
import unittest
from src import manager as m


class TestsGerdau20184T(unittest.TestCase):

    filename = None
    manager_pytesseract = None

    @classmethod
    def setUpClass(cls):
        cls.filename = 'gerdau_2018_4T.pdf'
        cls.manager_pytesseract = m.manager([cls.filename],
           'pytesseract', '--psm 5')

    @classmethod
    def tearDownClass(cls):
        cls.filename = None
        cls.manager_pytesseract = None

    def test_lucro_liquido_monetary(self):
        result =
           self.manager_pytesseract.run_lucro_liquido_monetary()[self.filer

        self.assertEqual(len(result), 0, 'lucro líquido (R$):
           tamanho resultado')

    def test_lucro_liquido_number(self):
        lucro_liquido_number_pytesseract =
           self.manager_pytesseract.run_lucro_liquido_number()
        result_pytesseract =
           lucro_liquido_number_pytesseract[self.filename]

        self.assertEqual(len(result_pytesseract), 0,
```

```
30                          'lucro líquido (número após conjunto de
                               busca): tamanho resultado
                               (pytesseract)')
31
32      def test_patrimonio_liquido_monetary(self):
33          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
34
35          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
36
37      def test_patrimonio_liquido_number(self):
38          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
39          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
40
41          self.assertEqual(len(result_pytesseract), 0,
42                           'patrimônio líquido (número após
                               conjunto de busca): tamanho resultado
                               (pytesseract)')
43
44      def test_roe_monetary(self):
45          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
46
47          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
48
49      def test_roe_number(self):
50          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
51
52          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
53
54      def test_roe_calculate(self):
55          calculate_roe_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
56          result_pytesseract =
               calculate_roe_pytesseract[self.filename]
57
```

```
58          self.assertEqual(len(result_pytesseract), 0, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
59
60
61  if __name__ == '__main__':
62      unittest.main()
```

repositorio/tests/reports/pytesseract03/py3_test_gerdau20192T.py

```
1   import unittest
2   from src import manager as m
3
4
5   class TestsGerdau20192T(unittest.TestCase):
6
7       filename = None
8       manager_pytesseract = None
9
10      @classmethod
11      def setUpClass(cls):
12          cls.filename = 'gerdau_2019_2T.pdf'
13          cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 5')
14
15      @classmethod
16      def tearDownClass(cls):
17          cls.filename = None
18          cls.manager_pytesseract = None
19
20      def test_lucro_liquido_monetary(self):
21          result =
               self.manager_pytesseract.run_lucro_liquido_monetary()[self.filen
22
23          self.assertEqual(len(result), 0, 'lucro líquido (R$):
               tamanho resultado')
24
25      def test_lucro_liquido_number(self):
26          lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
27          result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
28
```

```
29          self.assertEqual(len(result_pytesseract), 0,
30                           'lucro líquido (número após conjunto de
                                busca): tamanho resultado
                                (pytesseract)')
31
32      def test_patrimonio_liquido_monetary(self):
33          result =
                self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
34
35          self.assertEqual(len(result), 0, 'patrimônio líquido
                (R$): tamanho resultado')
36
37      def test_patrimonio_liquido_number(self):
38          patrimonio_liquido_number_pytesseract =
                self.manager_pytesseract.run_patrimonio_liquido_number()
39          result_pytesseract =
                patrimonio_liquido_number_pytesseract[self.filename]
40
41          self.assertEqual(len(result_pytesseract), 0,
42                           'patrimônio líquido (número após
                                conjunto de busca): tamanho resultado
                                (pytesseract)')
43
44      def test_roe_monetary(self):
45          result =
                self.manager_pytesseract.run_roe_monetary()[self.filename]
46
47          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
                resultado')
48
49      def test_roe_number(self):
50          result =
                self.manager_pytesseract.run_roe_number()[self.filename]
51
52          self.assertEqual(len(result), 0, 'ROE (número após
                conjunto de busca): tamanho resultado')
53
54      def test_roe_calculate(self):
55          calculate_roe_pytesseract =
                self.manager_pytesseract.run_calculate_roe()
56          result_pytesseract =
                calculate_roe_pytesseract[self.filename]
```

```
57
58          self.assertEqual(len(result_pytesseract), 0, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
59
60
61  if __name__ == '__main__':
62      unittest.main()
```

repositorio/tests/reports/pytesseract03/py3_test_weg20102T.py

```
1   import unittest
2   from src import manager as m
3
4
5   class TestsWeg20102T(unittest.TestCase):
6
7       filename = None
8       manager_pytesseract = None
9
10      @classmethod
11      def setUpClass(cls):
12          cls.filename = 'weg_2010_2T.pdf'
13          cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 5')
14
15      @classmethod
16      def tearDownClass(cls):
17          cls.filename = None
18          cls.manager_pytesseract = None
19
20      def test_lucro_liquido_monetary(self):
21          lucro_liquido_monetary_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_monetary()
22          result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
23
24          self.assertEqual(len(result_pytesseract), 0, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
25
26      def test_lucro_liquido_number(self):
27          result =
               self.manager_pytesseract.run_lucro_liquido_number()[self.filenar
```

```
28
29          self.assertEqual(len(result), 0, 'lucro líquido (número
               após conjunto de busca): tamanho resultado')
30
31      def test_patrimonio_liquido_monetary(self):
32          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
33
34          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
35
36      def test_patrimonio_liquido_number(self):
37          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
38          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
39
40          self.assertEqual(len(result_pytesseract), 0,
41                           'patrimônio líquido (número após
                              conjunto de busca): tamanho resultado
                              (pytesseract)')
42
43      def test_roe_monetary(self):
44          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
45
46          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
47
48      def test_roe_number(self):
49          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
50
51          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
52
53      def test_roe_calculate(self):
54          roe_calculate_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
55          result_pytesseract =
               roe_calculate_pytesseract[self.filename]
56
```

```
57          self.assertEqual(len(result_pytesseract), 0, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
58
59
60  if __name__ == '__main__':
61      unittest.main()
```

repositorio/tests/reports/pytesseract03/py3_test_weg20151T.py

```
1   import unittest
2   from src import manager as m
3
4
5   class TestsWeg20151T(unittest.TestCase):
6
7       filename = None
8       manager_pytesseract = None
9
10      @classmethod
11      def setUpClass(cls):
12          cls.filename = 'weg_2015_1T.pdf'
13          cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 5')
14
15      @classmethod
16      def tearDownClass(cls):
17          cls.filename = None
18          cls.manager_pytesseract = None
19
20      def test_lucro_liquido_monetary(self):
21          lucro_liquido_monetary_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_monetary()
22          result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
23
24          self.assertEqual(len(result_pytesseract), 0, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
25
26      def test_lucro_liquido_number(self):
27          result =
               self.manager_pytesseract.run_lucro_liquido_number()[self.filenar
28
```

```
29          self.assertEqual(len(result), 0, 'lucro líquido (número
               após conjunto de busca): tamanho resultado')
30
31      def test_patrimonio_liquido_monetary(self):
32          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
33
34          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
35
36      def test_patrimonio_liquido_number(self):
37          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
38          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
39
40          self.assertEqual(len(result_pytesseract), 0,
41                           'patrimônio líquido (número após
                              conjunto de busca): tamanho resultado
                              (pytesseract)')
42
43      def test_roe_monetary(self):
44          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
45
46          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
47
48      def test_roe_number(self):
49          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
50
51          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
52
53      def test_roe_calculate(self):
54          calculate_roe_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
55          result_pytesseract =
               calculate_roe_pytesseract[self.filename]
56
57          self.assertEqual(len(result_pytesseract), 0, 'ROE por
```

```
            cálculo: tamanho resultado (pytesseract)')
58
59
60  if __name__ == '__main__':
61      unittest.main()
```

repositorio/tests/reports/pytesseract03/py3_test_weg20172T.py

```
1   import unittest
2   from src import manager as m
3
4
5   class TestsWeg20172T(unittest.TestCase):
6
7       filename = None
8       manager_pytesseract = None
9
10      @classmethod
11      def setUpClass(cls):
12          cls.filename = 'weg_2017_2T.pdf'
13          cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 5')
14
15      @classmethod
16      def tearDownClass(cls):
17          cls.filename = None
18          cls.manager_pytesseract = None
19
20      def test_lucro_liquido_monetary(self):
21          lucro_liquido_monetary_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_monetary()
22          result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
23
24          self.assertEqual(len(result_pytesseract), 0, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
25
26      def test_lucro_liquido_number(self):
27          lucro_liquido_number_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_number()
28          result_pytesseract =
               lucro_liquido_number_pytesseract[self.filename]
```

```
29
30          self.assertEqual(len(result_pytesseract), 0,
31                           'lucro líquido (número após conjunto de
                                busca): tamanho resultado
                                (pytesseract)')
32
33      def test_patrimonio_liquido_monetary(self):
34          result =
               self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
35
36          self.assertEqual(len(result), 0, 'patrimônio líquido
               (R$): tamanho resultado')
37
38      def test_patrimonio_liquido_number(self):
39          patrimonio_liquido_number_pytesseract =
               self.manager_pytesseract.run_patrimonio_liquido_number()
40          result_pytesseract =
               patrimonio_liquido_number_pytesseract[self.filename]
41
42          self.assertEqual(len(result_pytesseract), 0,
43                           'patrimônio líquido (número após
                                conjunto de busca): tamanho resultado
                                (pytesseract)')
44
45      def test_roe_monetary(self):
46          result =
               self.manager_pytesseract.run_roe_monetary()[self.filename]
47
48          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
               resultado')
49
50      def test_roe_number(self):
51          result =
               self.manager_pytesseract.run_roe_number()[self.filename]
52
53          self.assertEqual(len(result), 0, 'ROE (número após
               conjunto de busca): tamanho resultado')
54
55      def test_roe_calculate(self):
56          roe_calculate_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
57          result_pytesseract =
```

```
            roe_calculate_pytesseract[self.filename]
58
59          self.assertEqual(len(result_pytesseract), 0, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
60
61
62  if __name__ == '__main__':
63      unittest.main()
```

repositorio/tests/reports/pytesseract03/py3_test_weg20192T.py

```
1   import unittest
2   from src import manager as m
3
4
5   class TestsWeg20192T(unittest.TestCase):
6
7       filename = None
8       manager_pytesseract = None
9
10      @classmethod
11      def setUpClass(cls):
12          cls.filename = 'weg_2019_2T.pdf'
13          cls.manager_pytesseract = m.manager([cls.filename],
               'pytesseract', '--psm 5')
14
15      @classmethod
16      def tearDownClass(cls):
17          cls.filename = None
18          cls.manager_pytesseract = None
19
20      def test_lucro_liquido_monetary(self):
21          lucro_liquido_monetary_pytesseract =
               self.manager_pytesseract.run_lucro_liquido_monetary()
22          result_pytesseract =
               lucro_liquido_monetary_pytesseract[self.filename]
23
24          self.assertEqual(len(result_pytesseract), 0, 'lucro
               líquido (R$): tamanho resultado (pytesseract)')
25
26      def test_lucro_liquido_number(self):
27          lucro_liquido_number_pytesseract =
```

```
            self.manager_pytesseract.run_lucro_liquido_number()
28          result_pytesseract =
            lucro_liquido_number_pytesseract[self.filename]
29
30          self.assertEqual(len(result_pytesseract), 0,
31                           'lucro líquido (número após conjunto de
                               busca): tamanho resultado
                               (pytesseract)')
32
33      def test_patrimonio_liquido_monetary(self):
34          result =
            self.manager_pytesseract.run_patrimonio_liquido_monetary()[self.file
35
36          self.assertEqual(len(result), 0, 'patrimônio líquido
            (R$): tamanho resultado')
37
38      def test_patrimonio_liquido_number(self):
39          patrimonio_liquido_number_pytesseract =
            self.manager_pytesseract.run_patrimonio_liquido_number()
40          result_pytesseract =
            patrimonio_liquido_number_pytesseract[self.filename]
41
42          self.assertEqual(len(result_pytesseract), 0,
43                           'patrimônio líquido (número após
                               conjunto de busca): tamanho resultado
                               (pytesseract)')
44
45      def test_roe_monetary(self):
46          result =
            self.manager_pytesseract.run_roe_monetary()[self.filename]
47
48          self.assertEqual(len(result), 0, 'ROE (R$): tamanho
            resultado')
49
50      def test_roe_number(self):
51          result =
            self.manager_pytesseract.run_roe_number()[self.filename]
52
53          self.assertEqual(len(result), 0, 'ROE (número após
            conjunto de busca): tamanho resultado')
54
55      def test_roe_calculate(self):
```

```
56          roe_calculate_pytesseract =
               self.manager_pytesseract.run_calculate_roe()
57          result_pytesseract =
               roe_calculate_pytesseract[self.filename]
58
59          self.assertEqual(len(result_pytesseract), 0, 'ROE por
               cálculo: tamanho resultado (pytesseract)')
60
61
62  if __name__ == '__main__':
63      unittest.main()
```