UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Cristian Assis Rocha Yamamoto

Implementação de algoritmos e sistemas de recomendações para uma plataforma imobiliária

Florianópolis
2021

Cristian Assis Rocha Yamamoto

Implementação de algoritmos e sistemas de recomendações para uma plataforma imobiliária

Relatório final da disciplina DAS5511 (Projeto de Fim de Curso) como Trabalho de Conclusão do Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Santa Catarina em Florianópolis.
Orientador: Prof. Rômulo Silva de Oliveira, Dr.
Supervisor: Andrio Renan Gonzatti Frizon, Eng.

Florianópolis
2021

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Yamamoto, Cristian Assis Rocha
Implementação de algoritmos e sistemas de recomendações para uma plataforma imobiliária / Cristian Assis Rocha Yamamoto ; orientador, Rômulo Silva de Oliveira, 2021.
82 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia de Controle e Automação, Florianópolis, 2021.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Sistemas de recomendações. 3. Plataforma Imobiliária. 4. Ciência de dados. I. Oliveira, Rômulo Silva de. II. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. III. Título.

Cristian Assis Rocha Yamamoto

Implementação de algoritmos e sistemas de recomendações para uma plataforma imobiliária

Esta monografia foi julgada no contexto da disciplina DAS5511 (Projeto de Fim de Curso) e aprovada em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação

Florianópolis, 13 de dezembro de 2021.

Prof. Hector Bessa Silveira, Dr.
Coordenador do Curso

Banca Examinadora:

Prof. Rômulo Silva de Oliveira, Dr.
Orientador
UFSC/CTC/DAS

Andrio Renan Gonzatti Frizon, Eng.
Supervisor
Jungle Devs

Prof. Eric Aislan Antonelo, Dr.
Avaliador
UFSC/CTC/DAS

Prof. Eduardo Camponogara, Dr.
Presidente da Banca
UFSC/CTC/DAS

AGRADECIMENTOS

À minha família, pelo apoio durante toda a graduação, em especial ao meu irmão Ígor, por me auxiliar durante todo desenvolvimento do projeto.

Aos meus colegas de 2016.1 da Engenharia de Controle e Automação, por terem me acompanhado durante toda essa jornada e por terem criado uma amizade vitalícia.

À equipe da Jungle Devs, pela oportunidade única de alavancar meu crescimento pessoal e profissional.

Aos professores do Departamento de Automação e Sistemas, em especial ao professor Rômulo, por sempre ter me atendido com muito respeito e prontidão, mesmo em tempos de pandemia.

E, por fim, à Universidade Federal de Santa Catarina, por ter me proporcionado experiências incríveis que me fizeram crescer como pessoa.

RESUMO

Este documento trata do Projeto de Fim de Curso (PFC) realizado na Jungle Devs, empresa de desenvolvimento de software. Atualmente muitas empresas utilizam sistemas de recomendações para apresentar conteúdos relevantes ao usuário, essa tecnologia é essencial para aumentar a interação entre empresa e cliente, permitindo uma experiência otimizada. O objetivo do PFC é apresentar uma implementação de um sistema de recomendações para a plataforma imobiliária do Canal Digital, e que tem como a finalidade aprimorar a experiência do usuário final. Durante o projeto, foram feitas análises dos tipos de sistemas de recomendações existentes e quais desses seriam utilizados durante a implementação no projeto, aqui incluem-se sistemas personalizados nos quais abrangem filtragem baseada em conteúdo, colaborativa e híbrida, e ainda, sistemas não personalizados baseados em popularidade. O sistema como um todo foi dividido em dois escopos, a recomendação *online* e a *offline*, o maior foco para esse PFC foi a parte *offline*. Dentro desse escopo, foram realizadas a extração dos dados de imóveis do banco de dados, a preparação e tratamento dos dados e a clusterização dos dados, salvando os resultados para poderem ser utilizados na parte *online* do sistema de recomendação. O projeto teve como base a metodologia CRISP-DM, um modelo padrão que aborda problemas de ciência de dados, tendo como etapas as fases de entendimento do negócio, de compreensão dos dados, de preparação dos dados, de modelagem, de avaliação e de implantação. Por fim, com os resultados obtidos, foi possível observar os impactos positivos e os pontos de melhorias que devem ser considerados durante a implementação do sistema na aplicação.

Palavras-chave: Sistemas de Recomendações. Plataforma Imobiliária. Ciência de Dados. Mineração de Dados.

ABSTRACT

This document refers to the End of Course Project carried out at Jungle Devs, a software development company. Nowadays many companies use recommendation systems to present relevant content to the user, this technology is essential to increase the interaction between the company and the customer, allowing an optimized experience. The purpose of the project is to present an implementation of a recommendation system for Canal Digital's real estate platform, which aims to improve the end-user experience. During the project, analysis were made regarding the types of existing recommendation systems and which of these were used during the implementation of the project, these types are the following: personalized systems which cover content-based, collaborative and hybrid filtering, and popularity-based non-personalized systems. The system as a whole was divided into two scopes, the *online* and *offline* recommendation, the major focus for this project was the offline part. Within this scope, the extraction of real estate data from the database, data preparation and treatment, and data clustering were performed, saving the results to be used in the *online* part of the recommendation system. The project was based on the CRISP-DM methodology, a standard model that addresses data mining issues, which has six major phases: Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation and Deployment. Finally, as a result, it was possible to observe the positive impacts and points of improvement that should be considered during the implementation of the system in the application.

Keywords: Recommendation Systems. Real Estate Platform. Data Science. Data Mining.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ALS	*Alternating Least Squares*
AWS	*Amazon Web Services*
CRISP-DM	*Cross-Industry Standard Process for Data Mining*
CSV	*Comma-separated values*
DBSCAN	*Density-based spatial clustering of applications with noise*
HTML	*HyperText Markup Language*
IMDb	*Internet Movie Database*
JSON	*JavaScript Object Notation*
MAE	*Mean absolute error*
MAP	*Mean Average Precision*
MSE	*Mean Squared Error*
NDCG	*Normalized Discounted Cumulative Gain*
PDF	*Portable Document Format*
PM	*Product Manager*
reST	*reStructuredText*
RMSE	*Root Mean Squared Error*
S3	*Simple Storage Service*
SaaS	*Software as a Service*
SGD	*Stochastic Grandient Descent*
SVM	*Support Vector Machine*
TI	Tecnologia da Informação
UI	*User Interface*
URL	*Uniform Resource Locator*
UX	*User Experience*

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo será introduzido a problemática global dentro da qual o tema foi escolhido, descrevendo as motivações e os objetivos pelos quais o projeto se baseou. Além disso, é apresentado como o documento está estruturado.

O tema do PFC (Projeto de Fim de Curso) se relaciona diretamente à área de atuação da Informática dentro do curso de Engenharia de Controle e Automação, entrando em estudos de algoritmos e aplicações de *Data Science*, e se integrando com a engenharia e o desenvolvimento de *softwares*.

O trabalho coloca em prática vários conceitos vistos durante o curso em um projeto real. Disciplinas que abordaram conceitos de tecnologia de informação, linguagens de programação, algoritmos, metodologias e técnicas de desenvolvimento de *softwares*, avaliação e integração de sistemas, e também, manipulação e tratamento de dados foram essenciais para o desenvolvimento do trabalho.

1.1 MOTIVAÇÃO

A Jungle Devs atualmente trabalha em conjunto com a Woliver no desenvolvimento da plataforma imobiliária Canal Digital, a plataforma tem como objetivo fazer agendamento de visitas, alugueis, propostas e negociações de imóveis.

O projeto foi escolhido com intuito de melhorar a experiência do usuário final quando se trata de recomendações de imóveis. Previamente à realização do projeto, o sistema de recomendações do Canal Digital era implementado de forma simplificada, e portanto tendo um grande potencial para melhorias com a introdução de algoritmos e sistemas mais complexos de recomendações.

A implementação de um sistema de recomendações é uma maneira simples de tornar o produto mais personalizado sem precisar reinventar o mercado. Hoje em dia, é possível encontrar empresas de muito sucesso utilizando dessa ferramenta para alavancar os negócios, e uma motivação para o projeto foram empresas como a *Netflix*, que revolucionou a área com o *Netflix Prize* (BELL; KOREN, 2007), a *Amazon* (HARDESTY, LARRY, 2019), maior plataforma de comércio eletrônico do mundo e que sem os algoritmos de recomendações possivelmente não teria crescido a esse nível, e ainda, com uma relação grande com esse projeto por ser o maior portal de imóveis dos EUA (Estados Unidos da América), a *Zillow Group* que compartilham o uso dessa tecnologia (ZILLOW, 2017).

Há uma grande motivação social e econômica por trás do projeto, dado que, com a implementação de um sistema mais otimizado na plataforma, as chances do usuário final possuir uma experiência ótima são muito maiores. Considerando a grande quantidade e variedade de escolhas que o usuário possui, pode-se afirmar que é muito difícil e trabalhoso analisar cada uma individualmente, e portanto, com um sistema de

recomendação bem implementado, o serviço oferecido facilitaria a vida do mesmo, pois seria possível economizar horas de procura pelo produto ideal, evitando frustrações que poderiam causar impactos negativos na empresa. E ainda, com uma satisfação de clientes aumentada, seria mais fácil atrair novos usuários e escalar o produto trazendo retornos positivos para empresa.

1.2 OBJETIVOS

O objetivo geral do projeto foi desenvolver e aprimorar o sistema de recomendações para a plataforma Canal Digital, começando com o entendimento da finalidade do negócio e do problema em si, podendo emoldurar em mais objetivos e questões de ciência de dados

Com o objetivo geral definido, pode-se elencar os objetivos específicos. Esses podem ser melhor compreendidos ao observar as fases da metodologia utilizada como base nesse projeto, a metodologia CRISP-DM (Cross-Industry Standard Process for Data Mining). É possível pontuá-los da seguinte forma:

- Objetivo de compreender os dados. Realizando a coleta inicial, explorando, identificando e descrevendo quais dados são mais importantes e relevantes para a resolução do problema.
- Objetivo de preparar os dados. Após identificados os conjuntos de dados que serão utilizados, faz-se a extração desses e, com os dados brutos, são feitas transformações e formatações para que se adéquem aos algoritmos que são desejados na aplicação do sistema.
- Objetivo de modelar o sistema. Com os dados preparados, pode-se partir para a aplicação de fato das técnicas e algoritmos estudados.
- Objetivo de avaliar o sistema. Acompanhando e alinhando os resultados obtidos em relação ao que era esperado e aos critérios de sucesso da empresa.

1.3 ESTRUTURA DO DOCUMENTO

Este documento está dividido em sete capítulos, sendo um deles o atual capítulo de introdução, e mais seis outros capítulos estruturados da seguinte forma:

No capítulo 2, será feita a contextualização do projeto contendo a descrição da empresa (Jungle Devs) na qual foi realizado o projeto e como a aplicação se encaixa dentro da empresa.

No capítulo 3, serão apresentados os conceitos fundamentais que formaram a base teórica do projeto, com explicações da metodologia, ambientes, algoritmos e conceitos técnicos utilizados durante o desenvolvimento.

No capítulo 4, será apresentado o projeto no qual o sistema será aplicado do ponto de visão de um usuário, trazendo o fluxo de funcionamento do projeto.

No capítulo 5, serão apresentadas as atividades realizadas durante o desenvolvimento e a implementação do projeto, contendo as informações de como foi feita a integração do sistema.

No capítulo 6, serão apresentados os resultados obtidos durante o desenvolvimento, observando os impactos do sistema no processo e levantando as vantagens e desvantagens da implementação.

E por último, no capítulo 7, serão feitas as perspectivas e conclusões finais do projeto.

2 CONTEXTUALIZAÇÃO

2.1 A EMPRESA

Este PFC foi desenvolvido durante a realização do estágio obrigatório na Jungle Devs, empresa originalmente sediada em Florianópolis, porém atualmente possui como característica o conceito de *remote first*, ou seja, no contexto da pandemia do coronavírus, agora todos os colaboradores trabalham remotamente em tempo integral. Isso fez com que pessoas do Brasil inteiro pudessem ingressar na empresa.

Hoje em dia, a Jungle possui mais de 100 funcionários, sendo que grande parte deles entraram apenas durante o último ano, demonstrando o grande potencial de crescimento que a empresa possui. E apesar de ser uma empresa brasileira, ela tem o inglês como língua oficial, pois além de possuir clientes ao redor do mundo inteiro em 7 países diferentes (Figura 1), há também um foco no crescimento da carreira pessoal de cada um, já que é a principal língua utilizada dentro da área de atuação da Jungle.

Figura 1 – *Jungle Devs by the numbers*

A empresa é focada no setor de TI (Tecnologia da Informação) e atua na área de desenvolvimento de *softwares*, especializando em aplicações para *Web* e *Mobile*. Os serviços da Jungle abrangem as áreas de *front-end* e *back-end* com perícia em tecnologias como Django, Node, React, Ruby on Rails, Kotlin entre outras, está presente na área de design de produtos atuando com técnicas de UI/UX (*User Interface* e *User Experience*), possui diversos gerentes de projetos que acompanham diariamente os trabalhos realizados utilizando métodos ágeis e outras técnicas de gerenciamento, também com presença de profissionais *DevOps* que trabalham com *AWS* (*Amazon Web Services*), fazendo a integração, implantação, monitoramento e manutenção das aplicações, e ainda atua na parte de *Data Science*, coletando, administrando e convertendo

dados crus em informações úteis.

A empresa foi fundada em 2018 por egressos do curso de Engenharia de Controle e Automação na UFSC (Universidade Federal de Santa Catarina) e teve desde o início como o principal missão e lema repensar a forma como as pessoas aprendem e criam tecnologia. E com isso, consegue-se afirmar que esse PFC só foi possível ser realizado por causa da existência do programa de capacitação da Jungle, chamado de programa *Academy* e que possui duração de até 1 ano. Nele a empresa da oportunidade a estagiários que partem do nível mais introdutório até o lançamento da carreira profissional do indivíduo, passando por períodos de fundamentação e aprendizado de conceitos básicos até a atuação no desenvolvimento de um projeto real, fazendo conexões com profissionais experientes e que já estão inseridos no mercado de trabalho.

Figura 2 – Logo da Jungle Devs

A estruturação e organização da Jungle Devs é baseada no modelo de tribos do *Spotify* (Figura 4), possuindo os grupos *Gigs*, *Tribes*, *Chapters* e *Guilds*, além do time de liderança da empresa.

Os *Gigs* são projetos que duram um certo período de tempo, até um ciclo acabar ou enquanto a empresa continuar com a necessidade específica. Podem ter um prazo curto e específico de duração ou um longo prazo e durar enquanto for preciso.

As *Tribes* são feitas para serem como pequenas empresas com múltiplos projetos, tendo todas as habilidades e ferramentas necessárias para desenvolver um projeto, que incluem gerenciamento, design, desenvolvimento, testes e lançamentos. Atualmente a empresa possui quatro *Tribes*, Nirvana, Appy, Woliver e Hapu, nomes inspirados em companhias parceiras, sendo que cada uma possui um *Tribe Lead*, que é responsável por coordenar os diferentes projetos dentro de uma tribo.

Os *Chapters* são grupos de pessoas que compartilham de habilidades similares e trabalham dentro de uma mesma área de competência. Cada *Chapter* possui um *Chapter Lead*, no qual tem responsabilidades tradicionais como organizar reuniões, discussões e compartilhar conhecimento e ideias através do *Chapter*. O líder ajuda a entender o que é importante para as pessoas da mesma área de acordo com o feedback dos membros do *Chapter*. E a presença do líder faz com que exista uma relação que corresponda ao modelo de ”*professor and entrepreneur*” ou ”professor e

empresário" recomendado por Mary e Tom Poppendieck (NARAYAN, 2015), onde o PM (*Project Manager*) é o empresário que tem como foco entregar um ótimo produto de qualidade e o *Chapter Lead* é o professor que foca na excelência técnica (Figura 3). Atualmente os *Chapters* existentes na empresa são de PM, Data, Design, Android, iOS, Frontend, Backend, Hybrid e People

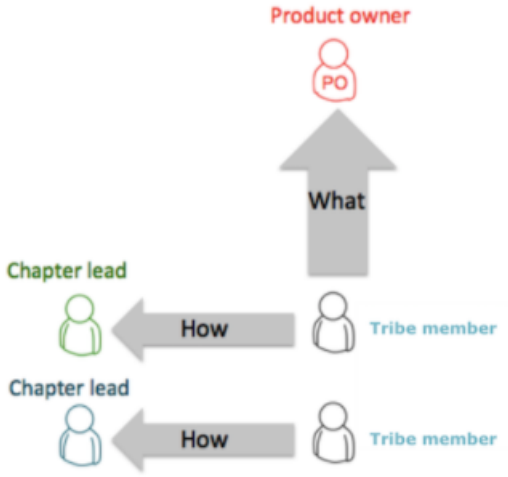

Figura 3 – Relação entre *Chapter Lead*, *Product Owner/Manager* e membro da tribo

E ainda, as *Guilds* que são grupos comunitários de interesses em comum, onde as pessoas podem entrar e sair a qualquer momento.

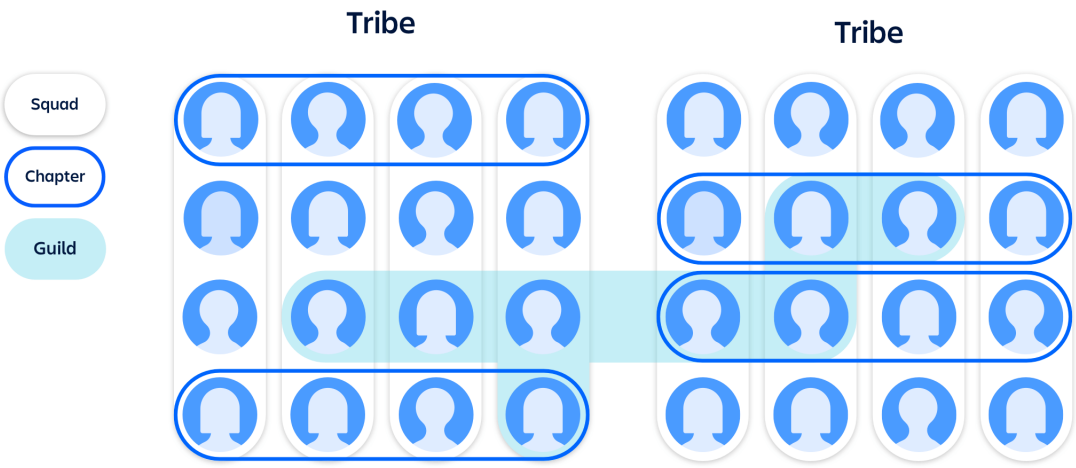

Figura 4 – Estrutura do Modelo *Spotify*

2.2 O SISTEMA DE RECOMENDAÇÕES

O projeto a ser desenvolvido está envolvido com o serviço de *Data Science* da empresa. Como dito anteriormente, a Jungle Devs tem parceria com a Woliver (Figura 5), empresa a qual tem o objetivo de usar a tecnologia para facilitar os processos imobiliários e aumentar os resultados financeiros a partir disso.

A Woliver desenvolveu um SaaS (*Software as a Service*), no qual oferece serviços de aluguel, agendamento de visitas e negociações de imóveis como um produto de *White Label* ou marca branca, ou seja, o serviço pode ser contratado por uma imobiliária porém cada uma tem a liberdade de ter sua própria marca utilizando todos os recursos que o sistema oferece. Essa plataforma se chama Canal Digital.

Um dos recursos que a plataforma tem é o sistema de recomendações de imóveis, o qual foi o foco de desenvolvimento desse PFC.

Figura 5 – Logo da Woliver

3 BASE TEÓRICA

Neste capítulo serão apresentadas as descrições conceituais da metodologia e das tecnologias que guiaram o desenvolvimento do projeto.

3.1 DATA SCIENCE

Este projeto tem como base a área de estudos de *Data Science* ou Ciência de Dados, portanto é fundamental entender os conceitos iniciais e a importância dessa área para poder compreender o contexto de aplicação do projeto realizado.

Nos últimos anos, tem-se observado grandes investimentos na infraestrutura de negócios, com isso a habilidade de coletar dados em um empreendimento teve um avanço significativo, apresentando muitas melhorias. Dessa forma, praticamente todos os aspectos de um negócio estão abertos para coleção de dados e podendo ser até instrumentos para coleção de dados. Aspectos de operações, manufatura, gestão de cadeia de suprimentos, comportamento do cliente, desempenho da campanha de marketing, procedimentos de fluxo de trabalho e assim por diante. Ao mesmo tempo, também houve um aumento da disponibilidade de informações. Portanto, somando todos os pontos, surgiu o aumento do interesse e a necessidade de encontrar métodos para extrair conhecimento e informações úteis de dados, e é dentro desse contexto que ergue-se o conceito de *Data Science*. (PROVOST; FAWCETT, 2013)

Resumidamente, *Data Science* é o processo de transformar dados estruturados e não-estruturados em informações úteis. Através da análise dos dados que consegue-se chegar a essas informações e então são obtidos *insights* ou percepções. A partir disto, é possível identificar tendências, padrões e correlações, nos quais o cientista de dados precisa contextualizar e entender tudo que foi extraído para que, no final, consiga aplicar a informação gerada em algo que faça sentido.

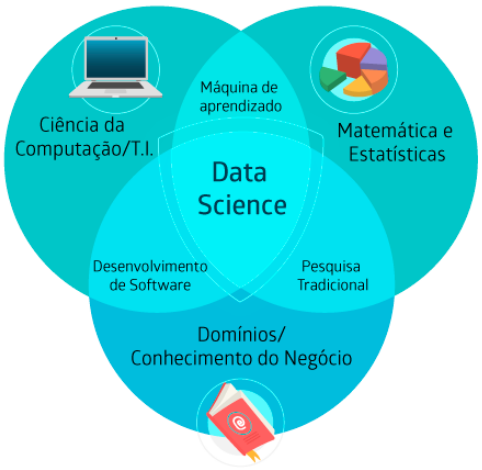

Figura 6 – Interdisciplinaridade de *Data Science*

Na Figura 6, observa-se que a Ciência de Dados é uma área interdisciplinar que engloba conhecimentos de matemática, estatística, computação e negócios.

A fim entender melhor como o processo todo funciona, a seguir será explicado com mais detalhes a metologia base utilizada para aplicar os conceitos de *Data Science* durante o projeto, que nada mais é que um modelo de processo padrão que descreve como abordar um problema de *Data*.

3.1.1 Metodologia CRISP-DM

A metodologia CRISP-DM ou *Cross Industry Standard Process for Data Mining* é descrita em termos de um modelo de processo hierárquico, e consiste de conjuntos de tarefas descritas em quatro níveis de abstração (de geral a específico): fases, tarefas genéricas, tarefas específicas, e instâncias de processos (Figura 7) (CHAPMAN *et al.*, 2000).

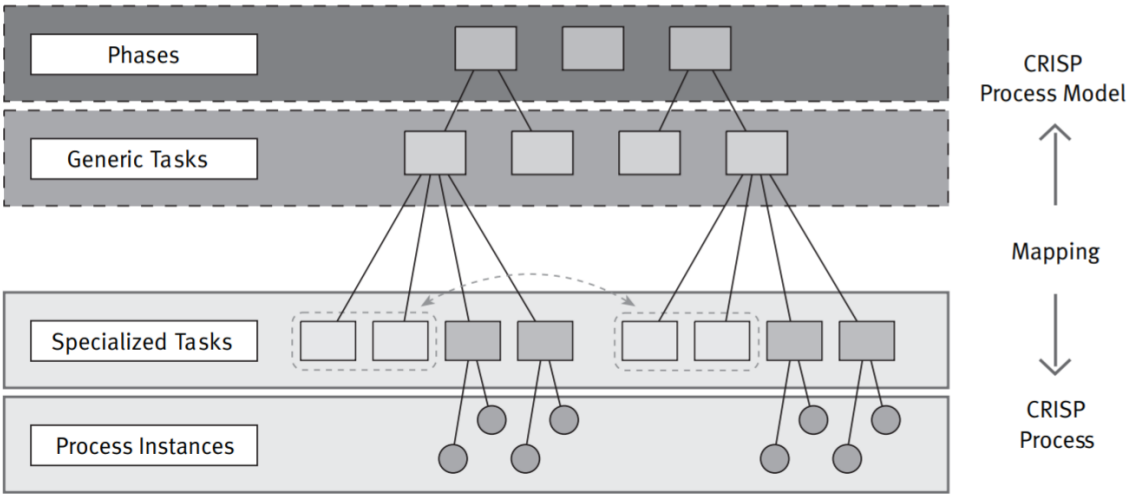

Figura 7 – Separação dos quatro níveis da metodologia

No nível superior, o processo de mineração de dados é organizado em fases, cada fase consiste em diversas tarefas genéricas. O segundo nível é chamado de genérico pois tem a intenção de cobrir de uma forma geral todas as situações possíveis de mineração de dados. As tarefas genéricas devem ser completas, significando que deve cobrir todo o processo de *Data Mining* e todas as aplicações possíveis e também estáveis, ou seja, o modelo deve ser válido para desenvolvimentos imprevistos, como novas técnicas de modelagem.

O terceiro nível, de tarefas específicas, é onde são descritas como as ações nas tarefas genéricas devem ser realizadas em determinadas situações específicas. Por exemplo, uma tarefa genérica pode ser chamada de limpar dados, o terceiro nível descreve como essa tarefa se diferencia em diferentes situações, como limpar dados numéricos contra limpar dados categóricos.

O quarto nível, instâncias de processos, é um registro das ações, decisões e os resultados de uma mineração de dados real. Uma instância de processo é organizada de acordo com as tarefas definidas em níveis superiores, porém representam o que realmente aconteceu em um engajamento particular, ao invés do que acontece em geral.

3.1.1.1 Modelo de referência do CRISP-DM

O ciclo de vida de um projeto de *Data Mining* consiste em seis fases (Figura 8). A sequência das fases não é sempre a mesma pois há a necessidade de toda hora ter uma movimentação de uma fase para outra. O resultado de cada fase irá determinar qual fase será realizada na sequência.

Observe que não é necessário falhar na implantação para iniciar o ciclo novamente. O estágio de avaliação pode revelar que os resultados não são satisfatórios e assim é necessário ajustar a definição do problema ou obter dados diferentes. Isso é representado pela conexão da fase de avaliação de volta à fase de entendimento de negócios. Na prática, devem existir atalhos de cada estágio para cada antecessor, porque o processo sempre retém alguns aspectos exploratórios, e um projeto deve ser flexível o suficiente para revisitar as etapas anteriores com base nas descobertas feitas.

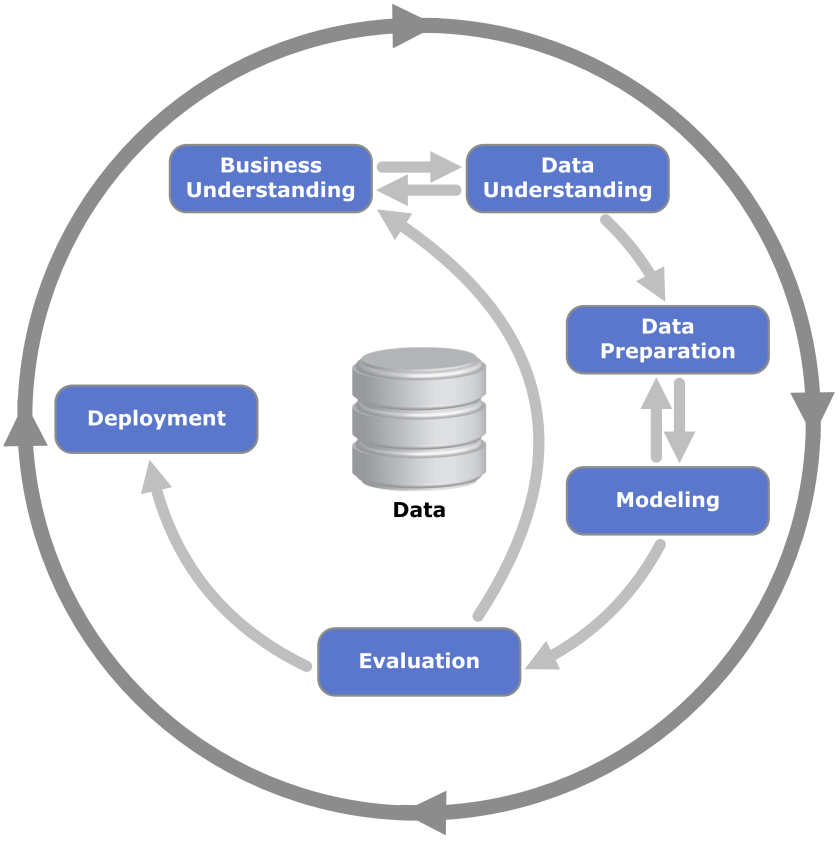

Figura 8 – Ciclo de vida CRISP-DM

O círculo externo da Figura 8 simboliza a natureza cíclica da mineração de dados, que não termina uma vez que a solução é implantada. As lições aprendidas durante o processo e da solução implantada podem gerar novas questões de negócios. O processos subsequentes irão se beneficiar das experiências anteriores.

A seguir, uma breve explicação de cada fase do modelo de referência do CRISP-DM:

- ***Business Understanding***: Fase de entendimento do negócio. É nessa fase que o objetivo do negócio e o entendimento do problema são definidos e emoldurados em um ou mais objetivos de *Data science*. A situação é avaliada e são levantados recursos, requisitos, premissas e restrições, é feito um plano de projeto e uma avaliação das ferramentas e técnicas iniciais.

- ***Data Understanding***: Fase de compreensão dos dados. É a fase na qual é realizada a coleta inicial dos dados, descrevendo, explorando e identificando quais dados são mais importantes para a resolução do problema.

- ***Data Preparation***: Fase de preparação dos dados. Nessa fase, após identificados os conjuntos de dados que serão utilizados, parte-se para uma etapa mais técnica, onde os dados brutos serão transformados e formatados para que seja possível aplicar os algoritmos desejados que resolvam o problema definido.

- ***Modeling***: Fase de modelagem. Assim que os dados são preparados, é possível então partir para a fase na qual serão aplicadas de fato as técnicas de modelagem utilizando uma abordagem experimental. Serão configurados e revisados os parâmetros dos modelos de construção e então esses serão avaliados para chegar à melhor alternativa.

- ***Evaluation***: Fase de avaliação. Nessa fase serão acompanhados e alinhados os resultados obtidos em relação aos objetivos e os critérios de sucesso da empresa. O processo será revisado e então aprovado ou não, para que em seguida possa ser determinado o próximo passo do projeto.

- ***Deployment***: Fase de implantação. Nessa fase serão feitos os planos de implantação, monitoramento e manutenção do projeto. Os modelos produzidos devem já estar prontos para uso em produção. Também é produzido um relatório final, resumindo e organizando os resultados obtidos e por fim revisando o que deu certo e o que deu errado no projeto.

Na Figura 9, observa-se um esboço das fases acompanhadas por tarefas genéricas (negrito) e resultados (itálico).

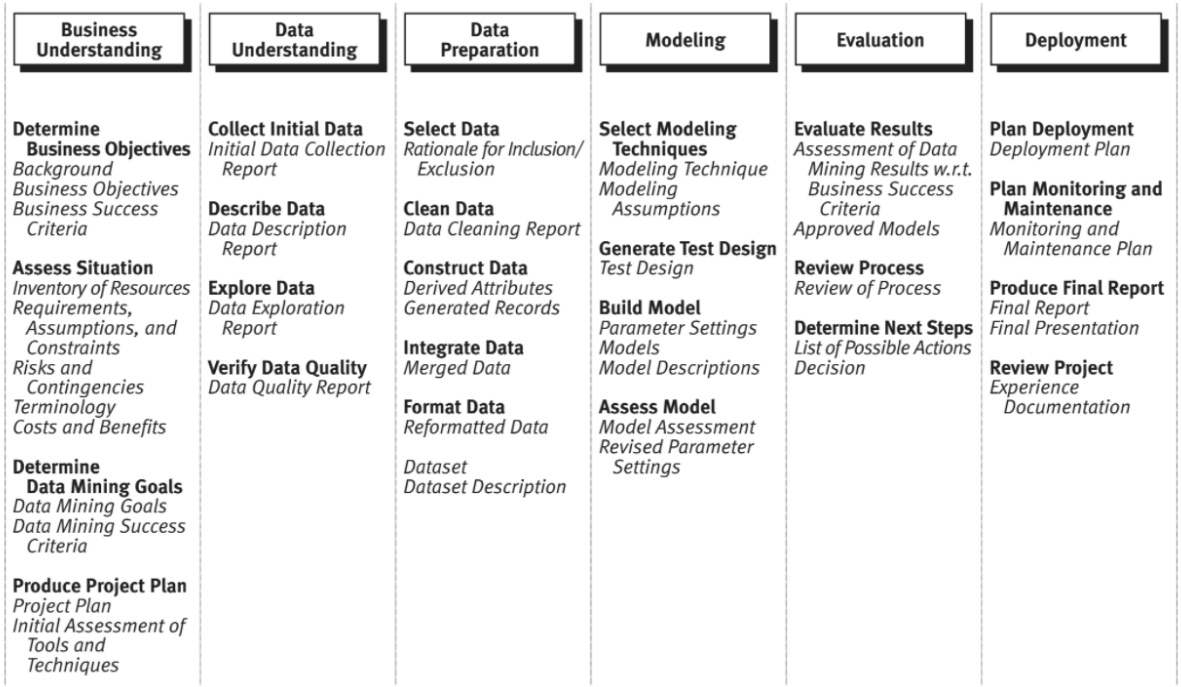

Figura 9 – Tarefas genéricas (negrito) e resultados (itálico) do modelo referência do CRISP-DM (CHAPMAN *et al.*, 2000)

3.2 PROGRAMAÇÃO

Foram indispensáveis os conceitos básicos de programação para o desenvolvimento do projeto, pois programar é uma das principais habilidades para conseguir aplicar problemas de *Data Science*. Através das linguagens de programação, consegue-se automatizar as análises de grande quantidade de dados que uma pessoa não conseguiria processar em tempo hábil e de forma satisfatória.

Durante a graduação foram estudados conceitos que abrangem lógicas de programação, entendimento do que são algoritmos, introdução a definição de variáveis, tipos e estruturas de dados, operadores aritméticos, relacionais e lógicos, controle de fluxo com estruturas de repetição e estruturas condicionais de tomadas de decisões, rotinas, sub-rotinas, métodos e funções, processos de iteração e recursão, e abstração de dados e objetos.

Outro conceito importante dentro das linguagens de programação, são os paradigmas, isso porque eles definem o estilo de programação e a maneira de como a solução de um problema computacional será abordada. Os principais paradigmas são: o Paradigma de Programação Imperativa ou Procedural, onde os programas são sequências de comandos em que o computador irá executar em uma ordem estabelecida; o Paradigma de Programação Declarativa, que descreve o que faz mas não especifica como o algoritmo em si deve agir; o Paradigma de Programação Funcional, no qual trata a computação como uma avaliação de funções matemáticas e evita

estados e dados mutáveis; o Paradigma de Programação Lógica ou Restritiva, que se baseia na utilização de sentenças lógicas e usa lógica simbólica; e o Paradigma de Programação Orientada a Objetos, sendo o mais difundido e fundamentado na interação entre objetos.

A seguir será feita uma descrição da linguagem de programação utilizada durante o projeto, que é uma linguagem multiparadigma, ou seja, suporta diversos paradigmas de desenvolvimento, como, orientado a objetos, funcional e imperativo.

3.2.1 Python

Atualmente, a linguagem de programação mais popular na área de *Data Science* é o Python (logo na Figura 10), no qual foi a linguagem escolhida para o desenvolvimento do sistema nesse trabalho. Com o Python (VAN ROSSUM; DRAKE JR, 1995) existe uma facilidade de gerar visualizações claras de todas as informações extraídas, interpretando e comunicando os resultados da melhor forma para todos envolvidos dentro de um negócio.

Figura 10 – Logo do Python

O Python possui muitas vantagens que podem ser citadas. A comunidade é ampla e possui muita popularidade, sendo fácil encontrar conteúdo em todos os lugares. Há um amplo número de bibliotecas e ferramentas utilizadas nessa área para facilitar o desenvolvimento de um projeto, como *NumPy*, *SciPy*, *scikit-learn*, *Jupyter Notebook*, *pandas*, entre outras, muitas delas utilizadas nesse projeto. E ainda, há uma facilidade de aprender a linguagem, já que possui uma sintaxe simples, fazendo com seja uma linguagem de muita praticidade.

Vale a pena salientar que um dos grande motivos da escolha dessa linguagem foi pelo fato do projeto do Canal Digital ser desenvolvido, na parte de *backend*, com o *framework* Django, que utiliza Python como linguagem, e portanto existe uma maior conveniência na integração do sistema de recomendações com o projeto da imobiliária.

3.2.2 Ferramentas e Bibliotecas

Nessa seção serão cobertas as principais ferramentas e bibliotecas que foram utilizadas para realização do projeto.

3.2.2.1 Jupyter Notebook

O Jupyter Notebook (KLUYVER *et al.*, 2016) foi uma ferramenta essencial para o desenvolvimento desse projeto, essa ferramenta é utilizada dentro do ambiente de desenvolvimento interativo baseado na web JupyterLab, que possui outros recursos além do Jupyter Notebook.

Figura 11 – Logo do Jupyter

O Jupyter Notebook é uma aplicação web *open-source* (código aberto) que permite a criação e compartilhamento de documentos que possuem código em tempo real, equações, visualizações e texto de narrativa. Com essa ferramenta é possível fazer limpeza e transformação de dados, simulação numérica, modelagem estatística, visualização de dados, aplicações de aprendizado de máquina, entre outras aplicações, e por isso é uma ótima ferramenta para o contexto desse projeto.

Esse é separado em dois componentes: Um aplicativo web, que é uma ferramenta baseada em navegador para criação interativa de documentos que combinam texto explicativo, matemática, cálculos e uma saída de mídia, e em documentos de Notebook, que são uma representação de todo conteúdo visível no aplicativo da web,

incluindo entradas e saídas de cálculos, texto explicativo, matemática, imagens e representações de objetos em mídia interativa.

Os principais recursos da aplicação web são:

- Edição de código no navegador, com destaque automático de sintaxe, indentação e conclusão/introspecção de tabulação.
- A capacidade de executar código a partir do navegador, com os resultados dos cálculos anexados ao código que os gerou.
- Exibição do resultado da computação usando representações de mídia interativa, como HTML, LaTeX, PNG, SVG, etc. Por exemplo, figuras com qualidade de publicação renderizadas pela biblioteca matplotlib podem ser incluídas.
- A edição no navegador para formato *Rich Text* usando a linguagem de marcação *Markdown*, que pode fornecer comentários para o código, não se limitando ao texto simples.
- A capacidade de incluir facilmente notação matemática em células de *markdown* usando LaTeX e renderizada nativamente por *MathJax*.

Os documentos de Notebook contêm as entradas e saídas de uma sessão interativa, bem como o texto adicional que acompanha o código, mas é feito para execução. Dessa forma, os arquivos do notebook podem servir como um registro computacional completo de uma sessão, intercalando o código executável com texto explicativo, matemática e representações dos objetos resultantes.

Esses documentos são internamente arquivos JSON (CROCKFORD, 2006) que são salvos com a extensão .ipynb. Como JSON é um formato de texto simples, é possível fazer um controle de versionamento e podem ser compartilhados com colegas. Os Notebooks podem ser exportados para uma variedade de formatos estáticos, como HTML, reST, LaTeX, PDF e apresentações de slides.

Além disso, qualquer documento de notebook .ipynb disponível em uma URL pública pode ser compartilhado por meio do *Jupyter Notebook Viewer*. Este serviço carrega o documento do bloco de notas da URL e o renderiza como uma página da web estática. Os resultados podem ser compartilhados com um colega ou como uma postagem de blog público, sem que outros usuários precisem instalar o Jupyter Notebook.

Na Figura 12, pode ser visto um exemplo de ambiente do Jupyter Notebook.

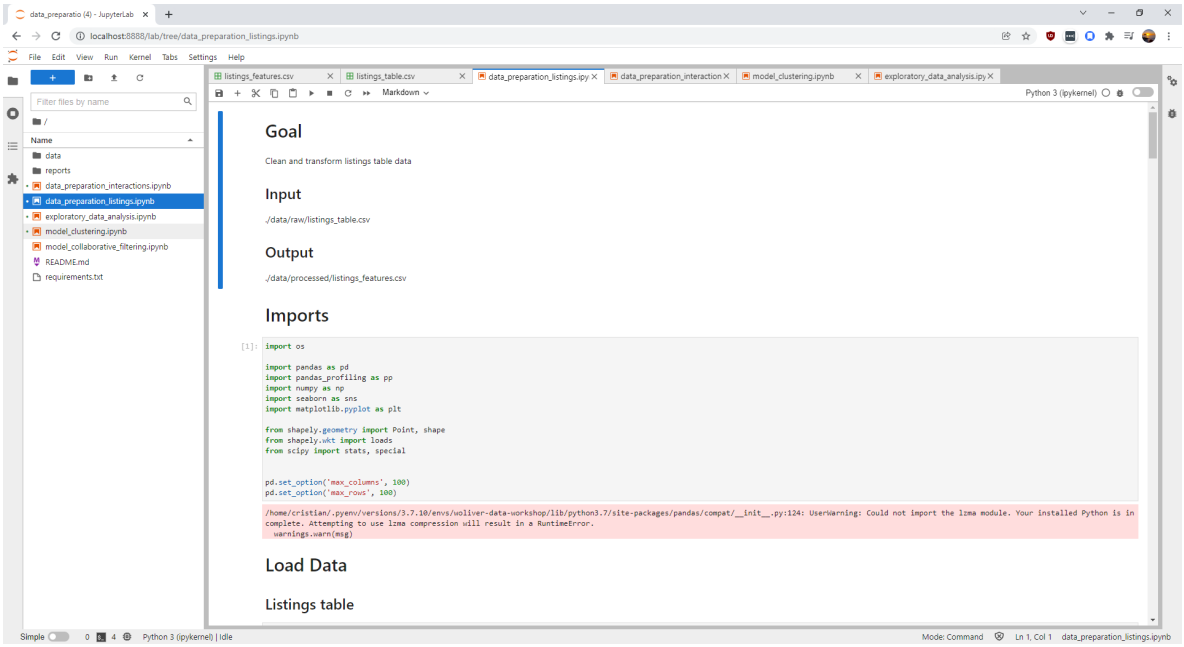

Figura 12 – Ambiente do Jupyter Notebook

3.2.2.2 Bibliotecas do Python

Algumas bibliotecas do Python foram essenciais para o desenvolvimento do projeto, dado que são ferramentas poderosas para a manipulação e análise de dados. A seguir serão descritas as que foram julgadas mais importantes no contexto do PFC.

Figura 13 – Logo do pandas

Pandas (TEAM, 2020) (logo na Figura 13) é uma delas, essa biblioteca *open-source*, com nome derivado do termo *panel data*, fornece diversos recursos para manipulação de dados e ela permite trabalhar com diferentes tipos de dados. Os dois principais objetos da biblioteca são as *Series* e os *DataFrames*.

As *Series* são tipos de array unidimensionais de qualquer tipo de dado especificado no módulo do pandas. Já o *DataFrame* é uma estrutura de dados bidimensional com tamanho mutável, onde os dados são alinhados de forma tabular em linhas e

colunas. Um *DataFrame* consiste em três componentes principais: os dados, linhas e colunas (Figura 14).

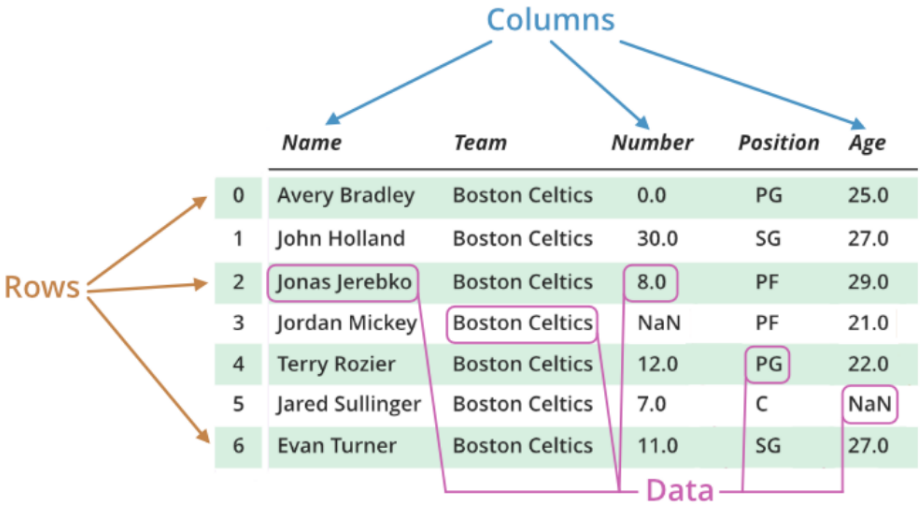

Figura 14 – Componentes de um *DataFrame*

Outra biblioteca é o NumPy (HARRIS *et al.*, 2020) (logo na Figura 15), que é um pacote de processamento de arrays com propósito geral, que fornece um objeto de matriz multidimensional de alto desempenho, e ferramentas para trabalhar com essas matrizes. Alguns recursos importantes são: um objeto poderoso de matriz n-dimensional, funções sofisticadas de transmissão, ferramentas de integração de código e capacidades úteis de álgebra linear, transformada de Fourier e números aleatórios.

Além disso o NumPy pode ser usado como um contêiner eficiente multidimensional de dados genéricos. Os tipos de dados arbitrários podem ser definidos usando o NumPy, o que permite que se integre sem esforço e rapidamente a uma ampla variedade de banco de dados.

Figura 15 – Logo do NumPy

E por fim, mais uma biblioteca, a scikit-learn (PEDREGOSA *et al.*, 2011) (logo na Figura 16), que fornece um uma variedade de algoritmos de *machine learning*. Ela inclui

algoritmos de classificação, regressão e agrupamento, possuindo SVM (*Support Vector Machine*), florestas aleatórias, *gradient boosting*, *KMeans* e DBSCAN. E ela é projetada para interagir com outras bibliotecas como o NumPy mencionado anteriormente.

Figura 16 – Logo da scikit-learn

O projeto ainda utilizou de outras bibliotecas como *Seaborn*, *matplotlib*, *pandas_profiling*, *Shapely*, *SciPy*, etc. Porém não serão detalhadas neste documento.

3.2.2.3 Git

Outra ferramenta utilizada foi o Git (TORVALDS; HAMANO, 2005), que é um sistema de controle de versão *open-source*, feito para registrar o histórico de alterações dos arquivos de projetos em desenvolvimento. Com esse controle, é possível fazer a visualização das mudanças, podendo submete-las a uma análise externa e reverte-las caso necessário. Isso faz com que a organização do conteúdo tenha uma qualidade melhor, e portanto tendo uma garantia maior de encontrar possíveis erros de lógica.

A plataforma do GitHub foi utilizada durante o projeto, onde foi criado um repositório central com a finalidade de manter a padronização do ambiente local para que fosse possível reproduzir o sistema em diferentes locais de trabalho.

Figura 17 – Logo do Git

3.3 ALGORITMOS E SISTEMAS DE RECOMENDAÇÕES

Nessa seção serão descritos os tipos de algoritmos de recomendações que foram estudados e considerados para aplicação durante a implementação do sistema, além de dar uma introdução de como um sistema de recomendação pode ser organizado.

Porém, antes de entrar especificamente em cada tipo de recomendação, é indispensável falar sobre de onde vêm os dados para conduzir os sistemas de recomendações e como eles são coletados, ou seja, quais são as entradas do sistema.

A fim de fazer uma recomendação de qualquer forma que seja, é necessário ter posse dos dados de entrada que darão a informação sobre quais itens os usuários provavelmente gostariam de consumir, ou seja, é preciso entender exatamente quais dados os recomendadores podem usar para aprender o que os usuários desejam, identificando quais tipos de dados são coletados do usuário e entendendo quando é possível e apropriado utilizar esses diferentes tipos de dados. Fazendo isso, será possível olhar para um sistema e identificar exatamente quais dados que o mesmo está usando para fornecer sua recomendação.

De forma ampla, o conceito que precisa ser entendido é o termo "Preferência". Esse termo é usado de maneira generalizada, já que ele pode ser usado em contextos diferentes. Por exemplo, para descobrir a preferência de um usuário, pode-se ter a informação de que o usuário gosta de filmes de ação ou qualquer outro gênero específico, ou também pode-se ter a informação de que os usuários do sistema acham que dois itens são comumente pareados juntos.

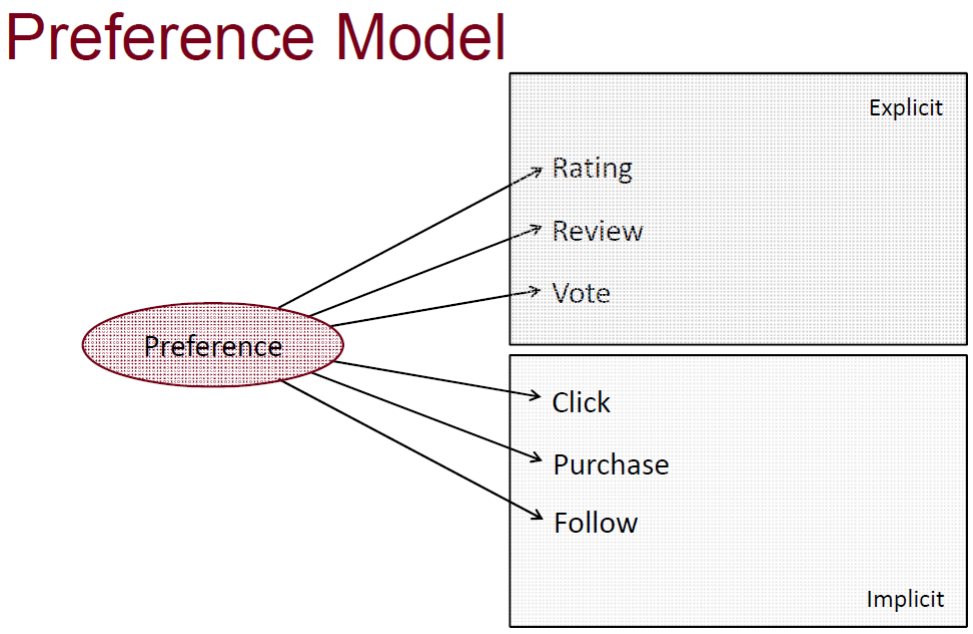

Figura 18 – Modelo de Preferência

Na Figura 18, observa-se o modelo de Preferência do usuário (KONSTAN; EKSTRAND, s.d.). Percebe-se que os usuários tem várias ações que podem ser realizadas, as quais dizem alguma informação sobre a sua preferência e que possuem algum tipo de conexão com o que o usuário deseja fazer, seja comprar algo, assistir, ler, etc. Essas ações são dividias em dois tipos, as explícitas e as implícitas.

Nas ações explícitas, o usuário tem a opção de fazer uma avaliação, onde ele está realizando uma ação com a finalidade de dizer ao sistema ou a outros usuários do sistema o que ele pensa sobre um determinado item. O usuário talvez avalie um item com uma nota 9 ou, dependendo do sistema de avaliação, com 4 estrelas por exemplo, para enfatizar que gostou. Outra ação que pode ser feita é uma revisão ou crítica sobre um item específico, incluindo textos com comentários mais detalhados e customizados. E ainda, ele pode simplesmente votar de forma ou negativa ou positiva, com intuito de demonstrar interesse ou não no item.

Nas ações implícitas, o usuário não irá performar uma ação a qual comunica diretamente sua preferência a um item, porém ainda é possível tirar conclusões sobre suas preferências ao analisar essas ações implícitas. Ações que podem ser realizadas incluem o usuário interagir ou clicar em algum item, fazer a aquisição dele, e/ou seguir um item específico.

Além disso, também é preciso entender quais são as saídas de um sistema de recomendações, e de quais maneiras as saídas do recomendador podem ser usadas. Para tanto, são levantadas as diferenças entre uma predição e uma recomendação.

A ideia de uma predição surge no momento em que um sistema de recomendações está calculando a estimativa de o quanto o usuário irá gostar de um item. Essa predição é frequentemente dimensionada para corresponder a alguma escala de avaliação e também é frequentemente vinculada à pesquisa ou navegação por produtos específicos. Por exemplo, o site IMDb, muito conhecido por trazer notas para filmes, séries, documentários, entre outros, mostra predições (Figura 19) do quanto os usuários em geral iriam gostar do conteúdo em específico.

Já recomendações não fazem as mesmas afirmações que uma predição faz. Recomendações são sugestões de itens que o usuário possa gostar ou que venham a ser úteis dentro do contexto no qual o usuário se encontra, e muitas das vezes são apresentadas em um formato de lista dos melhores itens. Por exemplo, é o que acontece no caso da recomendação por e-mail da Woliver (Figura 20), onde são elencados os quatro melhores imóveis de acordo com as características do usuário.

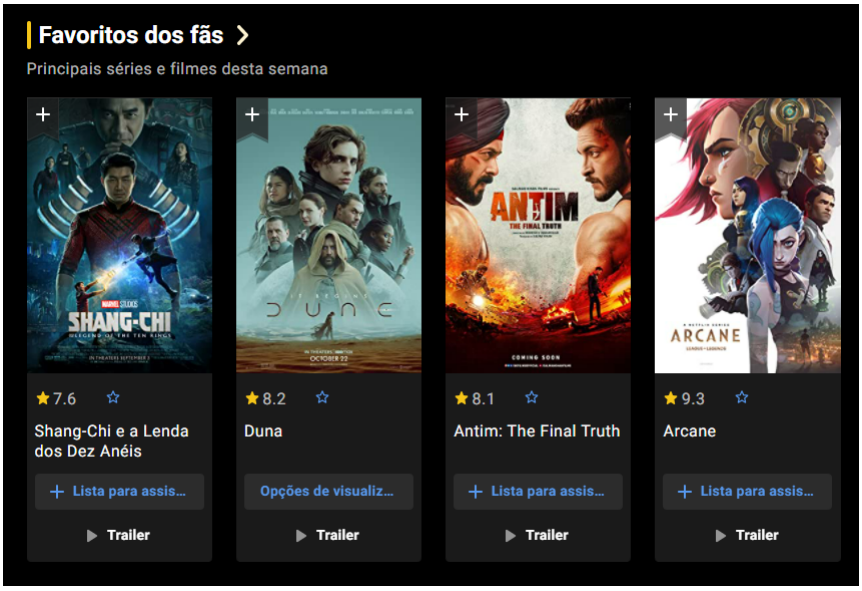

Figura 19 – Predições no site IMDb

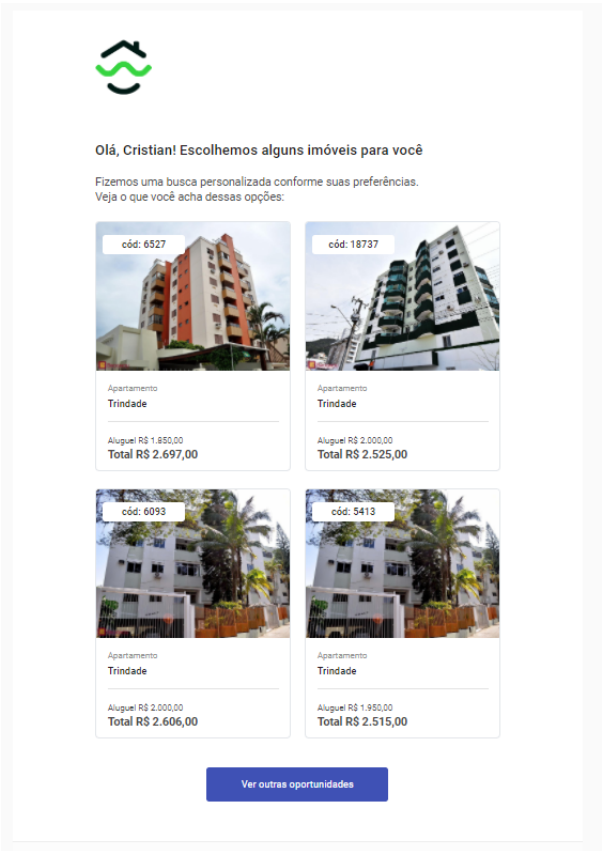

Figura 20 – Recomendação por e-mail da Woliver

3.3.1 Taxonomia de um Sistema de Recomendações

O objetivo dessa seção é trazer uma estrutura de um sistema de recomendações para analisá-lo de uma forma geral. Essa estrutura analítica pode ser abordada sendo dividida em oito dimensões de análise (KONSTAN; EKSTRAND, s.d.), que são as seguintes:

- **Domínio da Recomendação**: Nessa dimensão, é levantado a informação do que exatamente está sendo recomendado. Ex.: Notícias, produtos, filmes, livros, alimentos, músicas, etc. Também é possível diferenciar se o domínio está recomendando algo novo (e.g. filmes que nunca foram vistos pelo usuário) ou algo que já foi experimentado (e.g. músicas para escutar novamente).
- **Propósito da Recomendação**: Na segunda dimensão, define-se qual o propósito da recomendação, muitas das vezes é algo simples, como fazer com que o usuário compre um produto de uma loja, fazer com que assista um filme de um serviço de *streaming*, etc.
- **Contexto da Recomendação**: A terceira dimensão diz sobre o que o usuário está fazendo no momento da recomendação, como comprando, escutando uma música, saindo com outras pessoas, etc.
- **Opinião**: A quarta dimensão traz a informação da origem da opinião na qual o sistema de recomendação se baseia. Por exemplo, no site do *Rotten Tomatoes* que avalia filmes, há a opinião de críticos especializados em fazer esse tipo de avaliação e também há a opinião do público geral (Figura 21).

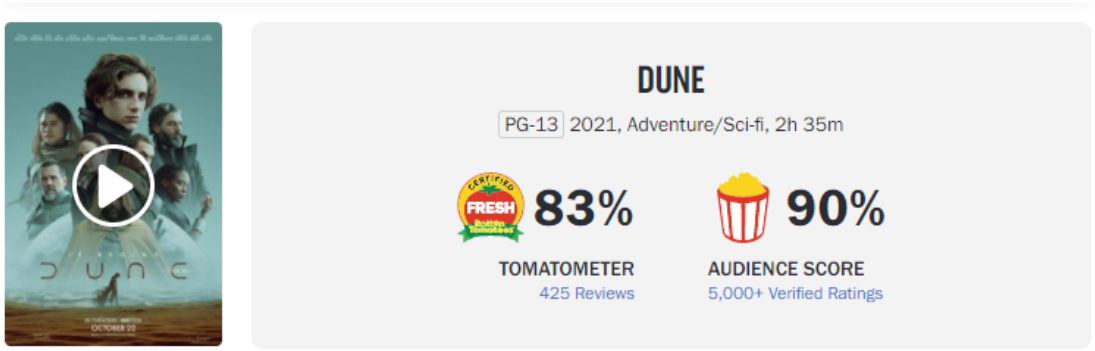

Figura 21 – *Rotten Tomatoes* - Opinião de críticos (*TOMATOMETER*) e da audiência

- **Nível de Personalização**: Nessa dimensão é analisado o nível de personalização, ou seja, indo de um nível genérico e não personalizado, onde todos os usuários recebem a mesma recomendação, passando por um nível semi personalizado, que por exemplo leva em conta a demografia dos usuários, onde

usuários menores de idade recebem uma recomendação diferente que usuário entre 18 e 25 anos, ou ainda, um nível de personalização efêmero, ou seja, algo que corresponde ao contexto e à atividade atual do usuário, e chegando em um nível personalizado com recomendações persistentes de longo prazo.

- **Privacidade e Confiabilidade**: A sexta dimensão leva em conta questões de privacidade e confiabilidade. Questões como quem precisa saber o que sobre o usuário, quais informações pessoais são reveladas, se é necessário saber quem é o usuário ou pode ser um usuário anônimo.
- **Interface**: Essa dimensão trata dos tipos de saídas, se são predições, recomendações, se está acontecendo algum tipo de filtragem e também se a apresentação da interface é orgânica, ou seja, natural para aquela interação, ou explícita, onde a recomendação vem de forma evidente, levantando a questão de como é estilo da interface. Por exemplo, se o sistema se apresenta como um agente ajudando o usuário, com algum diálogo ou está atuando mais parecido com uma ferramenta de procura. E também trata dos tipos de entradas mencionadas anteriormente, se são explícitas ou implícitas.
- **Algoritmos de Recomendações**: E na última dimensão, é categorizado quais algoritmos de recomendações se encaixam no sistema. Existem dois tipos principais: *Non-Personalized* ou não personalizado e *Personalized* ou personalizado (Figura 22). Esses algoritmos serão melhor detalhados nas seções em seguida.

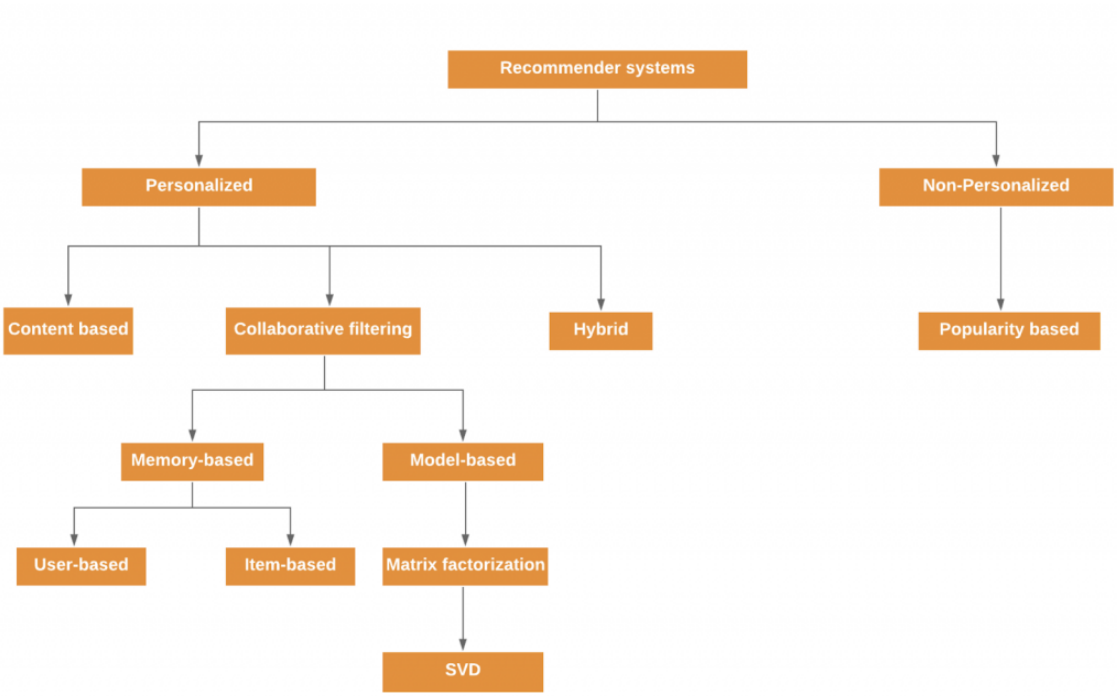

Figura 22 – Tipos de algoritmos de recomendações

3.3.2 Non-Personalized

Um sistema de recomendação não personalizado é o mais simples entre todos os tipos. Este tipo de recomendação não leva em consideração as preferências pessoais do usuário, com isso as recomendações produzidas por esse sistema serão idênticas para todos os clientes. (PORIYA *et al.*, 2014)

Esse sistema usa principalmente dois tipos de algoritmos baseados em popularidade: *Aggregated Opinion Recommender* e *Basic Product Association Recommender*.

3.3.2.1 Aggregated Opinion Recommender

A abordagem do *Aggregated Opinion Recommender* ou recomendador de opinião agregada utiliza um sistema não personalizado exibindo a média das avaliações dos usuários. Por exemplo, o site do IMDb citado anteriormente, possui essa abordagem, onde os filmes, as séries de TV, entre outros são recomendados pela nota média dos usuários. Outros exemplos, como guias de restaurantes, sites de viagens (e.g. TripAdvisor) ou sites de livros também utilizam essa abordagem.

As médias são úteis pois oferecem uma noção geral da opinião da população. No entanto, carecem de contexto durante a recomendação. Uma classificação ruim pode ter um peso muito grande e puxar para baixo uma avaliação que de outro modo seria excelente.

A fim de contornar isso, alguns sites controlam a porcentagem de pessoas que classificam um determinado item como bom ou ruim. Isso leva à abordagem do *Basic Product Association Recommender*.

3.3.2.2 Basic Product Association Recommender

Um recomendador por associação de produtos pode fornecer recomendações úteis dentro de um contexto. Uma grande quantidade de sites de compras online, como a *Amazon.com*, utilizam essa abordagem ao fornecer um recurso no qual revela que as pessoas que compraram um item1 e também compraram um item2, ou ainda as pessoas que procuraram um certo tópico, acabaram comprando um determinado item (Figura 23).

Essas recomendações são baseadas no que está presente no carrinho do usuário. Ou seja, as recomendações podem não ser necessariamente específicas para o usuário, mas específicas para o que o usuário está fazendo (e.g. visualizando, pesquisando, comprando, etc).

Figura 23 – Recomendador por associação - Amazon.com

Observando a taxonomia de um sistema de recomendação, nota-se que essa abordagem é efêmera, que depende do contexto. A ideia base é: pessoas que fizeram X, também fizeram Y. Uma fórmula simples que pode solucionar é a seguinte:

$$\frac{(X \cap Y)}{X} \tag{1}$$

Contudo, essa fórmula não compensa a popularidade geral de Y. Outra solução que contorna essa desvantagem é utilizar a métrica *lift*:

$$\frac{P(X \cap Y)}{P(X) * P(Y)} \tag{2}$$

A vantagem dos métodos não personalizados é que são simples de serem implementados, dado que somente os itens populares ou com classificação alta são exibidos, e ainda os dados dentro desses sistemas são fáceis de coletar.

Contudo, as recomendações são sempre as mesmas e, portanto, podem não agradar a todos. E ainda, esses sistemas enfrentam desafios ao serem utilizados em grupos de populações diversificadas, podendo entrar num problema conhecido como a armadilha da banana (*The Banana Trap*). Esse nome vem da situação onde em um supermercado, a maioria dos clientes compram bananas, porém se alguém compra uma banana e um barbeador por exemplo, não é possível afirmar que a compra de um influenciou na compra do outro e, por conseguinte, cai-se na armadilha da banana.

3.3.3 Personalized

Um sistema de recomendação personalizado faz a análise dos dados de usuários, suas compras, interações, avaliações e seus relacionamentos com outros usuários em mais detalhes. Dessa forma, é possível que todos os usuários tenham recomendações customizadas.

As principais abordagens de sistemas de recomendações personalizados são baseadas em conteúdo (*Content Based*) e filtragem colaborativa (*Collaborative Filtering*).

3.3.3.1 Content Based

Os sistemas de recomendações baseados em conteúdo usam metadados de usuários ou itens para criar recomendações específicas. O histórico de compras do usuário é observado, por exemplo, se um usuário já leu um livro de um autor ou comprou um produto de uma determinada marca, presume-se que o cliente tem uma preferência por aquele ator ou aquela marca e existe uma probabilidade maior de que o usuário compre um produto semelhante no futuro.

Em um outro cenário exemplo, supõe-se que um usuário assistiu e seja fã de um filme do Quentin Tarantino, *Pulp Fiction*, e o sistema saiba disso pela avaliação positiva do usuário. Há uma grande chance dos filmes que serão recomendados para este usuário específico estarem atrelados ao cineasta, como *Django Unchained* ou *Kill Bill*.

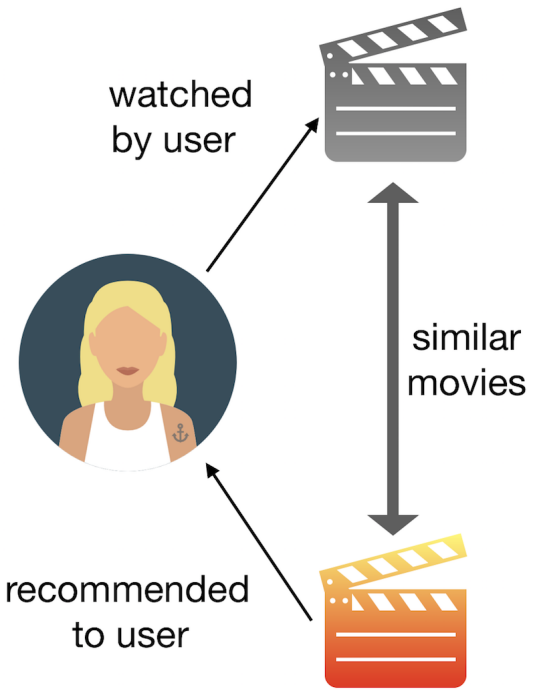

Figura 24 – Abordagem *Content Based*

Esse tipo de algoritmo utiliza duas amostras de dados. Primeiro, os gostos, preferências, interesses do usuário, as informações pessoais, como idade e gênero, ou até o histórico do usuário. Esses dados são representados pelo vetor do usuário. E segundo, informações relacionadas ao produto, as quais representam um vetor de itens e que contém as características entre itens as quais permitem calcular a similaridade deles.

As recomendações são calculadas usando a similaridade por cosseno, no final sendo ordenados de forma decrescente e os itens do topo são recomendados para o usuário. Com 'A' sendo o vetor do usuário e 'B', o vetor de itens:

$$\text{similaridade} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{3}$$

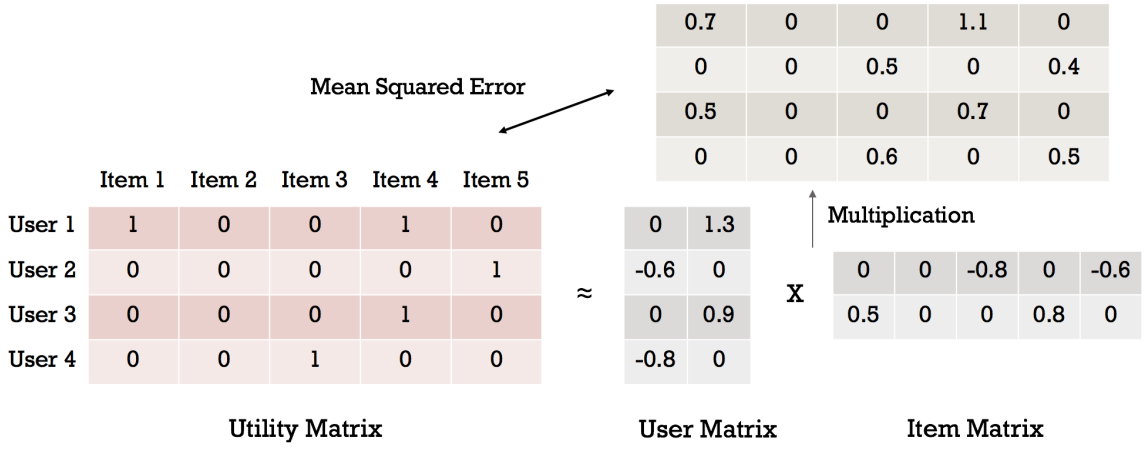

Figura 25 – Exemplo de cálculo de similaridade por cosseno

As desvantagens dessa abordagem são que o usuário não vai ter recomendações de itens diferentes, e é possível que entre em uma "bolha", onde caso o usuário não experimente itens diferentes por conta própria, as recomendações estarão restritas a interesses anteriores. Esse problema pode ser contornado utilizando o próximo tipo de recomendação, *Collaborative Filtering*.

3.3.3.2 Collaborative Filtering

As recomendações por filtragem colaborativa são feitas baseadas no comportamento dos usuários. E podem ser divididas em dois tipos: *Memory-based* e *Model-based*.

A técnica de *Memory-based* é aplicada a dados brutos sem pré-processamento. Ela é fácil de implementar e as recomendações resultantes são geralmente fáceis de explicar. Porém, toda vez é necessário fazer previsões sobre todos os dados, o que retarda o recomendador. É possível dividi-la nas duas categorias a seguir:

- ***User based***: Os produtos são recomendados ao usuário com base no fato de que foram comprados, consumidos, curtidos por usuários semelhantes ao usuário alvo (Figura 26).

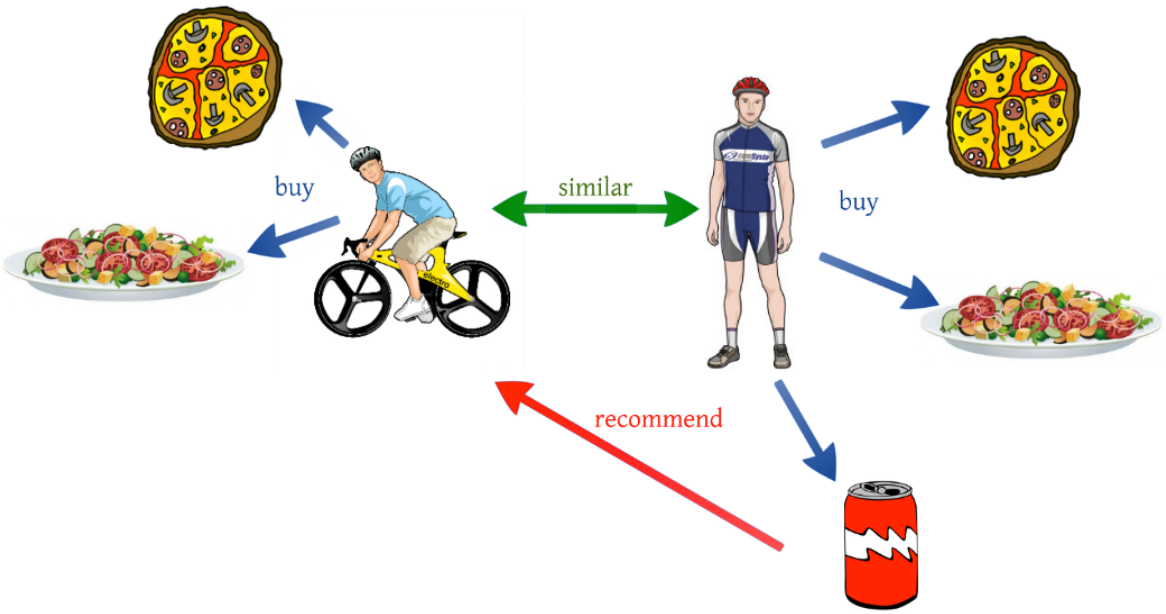

Figura 26 – Filtragem colaborativa baseada no usuário

- ***Item Based***: Nesse caso, em vez de considerar usuários semelhantes, itens semelhantes são considerados. Por exemplo, se alguns usuários compraram um livro, e também compraram mais outro em comum, caso o usuário alvo comprar o primeiro livro, ele irá receber como recomendação o segundo livro (Figura 27).

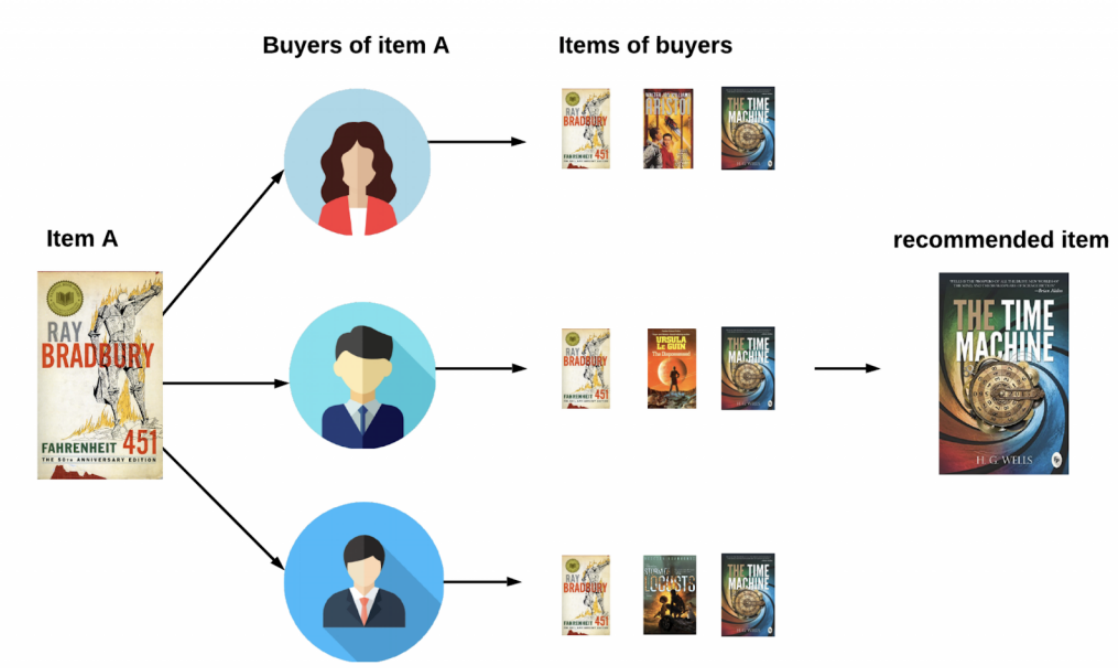

Figura 27 – Filtragem colaborativa baseada no item

Os passos para fazer uma recomendação baseado em memória são os seguintes:

1. Criar uma matriz de classificação usuário-item.
2. Criar uma matriz de similaridade usuário-usuário ou item-item utilizando o cálculo de similaridade por cosseno (Equação 3).
3. Procurar por usuários ou itens com maior semelhança ao alvo com o resultado da matriz de similaridade.
4. Gerar itens candidatos a serem recomendados, isto é, após encontrar os usuários ou itens mais semelhantes, são observadas as relações e preferências que cada usuário tem com um item.
5. Ranquear os candidatos. Dependendo das avaliações dos usuários, os candidatos a recomendação são ordenados de acordo com a similaridade do alvo.
6. Filtrar os candidatos. Após realizar a ordenação, é possível filtrar os itens candidatos de acordo com o que já foi experimentado pelo usuário alvo ou não.

Ao comparar as duas categorias de *Memory-based Collaborative Filtering*, tem-se que a similaridade entre itens é mais estável que a similaridade entre usuários, dado que as características de um item sempre serão as mesmas (e.g. um filme de ação sempre será um filme de ação), por outro lado os usuários podem mudar seus gostos

e preferências ao decorrer do tempo. Além disso, a abordagem baseada em itens é melhor para novos usuários recém inseridos no sistema, enquanto que a abordagem baseada no usuário não seria otimizada, pois não há informações suficientes sobre esses novos usuários.

Partindo agora para o *Model-based Collaborative Filtering*. Os algoritmos que se baseiam nesse modelo utilizam aprendizado de máquina. Um modelo é criado e a partir daí dá recomendações, o que agiliza o trabalho do sistema. Essa abordagem atinge uma escalabilidade melhor. A redução da dimensionalidade é frequentemente usada nesta abordagem. O tipo mais famoso dessa abordagem é a fatoração de matrizes.

Se houver feedback do usuário, por exemplo, um usuário assistiu a um determinado filme ou leu um livro específico e deu uma classificação, isso pode ser representado na forma de uma matriz onde cada linha caracteriza um usuário específico e cada coluna caracteriza um item particular. Como é quase impossível que o usuário avalie todos os itens, essa matriz terá muitos valores não preenchidos. Isso é chamado de esparsidade ou que configura uma matriz esparsa. Os métodos de fatoração de matriz são usados para encontrar um conjunto de fatores latentes e determinar as preferências do usuário usando esses fatores. As informações latentes podem ser relatadas analisando o comportamento do usuário.

Os passo para fatoração de matriz são os seguintes:

1. Inicialização de matriz de usuário e item aleatório.
2. Obtenção da matriz de classificação através da multiplicação entre matriz de usuário e matriz de item transposta.
3. Minimizar a função de perda (a diferença nas classificações das matrizes previstas e reais deve ser mínima). Cada classificação pode ser descrita como um produto escalar da linha na matriz do usuário e coluna na matriz do item.

E a fim de minimizar a função de perda, é possível aplicar algoritmos como SGD (*Stochastic Grandient Descent*) e ALS (*Alternating Least Squares*) (GRAHAM; MIN; WU, 2019). Ambos podem ser usados para atualizar o modelo de forma incremental conforme uma nova classificação chega.

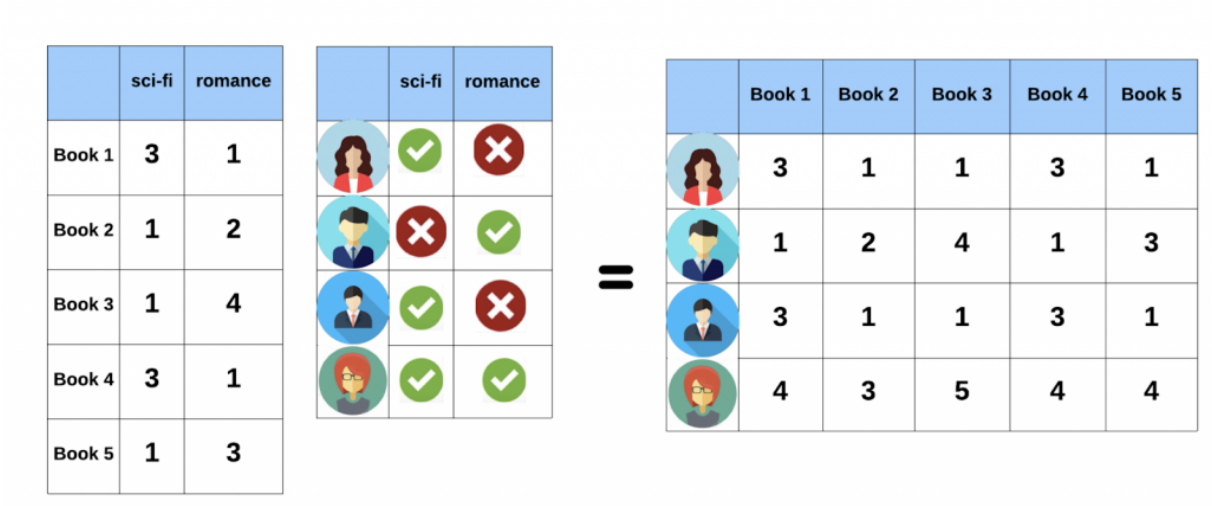

Figura 28 – Fatoração de matriz

3.3.3.3 Hybrid

Além disso existem os sistemas de recomendações híbridos. Hoje em dia, esse tipo consegue ser mais preciso em relação aos outros individualmente, pois leva em consideração o conjunto de fatores da filtragem colaborativa e da baseada em conteúdo, conseguindo ser mais eficiente e útil.

Há diversas abordagens que podem ser utilizadas para aplicar um sistema híbrido, é possível elencar algumas delas: *Weighted*, *Switching*, *Mixed*, *Feature Combination*, *Feature Augmentation*, *Cascade* e *Meta-Level* (BURKE, 2002). Este documento não entrará em detalhes sobre cada uma.

3.3.4 Avaliação e métricas

Outra parte muito importante que deve ser considerada, é como são feitas avaliações e quais métricas do sistema utilizar. Essa decisão depende muito do problema de negócios que está sendo resolvido. Também vale a pena salientar que nem sempre uma métrica ótima traduz em algo positivo na prática. O importante no final é que o usuário ganhe confiança sobre o sistema de recomendação. Mesmo podendo implementar o recomendador possível na teoria, o qual recomenda os 10 melhores itens elencados, há a possibilidade do usuário não gostar da maioria desses itens.

Dito isso, é importante traçar uma limiar de qualidade no recomendador para que não ocorra algo dessa grandeza.

Com base no documento do repositório da *Microsoft Recommenders* (GRAHAM; MIN; WU, 2019), é possível pontuar algumas técnicas de avaliações e métricas que podem ser usadas:

1. *Rating metrics*: São usados para avaliar o quão preciso é um recomendador na previsão de avaliações que os usuários deram aos itens.
    - RMSE (*Root Mean Squared Error*): Medida do erro médio nas avaliações previstas
    - R-squared (R^2): Quanto da variação total é explicado pelo modelo
    - MAE (*Mean absolute error*): Semelhante ao RMSE, mas usa o valor absoluto em vez de elevar ao quadrado e obter a raiz da média
    - *Explained Variance*: Quanto da variância nos dados é explicada pelo modelo
2. *Ranking metrics*: São usados para avaliar como as recomendações são relevantes para os usuários
    - *Precision*: Mede a proporção de itens recomendados que são relevantes
    - *Recall*: Mede a proporção de itens relevantes que são recomendados
    - NDCG (*Normalized Discounted Cumulative Gain*): Avalia o quão bem os itens previstos para um usuário são classificados com base na relevância
    - MAP (*Mean Average Precision*): Precisão média para cada usuário, normalizada para todos os usuários
3. Teste online A/B: Consiste em dividir a aplicação em duas versões, uma antiga e uma atualizada, com modificações na implementação do sistema, e compará-las.

4 WOLIVER E O CANAL DIGITAL

Este capítulo tem como objetivo introduzir o projeto e o contexto no qual o sistema de recomendação será aplicado, analisando como o projeto é visto de uma perspectiva de um usuário e também traçando as *pipelines* do sistema de recomendação como um todo.

É importante ressaltar que o autor deste documento esteve envolvido principalmente na parte de recomendação *offline* (Seção 4.2.2), fazendo a compreensão e extração dos dados e tendo maior foco no tratamento e preparação dos dados e na modelagem do sistema, aplicando algoritmos de recomendações do tipo *Collobora-tive Filtering* (ALS) e *Content Based* (clusterização com *KMeans*) com o propósito de integrar essas atividades dentro das *pipelines* do sistema de recomendações.

4.1 O PROJETO

Como já mencionado previamente no capítulo 2, a Jungle Devs trabalha juntamente com a Woliver no desenvolvimento de um SaaS (*Software as a Service*), o Canal Digital. Este oferece serviços de aluguel, agendamento de visitas e negociações de imóveis. Com tais serviços a plataforma digitaliza a experiência do usuário, conseguindo simplificar a burocracia no que se trata de operações imobiliárias.

Essa plataforma já existia antes da realização deste PFC, porém o sistema de recomendações era simplificado, não possuía grande complexidade e, portanto, tinha um grande potencial para melhorias. Dentro desse contexto que surgiu a ideia de desenvolver o projeto em cima desse tema.

A ideia do projeto é simples, fazer com que o usuário tenha a maior praticidade possível ao buscar por um imóvel. Na Figura 29 observa-se uma jornada simplificada com os processos envolvidos até o objetivo final de assinar o contrato do imóvel.

Figura 29 – Processos envolvidos

Um dos principais fluxos do Canal Digital é o agendamento de visitas, dado que

a maior parte da interação dos usuários se da nessa etapa, a qual acontece antes das etapas do cliente realizar uma proposta e fazer a análise de crédito.

No primeiro momento o usuário estará navegando pela página inicial da plataforma (Figura 30), ao acessá-la ele terá em sua disposição uma ferramenta de busca, onde é possível selecionar alguns campos: cidade, bairro, valor do aluguel e número de quartos do imóvel.

Figura 30 – Página inicial - Woliver

Após preencher os campos desejados, o usuário poderá visualizar os resultados dos imóveis similares a busca realizada (Figura 31).

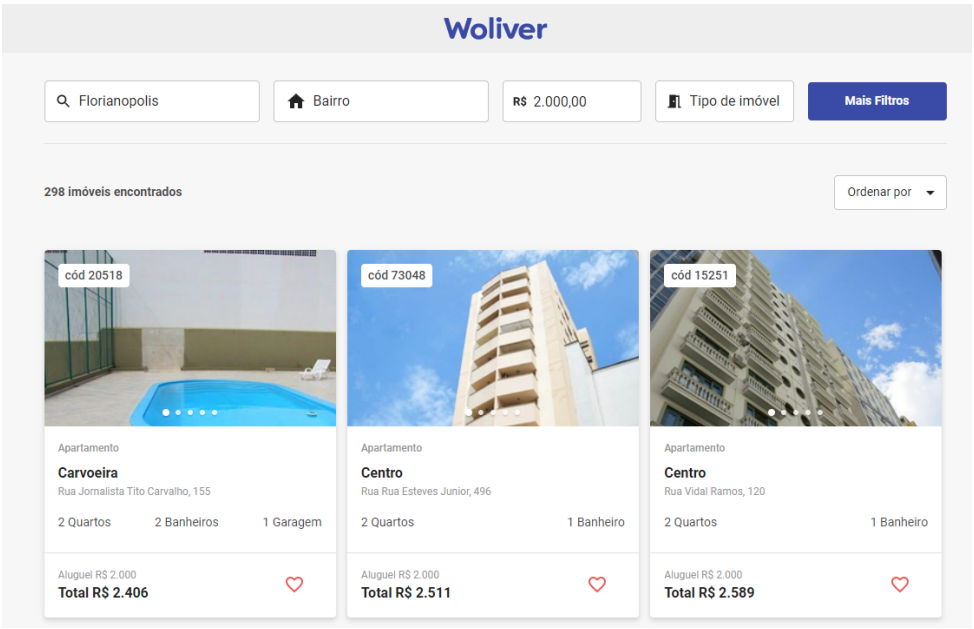

Figura 31 – Imóveis em exibição após realizada a busca

Ao clicar em um dos imóveis exibidos na busca, o usuário entrará na página específica do imóvel, a qual contém informações detalhadas sobre o mesmo (Figura 32).

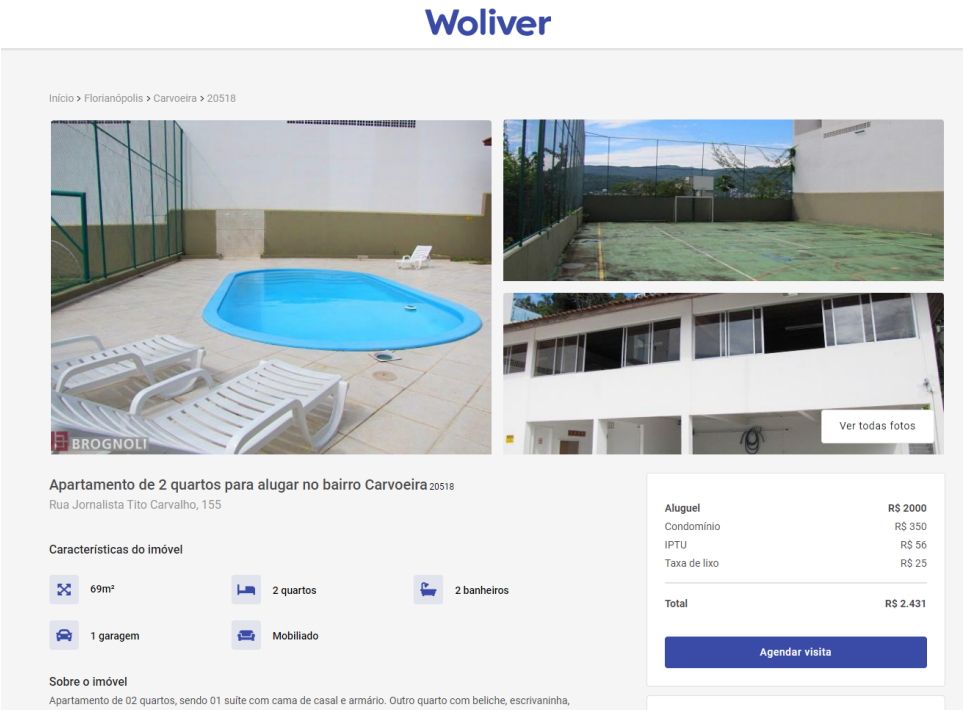

Figura 32 – Informações do imóvel específico

Na página do imóvel, o usuário terá a opção de agendar uma visita. Ao clicar nessa opção, ele será redirecionado a um formulário para agendamento de horário da visita, e em seguida será pedido para confirmar o horário marcado (Figura 33).

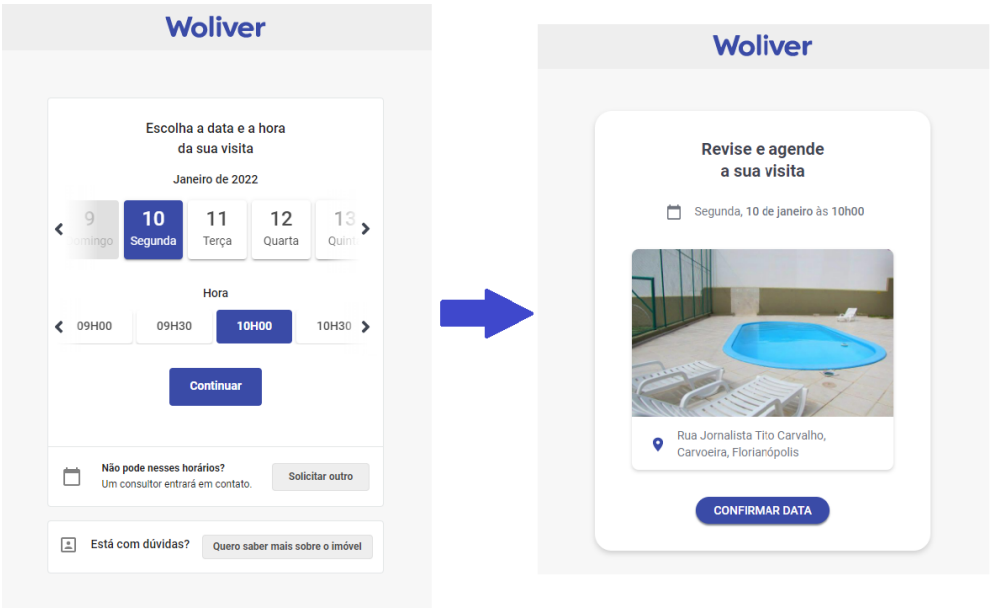

Figura 33 – Agendamento e confirmação de horário

Após a confirmação, será necessário que o usuário esteja logado, caso contrário ele será redirecionado ao cadastro do sistema, onde irá informar o celular, será enviado um código por SMS para verificação e então o mesmo deverá preencher um formulário com suas informações básicas (Figura 34).

Figura 34 – Cadastro do usuário

Feito o cadastro, o agendamento será realizado e as informações sobre a visita serão exibidas (Figura 35).

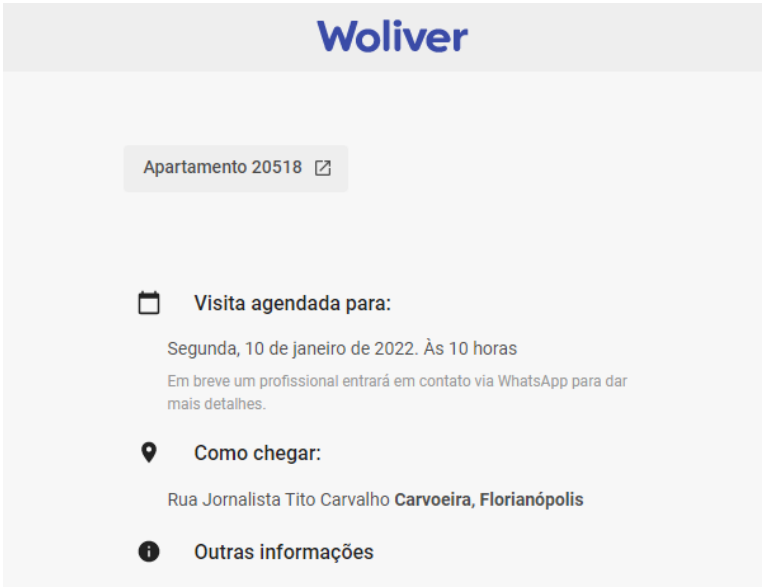

Figura 35 – Visita agendada

4.2 AS PIPELINES DO SISTEMA DE RECOMENDAÇÕES

Com o intuito de entender melhor a visão geral do sistema de recomendações que será utilizado no projeto, é possível enxergar o problema e separá-lo dentro de dois escopos. Um sendo a recomendação *online* e outro a recomendação *offline*.

Os escopos foram desenvolvidos pela equipe de *Data Science* da Jungle Devs com o auxílio de um consultor profissional especializado na área.

4.2.1 Recomendação *online*

A *pipeline* do processo *online* (Figura 36) pode ser dividido em sete etapas:

1. **Filtro de Busca**: O usuário irá fornecer dados de entrada (cidade, bairro, faixa de preço do aluguel, número de quartos, etc) (Figura 30).

2. **Consulta no Banco de Dados**: Faz-se uma comparação direta de todos os imóveis do banco de dados com os parâmetros do filtro selecionado. Por exemplo, se o usuário preenche no filtro o bairro Trindade e valor de aluguel 2000 reais, então será consultado imóvel por imóvel se essas características correspondem.

3. **Cálculo do *Score***: Calcula-se um *score* entre 0 e 1 que irá definir se o imóvel é um *exact match* ou não dependendo das características comparadas anteriormente, existindo um limiar (e.g. *score* maior que 0,8) que determina caso seja ou não. No final retornando uma lista de imóveis que atendem ao limiar.

4. **Cálculo dos centroides**: Filtrados os imóveis que estão na lista de *exact match*, é possível calcular o centroide de cada *cluster* dos imóveis existente. Essa clusterização dos imóveis é feita previamente em uma etapa do processo *offline*, que será comentada posteriormente, e assume-se nessa etapa que já é conhecido o *cluster* a qual um imóvel pertence.

5. **Cálculo da distância em relação ao centroide**: Sabendo os centroides de cada *cluster*, calcula-se a distância de todos os imóveis em relação ao centroide de seu respectivo *cluster* e a partir do valor dessa distância é feito um ranqueamento dos imóveis.

6. **Elencar melhores imóveis**: Com o ranking dos imóveis, é possível fazer a escolha dos melhores de forma ponderada, ou seja, dependendo da quantidade de imóveis presentes em cada *cluster*. Por exemplo, um *cluster* A representa 60% do total dos imóveis, um B 30% e um C 10%, então de uma lista de 10 imóveis, 6 pertencerão no *cluster* A, 3 no B e 1 no C.

7. **Reordenar usando ALS**: Por fim, com intuito de acrescentar um elemento de filtragem colaborativa, utiliza-se os *ratings* calculados com o algoritmo ALS para reordenar a lista dos imóveis.

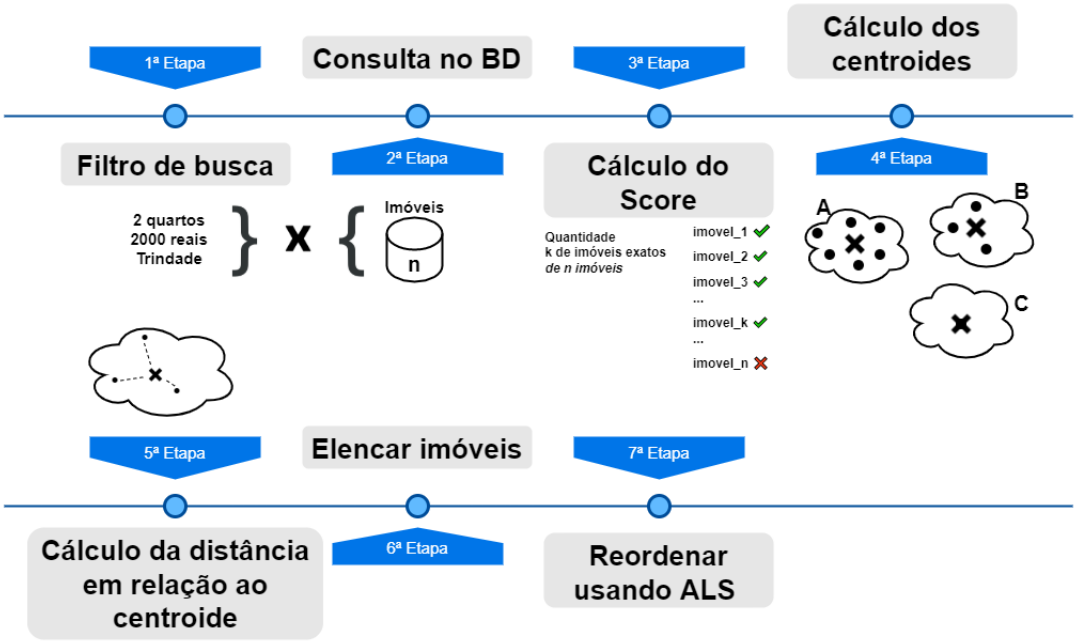

Figura 36 – *Pipeline* da recomendação *online*

4.2.2 Recomendação *offline*

A *pipeline* do processo *offline* pode ser dividida em cinco etapas:

1. **Extração de dados**: Nessa etapa é feita a extração dos dados que foram julgados úteis, esses dados podem ter informações dos imóveis, usuários e interações explícitas imóvel-usuário.

2. **Tratamento e preparação dos dados**: Nessa etapa uma série de processos, operações e transformações em cima dos dados são realizados a fim de preparar os dados da melhor forma possível para que possam ser usados nos algoritmos.

3. **Salvar dados tratados para consulta *online***: Os dados que foram tratados em um ambiente *offline* são salvos para serem usados dentro dos processos *online*.

4. **Treinamento/Clusterização**: Nessa etapa, o objetivo é fazer a divisão de todos os imóveis em uma quantidade de *clusters* que seja ideal para o processo, analisando as melhores maneiras de realizar clusterizações nas características dos imóveis.

5. **Salvar classes/agrupamentos gerados durante a clusterização**: Assim como os dados tratados são salvos para os processos *online*, as informações dos *clusters* gerados com todos os imóveis já classificados dentro de um dos *clusters* e a quantidade deles também são salvas.

5 DESENVOLVIMENTO

Este capítulo trata do desenvolvimento efetivo da aplicação, descrevendo os processos envolvidos para construção do projeto. Este desenvolvimento foi feito pelo autor, com a ajuda da equipe de *Data Science* da Jungle Devs. O capítulo está dividido em três partes principais: a compreensão dos dados, a preparação dos dados e a modelagem.

Antes de entrar nos detalhes das principais partes, deve-se recapitular o entendimento do problema que está sendo explorado. Para isso, é possível fazer uma caracterização do sistema de recomendações específico dentro do contexto desse trabalho utilizando a estrutura analítica especificada na seção 3.3.1:

- **Domínio da Recomendação**: Imóveis estão sendo recomendados.
- **Propósito da Recomendação**: Fazer com que o cliente se interesse por um imóvel específico, com a finalidade de ocorre algum tipo de transação.
- **Contexto da Recomendação**: O usuário estará pesquisando por imóveis.
- **Opinião**: A opinião vem dos usuários da plataforma baseada nos dados que são fornecidos por eles, e que serão destrinchados na seção de compreensão dos dados (Seção 5.1).
- **Nível de Personalização**: Personalizado, utilizando as informações de entrada de usuários específicos.
- **Privacidade e Confiabilidade**: A imobiliária terá informações específicas sobre os imóveis, informações básicas sobre o usuário e interações entre o imóvel e o usuário. Também mais detalhadas na seção 5.1.
- **Interface**: O sistema irá trazer recomendações explícitas de imóveis, exibindo uma lista de melhores imóveis para o usuário.
- **Algoritmos de Recomendações**: São utilizados algoritmos que se caracterizam no tipo personalizado, incluindo *Content Based* e *Collaborative Filtering*.

5.1 COMPREENSÃO DOS DADOS

Após entender todo o contexto do projeto, é possível partir para compreensão dos dados. Nessa etapa, deve-se definir, separar e organizar quais dados serão úteis para a aplicação.

Dessa forma, a partir das informações iniciais, os dados foram organizados nas seguintes tabelas: *Bookings*, *Disliked by*, *Liked by*, *Proposals*, *Users* e *Listings*. Essas

tabelas contêm informalões sobre imóveis, usuários e interações explícitas imóvel-usuário. E com as tabelas definidas, foram feitos dicionários de dados que possuem os metadados, ou seja, descrevem cada campo presente dentro de cada uma das tabelas.

5.1.1 Bookings

A tabela de *Bookings* possui informações sobre o agendamento de visitas. Os campos dessa tabela são: ID do agendamento de visita, ID do usuário que está realizando o agendamento, ID do imóvel que está recebendo a visita, o estado cuja visita se encontra, a data e hora marcada para a visita, a data e hora da criação do agendamento e a data e hora da última atualização do agendamento.

Na Tabela 1, observa-se o dicionário de dados contendo os metadados da tabela de *Bookings*.

Nome	**Tipo**	**Descrição**
id	integer	id do agendamento de visita
user_id	integer	id do usuário
listing_id	integer	id do imóvel
lease__status_booking	string	estado da visita
date_and_time	timestamp	data e hora marcada para a visita
created_at	timestamp	data e hora da criação do agendamento de visita
updated_at	timestamp	data e hora da atualização do agendamento de visita

Tabela 1 – Dicionário de dados da tabela de *Bookings*

Os estados que o agendamento da visita pode se encontrar são os seguintes:

- Agendamento marcado (*booking_scheduled*);
- Agendamento remarcado (*booking_rescheduled*);
- Esperando retorno (*booking_waiting_feedback*);
- Visita aconteceu (*booking_happened*);
- Visita não aconteceu (*booking_not_happened*);
- Problema relatado (*booking_issue_reported*);
- Visita cancelada (*booking_cancelled*);
- Desistiu do contrato (*booking_gave_up_listing*).

5.1.2 Disliked by e Liked by

A tabela de *Disliked by* tem como objetivo representar uma interação imóvel-usuário. Isso significa que, caso exista uma instância dessa tabela, um usuário não gostou de um imóvel.

E a tabela de *Liked by* tem o mesmo objetivo que a de *Disliked by*, porém com significado oposto, ou seja, representa que um usuário gostou de um imóvel.

As duas possuem os seguintes campos: ID da tabela ternária, ID do usuário e ID do imóvel. Isso faz com que o dicionário de dados (Tabela 2) das tabelas *Disliked by* e *Liked by* sejam exatamente iguais.

Nome	Tipo	Descrição
id	integer	id tabela ternária
user_id	integer	id do usuário
listing_id	integer	id do imóvel

Tabela 2 – Dicionário de dados das tabelas de *Disliked by* e *Liked by*

5.1.3 Proposals

A tabela de *Proposals* possui informações sobre as propostas de contratos de um usuário sobre um imóvel, se caracterizando como uma interação imóvel-usuário. Os campos da tabelas são: ID da proposta, ID do usuário, ID do imóvel, estado da proposta, data e hora da criação da proposta, e data e hora da última atualização da proposta.

Nome	Tipo	Descrição
id	integer	id da proposta no imóvel
user_id	integer	id do usuário
listing_id	integer	id do imóvel
lease__status_proposal	string	status da proposta
created_at	timestamp	data e hora da criação da proposta
updated_at	timestamp	data e hora da atualização da proposta

Tabela 3 – Dicionário de dados da tabela de *Proposals*

5.1.4 Users

A tabela de *Users* possui as informações básicas de um determinado usuário. Os campos da tabela são: ID do usuário, data e hora do registro, date e hora da última vez que o usuário entrou na plataforma, data de nascimento, gênero, cargo, Estado, cidade e bairro.

A Tabela mostra o dicionário de dados da tabela de *Usuários*.

Nome	Tipo	Descrição
id	integer	id do usuário
date_joined	timestamp	data e hora do registro
last_login	timestamp	data e hora do último login na plataforma
date_of_birth	string	data de nascimento
gender	string	gênero
role	string	client, staff, owner
state	string	estado
city	string	cidade
neighborhood	string	bairro

Tabela 4 – Dicionário de dados da tabela de *Users*

5.1.5 Listings

E por último, a tabela de *Listings* possui as informações específicas de um imóvel. Essa é a tabela mais complexa e que dispõe de muito mais campos do que o restante das tabelas vistas.

Nas Tabelas 5 e 6, é possível observar o dicionário de dados da tabela de *Listings*.

Nome	Tipo	Descrição
id	integer	id único da listagem utilizado como pk no banco de dados
created_at	datetime	Data em que a listagem foi adicionada ao sistema
updated_at	datetime	Data da ultima vez que a listagem foi alterada
external_id	string	id da listagem utilizado externamente
description	string	Descrição do imóvel (anúncio)
listing_type	enum string	Tipos de imóveis mapeados
bedrooms	integer	Número de quartos
bathrooms	integer	Número de banheiros
number_of_ensuites	integer	Número de suítes
parking_spots	integer	Número de vagas de gararem
living_area	float	Área útil em m2
total_area	float	Área total em m2
furnished	enum string	Descreve o "quão"mobiliado o imóvel está. Opções: not_furnished, semi_furnished, furnished
pets	Boolean	Indica se animais de estimação são permitidos
coordinates	string	PostGIS SRID 4326 POINT string exemplo: SRID=4326;POINT(-118.4079 33.9434)

Tabela 5 – Dicionário de dados da tabela de *Listings* - Parte 1

Nome	Tipo	Descrição
taxes	float	Valor de imposto (IPTU)
rent_price	float	Valor do aluguel em R$
condominium_fee	float	Valor das taxas de condomínio em R$
available	Boolean	Indica se o imóvel está disponível
condominium_description	string	Descrição do condomínio
insurance	float	Valor do seguro em R$
extra_address_info	string	Complemento do endereço
title	string	Título do imóvel/anúncio
sale_price	float	Valor de venda em R$
pictures	string array	Fotos da listagem (URLs)
video	string array	Vídeos da listagem (URLs)
street_number	string	Número na rua (endereço)
purpose_type	enum string	Propósito do imóvel. Opções: comercial, residential, mixed
transaction_type	enum string	Tipo de anúncio. Opções: rent, sale, rent_sale
origin_id	integer	id da fonte de dados (XML) do anúncio
address	string	Endereço
neighborhood_slug	string	Slug do bairo. Ex: joao_paulo
city_slug	string	Slug da cidade. Ex: sao_paulo

Tabela 6 – Dicionário de dados da tabela de *Listings* - Parte 2

5.2 PREPARAÇÃO DOS DADOS

Após feita a compreensão dos dados, é possível então partir para a preparação dos dados. Porém, antes de realmente começar a fazer as operações em cima dos dados, é necessário fazer a extração desses e ainda preparar o ambiente de desenvolvimento.

Para isso, utilizou-se uma função diretamente dentro da *codebase* da Woliver a fim de fazer o descarregamento dos dados. Essa função pega os dados das tabelas existentes do banco de dados da Woliver, transforma-os em arquivos CSV e descarrega-os no Amazon S3 (*Simple Storage Service*) (Figura 37), para que em seguida seja feito o download dos mesmos.

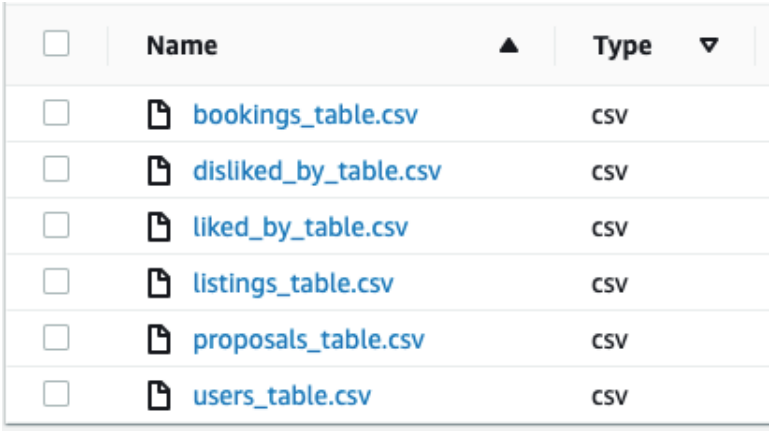

Figura 37 – Tabelas em CSV no Amazon S3

Após realizado o download, o ambiente de desenvolvimento foi organizado e padronizado utilizando um repositório central no GitHub (Figura 38). Esse repositório contém alguns arquivos de Jupyter Notebook, duas pastas, uma delas contendo os dados, tanto brutos como processados, e a outra contendo *profiling reports*, e ainda, arquivos auxiliares (*gitignore*, *README* e *requirements*) para ajudar na padronização do ambiente, possibilitando a reprodução em diferentes locais. Na sequência será detalhado o conteúdo de cada um dos arquivos.

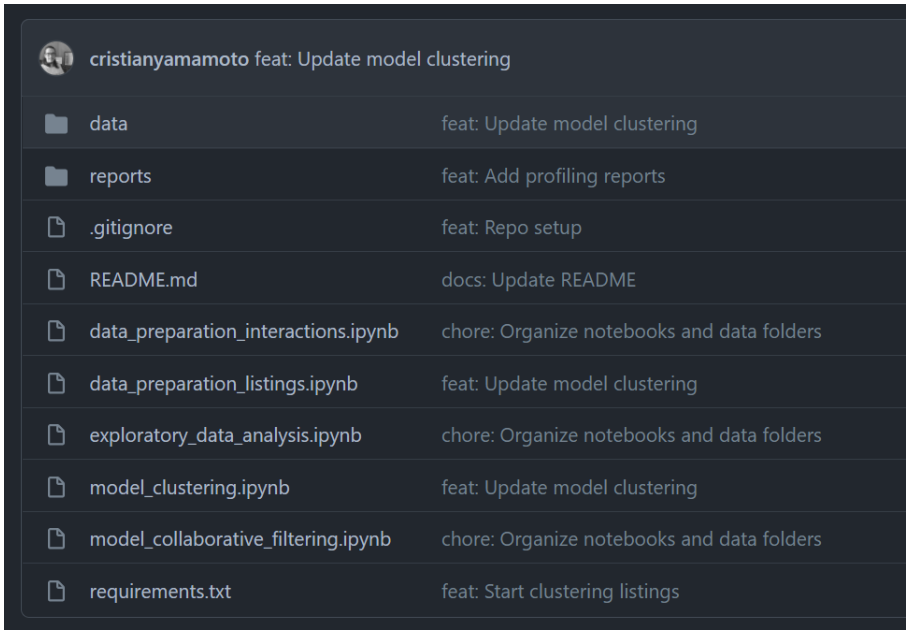

Figura 38 – Repositório no GitHub

5.2.1 Análise exploratória

No arquivo de Jupyter Notebook *exploratory_data_analysis.ipynb* estão sendo criados arquivos chamados de *profiling reports* os quais contêm relatórios de perfil, um para cada tabela de dados. Com esses relatórios, é possível fazer uma análise exploratória dos dados, com intuito de descobrir quais operações são as mais adequadas de fazer para cada campo.

Listings table

```
[50]: listings_df = pd.read_csv("data/raw/listings_table.csv", index_col=0)
      print(listings_df.shape)
      listings_df.head()
```

```
(5838, 34)
```

[50]:

	id	created_at	updated_at	external_id	description	listing_type	bedrooms	bathrooms	number_of_ensuites	parking_spots	living_area	total_area	furnished	pets	coordinates	rent_price	condomi
0	237	2019-07-25 19:16:48.614212+00:00	2021-07-28 12:03:28.501575+00:00	71234	Excelente de imóvel de 1 quarto, cozinha, sal...	res_kitnet	1.0	1.0	0.0	NaN	40.0	NaN	unfurnished	False	SRID=4326;POINT (-48.632 -27.55486)	650.0	
1	716	2019-08-27 06:25:02.094116+00:00	2021-07-08 11:31:55.188796+00:00	29868	Código do Imóvel Brognoli: 29868. Excelente ap...	res_apartment	2.0	2.0	1.0	1.0	72.0	NaN	semi_furnished	False	SRID=4326;POINT (-48.52014 -27.59326)	3200.0	
2	5465	2021-05-10 19:25:01.064968+00:00	2021-05-13 04:20:51.932031+00:00	814499	NaN	res_home	NaN	NaN	NaN	NaN	NaN	NaN	NaN	False	SRID=4326;POINT (-48.63564 -27.84699)	NaN	
3	1358	2019-11-08 04:22:59.282594+00:00	2021-07-28 12:03:29.571459+00:00	26476	Apartamento em ótima localização. 02 quartos, ...	res_apartment	2.0	2.0	0.0	1.0	65.0	82.0	furnished	False	SRID=4326;POINT (-48.52447 -27.58542)	1500.0	
4	2419	2020-04-15 01:49:51.162872+00:00	2021-03-31 07:21:42.655015+00:00	35623	Excelente casa localizada no Estreito. 03 dor...	res_home	3.0	2.0	2.0	4.0	90.0	90.0	semi_furnished	False	SRID=4326;POINT (-48.59641 -27.58929)	1750.0	

Figura 39 – Leitura do arquivo CSV da tabela de *Listings*

O principal relatório de perfil trata da tabela de imóveis. Esse relatório é gerado fazendo a leitura dos arquivos CSV de dados brutos (com *read_csv* da biblioteca do pandas) (Figura 39), e em seguida utilizando uma função da biblioteca *pandas_profiling* (BRUGMAN, 2019) que transforma os dados em um arquivo HTML (Figura 40).

```
[10]: # Generate profile report from listings data
      file_name = "./reports/listings-profile-report.html"
      pp.ProfileReport(listings_df).to_file(file_name)
```

Figura 40 – Gerando o relatório de perfil

Com o relatório de perfil gerado, cada campo da tabela pode ser analisado um por um e então concluir o processo específico que será aplicado no dado.

Na Figura 41, observa-se informações de alguns campos da tabela de *Listings*. Por exemplo, no campo *listing_type* que se refere ao tipo do imóvel, é possível observar que os imóveis registrados no banco de dados possuem cinco categorias diferentes: *res_apartment*, *res_home*, *res_kitnet*, *res_penthouse* e *res_sobrado*. Com essa informação, conclui-se que uma boa opção de tratamento desse dado é utilizar o método do *One Hot Encoding*.

Na próxima seção serão pontuadas as operações feitas em cada campo da tabela.

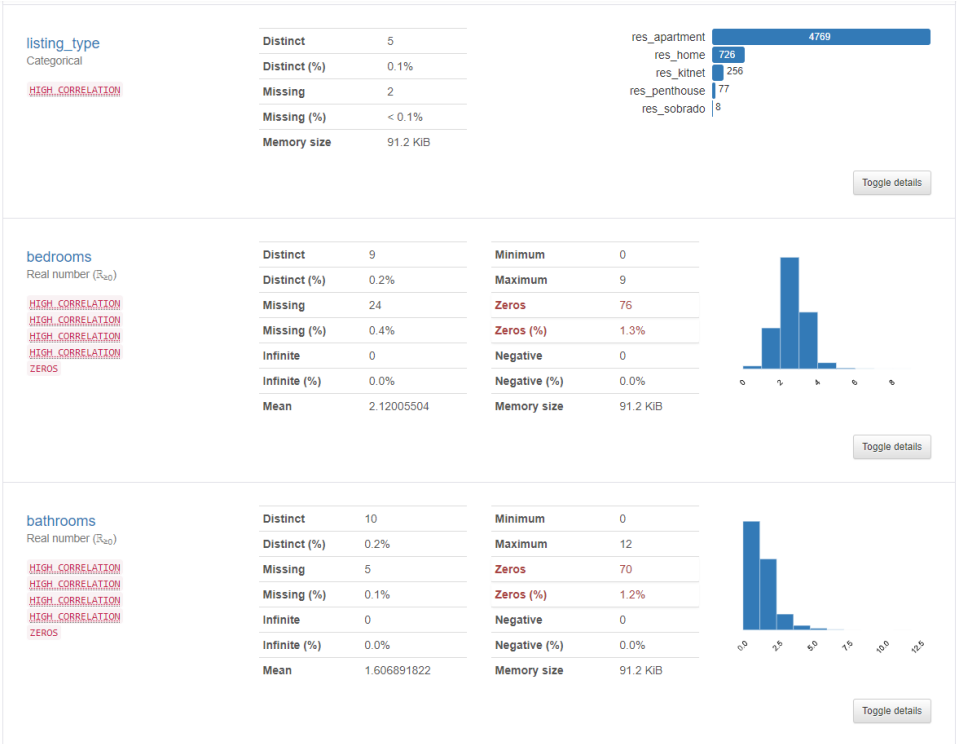

Figura 41 – Relatório de perfil exemplos

5.2.2 Processamento dos dados

Feita a análise individual de cada campo, agora é possível aplicar de fato todos os métodos de tratamento elencados (sendo de limpeza ou algum tipo de transformação) nos dados brutos através da programação do código.

A seguir apresenta-se a listagem dos métodos para cada item:

- id: remover
- *created_at*: remover;
- *updated_at*: remover;
- *external_id*: remover;
- *description*: remover;
- *listing_type*: one hot encoding;
- *bedrooms*: normalizar, transformar entrada nula (Np.nan=0 or 1);
- *bathrooms*: normalizar, transformar entrada nula (Np.nan=0);

- *number_of_ensuites*: normalizar, transformar entrada nula (Np.nan=0);
- *parking_spots*: normalizar, transformar entrada nula (Np.nan=0), remover *outliers*;
- *living_area*: normalizar, fundir com total_area, transformar entrada nula (mediana);
- *total_area*: normalizar, fundir com living_area, transformar entrada nula (mediana);
- *furnished*: mapear valores (0, 0.5, 1), entrada nula (Np.nan=0);
- *pets*: remover;
- *coordinates*: transformar com método personalizado, remover entrada nula;
- *rent_price*: normalizar, remover *outliers*, remover entrada nula;
- *condominium_fee*: normalizar, transformar entrada nula (mediana);
- *taxes*: normalizar, transformar entrada nula (mediana);
- *available*: remover;
- *listing_description*: remover;
- *condominium_description*: remover;
- *insurance*: remover;
- *extra_address_info*: remover;
- *title*: remover;
- *sale_price*: remover;
- *pictures*: transformar a contagem do número de imagens dentro da matriz;
- *video*: transformar para valores binários (0, 1);
- *street_number*: remover;
- *purpose_type*: remover;
- *transaction_type*: filtrar aluguel;
- *origin_id*: remover;
- *address*: remover;
- *neighborhood_slug*: remover entrada nula;

- *city_slug*: remover entrada nula, one hot enconding.

Esses processos estão sendo aplicados no arquivo de Jupyter Notebook *data_preparation_listings.ipynb*. E assim como na análise exploratória, primeiramente faz-se a leitura dos arquivos CSV de dados brutos (Figura 39).

Ao fazer a leitura, é instanciado um *Data Frame* (estrutura de dados do pandas) na variável *listings_df*. É a partir desse *Data Frame* que os dados serão processados. Na Figura 42 estão presentes alguns métodos utilizados para processar os dados.

O primeiro que aparece é o *parse_coordinates()*, esse método irá fazer a transformação do campo de coordenadas, dado que esse campo armazena um tipo *string* complexo de ser tratado (e.g. "SRID=4326;POINT (-48.632 -27.55486)"). Três novas colunas serão adicionadas na tabela: "lon", "lat"e "*coordinates_tuple*", as quais representam longitude, latitude e uma tupla da coordenada respectivamente.

Logo em seguida, há o método que fará a operação do *One Hot Encoding*. Esse método transforma as categorias de um campo em novas colunas com valores binários, que representam a presença ou não da categoria. Por exemplo no caso do *listing_type*, as cinco categorias existentes (*res_apartment*, *res_home*, *res_kitnet*, *res_penthouse* e *res_sobrado*) irão virar colunas do *Data Frame*.

```
BR_LAT_MIN = -29
BR_LAT_MAX = -25
# BR_LON_MIN = -73.9830625
BR_LON_MIN = -50
BR_LON_MAX = -45

def parse_coordinates(df: pd.DataFrame) -> pd.DataFrame:
    def load_coordinate(coord: str) -> str:
        if isinstance(coord, str):
            coordinate = loads(coord)
            if coordinate.x >= BR_LON_MIN and coordinate.x <= BR_LON_MAX and coordinate.y >= BR_LAT_MIN and coordinate.y <= BR_LAT_MAX:
                return coordinate
        return None
    df.coordinates = df.coordinates.str.replace('SRID=4326;', '').apply(load_coordinate)
    df['lon'] = df.coordinates.apply(lambda coord: coord.x if coord else np.nan)
    df['lat'] = df.coordinates.apply(lambda coord: coord.y if coord else np.nan)
    df['coordinates_tuple'] = df.coordinates.apply(lambda coord: (coord.x, coord.y) if coord else np.nan)
    return df

def one_hot_encode(df: pd.DataFrame, column) -> pd.DataFrame:
    one_hot = pd.get_dummies(df[column])
    df = df.drop(column, axis=1)
    df = df.join(one_hot)
    return df

def transform_video(df: pd.DataFrame) -> pd.DataFrame:
    df.video = df.video.notna()
    return df

def normalize_min_max(df: pd.DataFrame, column) -> pd.DataFrame:
    normalized_column = (df[column] - df[column].min()) / (df[column].max() - df[column].min())
    df[column] = normalized_column
    return df

def remove_outliers(df: pd.DataFrame, column) -> pd.DataFrame:
    z_scores = stats.zscore(df[column])
    abs_z_scores = np.abs(z_scores)
    return df.drop(df[(abs_z_scores > 3)].index)
```

Figura 42 – Métodos para tratamento dos dados - Parte 1

Depois, o método *transform_video* servirá na transformação do campo de *video*

para valores de verdadeiro ou falso, os quais serão mapeados por 0 ou 1 posteriormente.

Também estão presentes na Figura 42, o *normalize_min_max* e o *remove_outliers*, que fazem a normalização utilizando a abordagem *min-max*, a qual redimensiona o alcance dos valores para que fiquem entre 0 e 1, e a remoção dos *outliers* (valores atípicos) respectivamente.

No fragmento de código da Figura 43 está o restante dos métodos.

```
def transform_pictures(df: pd.DataFrame) -> pd.DataFrame:
    for i in df.loc[pd.notna(df.pictures)].pictures.index:
        num_pictures = df.pictures[i].count('\'') / 2
        df.loc[i, ['pictures']] = num_pictures
    return df

def normalize_box_cox(df: pd.DataFrame, column) -> pd.DataFrame:
    column_array = df[column].values
    column_array += 1
    fitted_data = stats.boxcox(column_array)[0]
    df[column] = fitted_data
    return df

def normalize_yeo_johnson(df: pd.DataFrame, column) -> pd.DataFrame:
    column_array = df[column].values
    fitted_data = stats.yeojohnson(column_array)[0]
    df[column] = fitted_data
    return df

def filter_rent_transaction_type(df: pd.DataFrame) -> pd.DataFrame:
    df = df[df.transaction_type == "rent"]
    return df

def merge_total_living_area(df: pd.DataFrame) -> pd.DataFrame:
    df['living_area'] = df['living_area'].combine_first(df['total_area'])
    return df

def fill_kitnet_bedrooms(df: pd.DataFrame) -> pd.DataFrame:
    df.loc[df["listing_type"] == "res_kitnet", "bedrooms"] = 1
    return df
```

Figura 43 – Métodos para tratamento dos dados - Parte 2

O *transform_pictures* irá transformar o campo de *pictures*, que originalmente representava uma lista com os URLs das fotos do imóvel, em um *integer* que representa a quantidade de fotos que o imóvel possui registrado.

O *normalize_box_cox* e o *normalize_yeo_johnson* servem para fazer a normalização com as abordagens *Box Cox* e *Yeo Johnson*, como o próprio nome indica.

O *filter_rent_transaction_type* irá filtrar todos os imóveis para que sejam apenas do tipo *rent* (aluguel).

O *merge_total_living_area* irá combinar os campos de *living_area* e *total_area* em apenas um campo.

E por último, o *fill_kitnet_bedrooms* transformará a quantidade de quartos em 1 para todos os imóveis que são do tipo *kitnet*.

Continuando no código, o *snippet* da Figura 44 lidará com o mapeamento e preenchimento dos valores nos campos em que essas operações são necessárias.

```
mapping = {
    'furnished': {
        'furnished': 1,
        'semi_furnished': 0.5,
        'unfurnished': 0,
        'unknown': 0,
    },
    'video': {
        True: 1,
        False: 0
    }
}

filling = {
    "bedrooms": 0,
    "bathrooms": 0,
    "number_of_ensuites": 0,
    "parking_spots": 0,
    "furnished": 0,
    "living_area": listings_df.living_area.median(),
    "condominium_fee": listings_df.condominium_fee.median(),
    "taxes": listings_df.taxes.median(),
    "pictures": 0
}
```

Figura 44 – Mapeamento e preenchimento de valores

E finalmente, presente na Figura 45, acontecerá o processamento de fato das operações, pegando o *Data Frame listings_df* que possui os dados brutos, aplicando todos os métodos mencionados em cada campo e armazenando em um novo *Data Frame* com os dados processados (*listings_df_processed*).

```
listings_df_processed = (
    listings_df
    .pipe(merge_total_living_area)
    .pipe(fill_kitnet_bedrooms)
    .pipe(filter_rent_transaction_type)
    .drop(columns=["id",
                   "created_at",
                   "updated_at",
                   "external_id",
                   "description",
                   "total_area",
                   "pets",
                   "available",
                   "listing_description",
                   "condominium_description",
                   "insurance",
                   "extra_address_info",
                   "title",
                   "sale_price",
                   "street_number",
                   "purpose_type",
                   "origin_id",
                   "address",
                  ]
          )
    .dropna(subset=["coordinates", "rent_price", "neighborhood_slug", "city_slug"])
    .fillna(value=filling)
    .pipe(parse_coordinates)
    .pipe(one_hot_encode, 'listing_type')
    .pipe(one_hot_encode, 'city_slug')
    .pipe(transform_video)
    .pipe(remove_outliers, 'parking_spots')
    .pipe(remove_outliers, 'rent_price')
    .pipe(normalize_yeo_johnson, 'bedrooms')
    .pipe(normalize_yeo_johnson, 'bathrooms')
    .pipe(normalize_yeo_johnson, 'number_of_ensuites')
    .pipe(normalize_yeo_johnson, 'parking_spots')
    .pipe(normalize_yeo_johnson, 'living_area')
    .pipe(normalize_yeo_johnson, 'rent_price')
    .pipe(normalize_yeo_johnson, 'condominium_fee')
    .pipe(normalize_yeo_johnson, 'taxes')
    .replace(mapping)
    .pipe(transform_pictures)
)
```

Figura 45 – Processamento da operações

Enfim, com o novo *Data Frame* contendo os dados processados, pode-se exportá-los em um documento CSV utilizando a função *to_csv()* (Figura 46).

Export processed listings to CSV ¶

```
[26]: listings_df_processed.to_csv("./data/processed/listings_features.csv")
```

Figura 46 – Exportando dados processado para .csv

5.3 MODELAGEM

Com os dados preparados, parte-se para a fase de modelagem. Nessa fase são aplicadas as técnicas de recomendações escolhidas para implementação no sistema desse trabalho.

O objetivo é completar a *pipeline* vista na seção 4.2, incluindo a abordagem de filtragem colaborativa utilizando o algoritmo ALS (*Alternating Least Square*) e a abordagem baseada em conteúdo utilizando algoritmos de clusterização.

5.3.1 ALS

```
[1]: import pandas as pd

[2]: listings_df = pd.read_csv("data/raw/listings_table.csv", index_col=0)
     users_df = pd.read_csv("data/raw/users_table.csv", index_col=0)
     users_df = users_df.drop(columns=['city', 'neighborhood', 'state'])
     users_df['user_id'] = users_df.id
     bookings_df = pd.read_csv("data/raw/bookings_table.csv", index_col=0)
     proposals_df = pd.read_csv("data/raw/proposals_table.csv", index_col=0)
     liked_by_df = pd.read_csv("data/raw/liked_by_table.csv", index_col=0)
     liked_by_df["liked"] = True
     disliked_by_df = pd.read_csv("data/raw/disliked_by_table.csv", index_col=0)
     disliked_by_df["disliked"] = True

[3]: # The goal: we want to have a big ass table that basically links:
     # user_id, listing_id, score

[4]: merged_df = liked_by_df.merge(bookings_df, "outer", on=["user_id", "listing_id"])
     merged_df = merged_df.drop(columns=["id", "date_and_time", "created_at", "updated_at"])
     merged_df = merged_df.merge(disliked_by_df, "outer", on=["user_id", "listing_id"])
     merged_df = merged_df.drop(columns=["id"])
     merged_df = merged_df.merge(proposals_df, "outer", on=["user_id", "listing_id"])
     merged_df = merged_df.drop(columns=["created_at", "updated_at"])
     merged_df = merged_df.merge(users_df, "left", on=["user_id"])
     merged_df = merged_df.drop(columns=["date_joined", "last_login", "date_of_birth", "gender"])
     merged_df = merged_df[merged_df["role"] == "client"]
     merged_df = merged_df.fillna(False)
     merged_df
```

[4]:

	listing_id	user_id	liked	lease_status_booking	disliked	lease_status_proposal	id	role
0	389	42	True	False	False	False	42	client
1	2093	119	True	False	False	False	119	client
2	346	119	True	False	False	False	119	client
3	1432	54	True	False	False	False	54	client
4	347	119	True	False	False	False	119	client
...	...	...	...	...	...	...	...	...
22773	3544	3304	False	False	True	False	3304	client
22774	1057	3950	False	False	True	False	3950	client
22775	4756	5868	False	False	True	False	5868	client
22776	1635	6104	False	False	True	False	6104	client
22777	913	1775	False	False	False	proposal_pending	1775	client

22054 rows × 8 columns

Figura 47 – Carregando os dados de interação imóvel-usuário

No *Notebook model_collaborative_filtering.ipynb* foi feita a análise do algoritmo ALS com os dados de interações, utilizando o modelo do algoritmo ALS encontrado no repositório da *Microsoft Recommenders* (GRAHAM; MIN; WU, 2019).

Inicialmente é feita a integração de todos os dados de interações, criando uma tabela com todas a relações (Figura 47).

A partir do momento em que é criada a tabela, é possível definir uma heurística com os dados de interação para montar uma pontuação para cada par imóvel-usuário.

[6]:
```
BOOKING_STATUS_SCORE = {
    "booking_scheduled": 1,
    "booking_rescheduled": 3,
    "booking_waiting_feedback": 3,
    "booking_happened": 3,
    "booking_not_happened": 1,
    "booking_issue_reported": -1,
    "booking_cancelled": 0,
    "booking_gave_up_listing": -1
}
PROPOSAL_STATUS_SCORE = {
    "proposal_pending": 5,
    "proposal_rejected": 3,
    "proposal_approved": 10,
    "proposal_cancelled": 3,
}

def monkey_score_function(row):
    monkey_score = 0
    if row['liked']:
        monkey_score += 2.5
    if row['disliked']:
        return -10  # Would we ever ponder people disliking things by accident?
    if row['lease__status_booking']:
        monkey_score += BOOKING_STATUS_SCORE.get(row['lease__status_booking'], 0)
    if row['lease__status_proposal']:
        monkey_score += PROPOSAL_STATUS_SCORE.get(row['lease__status_proposal'], 0)
    return monkey_score

merged_df['score'] = merged_df.apply(lambda row: monkey_score_function(row), axis=1)
merged_df = merged_df.sort_values(by=['score'], ascending=False)
merged_df
```

[6]:

	listing_id	user_id	liked	lease__status_booking	disliked	lease__status_proposal	id	role	score
11726	5271	7544	True	booking_waiting_feedback	False	proposal_pending	7544	client	10.5
11727	5271	7544	True	booking_waiting_feedback	False	proposal_pending	7544	client	10.5
6482	4028	3304	True	booking_waiting_feedback	False	proposal_pending	3304	client	10.5
6480	4297	3304	True	booking_waiting_feedback	False	proposal_pending	3304	client	10.5
6479	4016	3304	True	booking_waiting_feedback	False	proposal_pending	3304	client	10.5
...	...	...	...	...	...	...	...	...	...
22748	2002	751	False	False	True	False	751	client	-10.0
22749	531	751	False	False	True	False	751	client	-10.0
22750	2713	751	False	False	True	False	751	client	-10.0
22751	531	780	False	False	True	False	780	client	-10.0
22736	2394	632	False	False	True	False	632	client	-10.0

22054 rows × 9 columns

Figura 48 – Cálculo de pontuação utilizando heurística

Com isso constrói-se uma tabela inicial com usuário, imóvel e *score*, e é feito um treinamento dos dados para que possam ser usados no algoritmo ALS (Figura 49).

```
final_df = merged_df.drop(columns=['liked', 'disliked', 'lease__status_booking', 'lease__status_proposal', 'role', 'id'])
final_df = final_df.drop_duplicates()
final_df.score = final_df.score - 2
final_df
```

```
...

...

[Stage 0:>                                                          (0 + 1) / 1]
+-------+----------+-----+
|user_id|listing_id|score|
+-------+----------+-----+
|   5271|      7544|  8.5|
|   4028|      3304|  8.5|
|   4297|      3304|  8.5|
|   4016|      3304|  8.5|
|   4363|      3304|  8.5|
|   3957|      3304|  8.5|
|    409|      3304|  8.5|
|    945|      6783|  8.5|
|   5020|      6735|  8.5|
|   5020|      6984|  8.5|
|   4209|      3537|  8.5|
|   3924|      1365|  8.5|
|   1938|      7716|  8.5|
|   3895|      2108|  8.5|
|   4534|      7951|  8.5|
|   3805|      2083|  8.5|
|   5931|      8485|  8.5|
|   6175|      8927|  8.5|
|   4319|      6278|  8.5|
|   5102|      6453|  8.5|
+-------+----------+-----+
only showing top 20 rows
```

```
train, test = spark_random_split(data, ratio=0.75, seed=123)
print ("N train", train.cache().count())
print ("N test", test.cache().count())
```

```
N train 15841
N test 5221
```

Figura 49 – Treinamento dos dados

O algoritmo ALS tentará prever um *score* entre usuário e imóvel. Após a previsão, é possível utilizar métricas de *ranking* e *rating* para descobrir os resultados da aplicação do algoritmo (Figuras 50 e 51).

Observando os resultados, conclui-se que foram muito insatisfatórios, principalmente por causa da falta de dados que dizem a respeito as interações entre usuário e imóvel. Portanto, decidiu-se focar na implementação da clusterização.

Evaluate Rankings

```
[37]: als_rank_eval = SparkRankingEvaluation(test, top_all, k = TOP_K, col_user="user_id", col_item="listing_id",
                                             col_rating="score", col_prediction="prediction",
                                             relevancy_method="top_k")

print("Model:\tALS",
      "Top K:\t%d" % als_rank_eval.k,
      "MAP:\t%f" % als_rank_eval.map_at_k(),
      "NDCG:\t%f" % als_rank_eval.ndcg_at_k(),
      "Precision@K:\t%f" % als_rank_eval.precision_at_k(),
      "Recall@K:\t%f" % als_rank_eval.recall_at_k(), sep='\n')
```

```
[Stage 1789:=================================================> (194 + 6) / 200]
Model:  ALS
Top K:  4
MAP:    0.000462
NDCG:   0.000698
Precision@K:    0.000378
Recall@K:       0.001259
```

```
[38]: rand_rank_eval = SparkRankingEvaluation(test, top_all, k = TOP_K, col_user="user_id", col_item="listing_id",
                                              col_rating="score", col_prediction="random_prediction",
                                              relevancy_method="top_k")

print("Model:\tRandom",
      "Top K:\t%d" % rand_rank_eval.k,
      "MAP:\t%f" % rand_rank_eval.map_at_k(),
      "NDCG:\t%f" % rand_rank_eval.ndcg_at_k(),
      "Precision@K:\t%f" % rand_rank_eval.precision_at_k(),
      "Recall@K:\t%f" % rand_rank_eval.recall_at_k(), sep='\n')
```

```
[Stage 2001:=================================================>(197 + 3) / 200]
Model:  Random
Top K:  4
MAP:    0.000323
NDCG:   0.000637
Precision@K:    0.000504
Recall@K:       0.001238
```

Figura 50 – Métricas de *Ranking*

Evaluate Predictions

```
[39]: # Generate predicted ratings.
prediction = model.transform(test).withColumn('random_prediction', rand(seed=42) * 10)
prediction.cache().show()
```

```
•••
```

```
[40]: als_rating_eval = SparkRatingEvaluation(test, prediction, col_user="user_id", col_item="listing_id",
                                             col_rating="score", col_prediction="prediction")

print("Model:\tALS score prediction",
      "RMSE:\t%f" % als_rating_eval.rmse(),
      "MAE:\t%f" % als_rating_eval.mae(),
      "Explained variance:\t%f" % als_rating_eval.exp_var(),
      "R squared:\t%f" % als_rating_eval.rsquared(), sep='\n')
```

```
Model:  ALS score prediction
RMSE:   1.182853
MAE:    0.506151
Explained variance:     -0.096083
R squared:      -0.170122
```

```
[41]: rand_rating_eval = SparkRatingEvaluation(test, prediction, col_user="user_id", col_item="listing_id",
                                               col_rating="score", col_prediction="random_prediction")

print("Model:\tRandom score prediction",
      "RMSE:\t%f" % rand_rating_eval.rmse(),
      "MAE:\t%f" % rand_rating_eval.mae(),
      "Explained variance:\t%f" % rand_rating_eval.exp_var(),
      "R squared:\t%f" % rand_rating_eval.rsquared(), sep='\n')
```

```
Model:  Random score prediction
RMSE:   5.362870
MAE:    4.471864
Explained variance:     -7.125919
R squared:      -23.052725
```

Figura 51 – Métricas de *Rating*

5.3.2 Clusterização

Na análise da clusterização dos dados o objetivo é encontrar a quantidade de *clusters* ideal para implementação dentro da *pipeline* existente do sistema de recomendações.

Foram utilizados os dados do imóveis que haviam sido preparados anteriormente para fazer essa análise (Figura 52).

```
listings_df = pd.read_csv("data/processed/listings_features.csv", index_col=0)
listings_df = listings_df.dropna(subset=['lat', 'lon'])

print(listings_df.shape)
listings_df.head()
```

```
(5632, 27)
```

	bedrooms	bathrooms	number_of_ensuites	parking_spots	living_area	furnished	coordinates	rent_price	condominium_fee	taxes	pictures	video	transaction_type	neighborhood_slug	lon	lat	coordinates_tuple	res_apartm
0	0.905706	0.618088	-0.000000	0.000000	6.443372	0.0	POINT (-48.632 -27.55486)	4.508251	0.000000	3.831484	16.0	0	rent	areias	-48.63200	-27.55486	(-48.632, -27.55486)	
1	1.695602	0.918053	0.462824	0.762858	8.170753	0.5	POINT (-48.52014 -27.59326)	5.176381	30.535307	4.397604	33.0	0	rent	trindade	-48.52014	-27.59326	(-48.52014, -27.59326)	
3	1.695602	0.918053	-0.000000	0.762858	7.848910	1.0	POINT (-48.52447 -27.58542)	4.874523	39.750544	4.243316	32.0	0	rent	trindade	-48.52447	-27.58542	(-48.52447, -27.58542)	
4	2.418588	0.918053	0.595863	2.019676	8.906152	0.5	POINT (-48.59641 -27.58929)	4.938161	0.000000	3.532321	20.0	0	rent	coloninha	-48.59641	-27.58929	(-48.59641, -27.58929)	
6	0.905706	0.618088	-0.000000	0.000000	6.443372	1.0	POINT (-48.52883 -27.60265)	4.722728	0.000000	4.317624	12.0	0	rent	carvoeira	-48.52883	-27.60265	(-48.52883, -27.60265)	

Figura 52 – Leitura dos dados processados

Com esses dados, utilizou-se do algoritmo de clusterização *KMeans* para estudar as possíveis abordagens que podem ser tomadas.

K Means

Multiple layers of clusters

```
[22]: # Clustering by latitude and longitude
kmeans = KMeans(n_clusters=5, random_state=0).fit(listings_df[['lat', 'lon']])
listings_df['first_cluster'] = kmeans.labels_
print(listings_df.first_cluster.value_counts())
sns.scatterplot(x='lon', y='lat', data=listings_df, hue='first_cluster')
```

```
1    2549
4    1848
3     502
0     463
2     270
Name: first_cluster, dtype: int64
```

```
[22]: <AxesSubplot:xlabel='lon', ylabel='lat'>
```

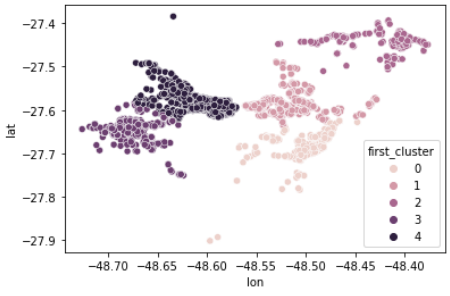

Figura 53 – Primeira clusterização de múltiplas camadas

Primeiro, foi feita uma análise da diferenciação entre utilizar múltiplas camadas de *clusters* ou somente uma camada. Na Figura 53, observa-se a aplicação da primeira clusterização, utilizando os campos de latitude e longitude para que fosse possível visualizar no gráfico.

A segunda clusterização foi aplicada com a finalidade de criar novos *clusters* a partir de um *cluster* existente aplicando o algoritmo no mesmo (Figura 54).

```
[23]: # Getting first cluster group
listings_df_first_clustering = (
    listings_df[listings_df['first_cluster'] == 0][['bedrooms', 'bathrooms', 'living_area', 'rent_price']]
    .dropna()
)
print(listings_df_first_clustering.shape)
listings_df_first_clustering.head()
```

```
(463, 4)
```

[23]:

	bedrooms	bathrooms	living_area	rent_price
23	0.905706	0.618088	5.442410	4.874523
28	1.695602	1.107330	9.800676	5.260207
89	1.695602	0.918053	9.163562	5.125141
119	1.695602	0.918053	9.127625	4.860370
122	0.905706	0.618088	6.090221	4.901308

```
[24]: # Clustering it again by bedrooms, bathrooms, living area and rent price
kmeans = KMeans(n_clusters=5, random_state=0).fit(listings_df_first_clustering)
listings_df_first_clustering['second_cluster'] = kmeans.labels_
print(listings_df_first_clustering.second_cluster.value_counts())
```

```
1    148
0    123
4     90
3     72
2     30
Name: second_cluster, dtype: int64
```

Figura 54 – Segunda clusterização de múltiplas camadas

A ideia de aplicar em múltiplas camadas acabou sendo descartada dado que não há evidências concretas de que haveria alguma vantagem em relação a apenas uma camada, e ainda essa abordagem acabaria gastando mais poder de processamento. Com isso, o foco foi analisar como fazer a aplicação para apenas uma camada.

No código da Figura 55, foi feita a clusterização para diferentes números de *clusters*, utilizando alguns campos julgados como mais relevantes para o usuário final. Observa-se pelo coeficiente de silhueta obtido que, nesse caso, cinco *clusters* teria vantagem sobre dez *clusters*.

O coeficiente de silhueta é uma métrica utilizada para calcular a eficácia de uma técnica de clusterização, o valor desse pode variar entre -1 e 1. Se o coeficiente tiver valor 1, significa que os *clusters* estão bem separados uns dos outros e claramente distintos. Com valor igual a 0, significa que eles são indiferentes, ou a distância entre os eles são insignificativas. E quanto mais perto de -1, significa que estão atribuídos erroneamente.

One layer of cluster

```
[52]: # Clustering by location, bedrooms, bathrooms, living area and rent price (5 clusters)

kmeans = KMeans(n_clusters=5, random_state=0).fit(
        listings_df[['lat', 'lon', 'bedrooms', 'bathrooms', 'living_area', 'rent_price']]
)
listings_df['cluster'] = kmeans.labels_
print(listings_df.cluster.value_counts())
```

```
4    2568
0    1845
1     498
2     451
3     270
Name: cluster, dtype: int64
```

```
[53]: silhouette_score(listings_df[['lat', 'lon', 'bedrooms', 'bathrooms', 'living_area', 'rent_price']], listings_df['cluster'])
```

```
[53]: 0.42259905437720136
```

```
[35]: # Clustering by location, bedrooms, bathrooms, living area and rent price (10 clusters)

kmeans = KMeans(n_clusters=10, random_state=0).fit(
    listings_df[['lat', 'lon', 'bedrooms', 'bathrooms', 'living_area', 'rent_price']]
)
listings_df['cluster'] = kmeans.labels_
print(listings_df.cluster.value_counts())
```

```
5    1426
0     779
8     777
4     773
9     623
7     519
2     406
1     217
6      85
3      27
Name: cluster, dtype: int64
```

```
[36]: silhouette_score(listings_df[['lat', 'lon', 'bedrooms', 'bathrooms', 'living_area', 'rent_price']], listings_df['cluster'])
```

```
[36]: 0.36843463726188763
```

Figura 55 – Clusterização com diferentes números de *clusters* (5 e 10)

A pontuação do coeficiente de silhueta é calculada da seguinte maneira:

$$\text{score} = \frac{B - A}{\max(A, B)} \tag{4}$$

Onde A é a distância média entre cada ponto dentro de um *cluster* e B a distância média entre todos os *clusters*.

Também foram feitas outras análises comparando a clusterização utilizando pesos nos campos e utilizando todos os campos possíveis para aplicação dos dados processados (Figuras 56 e 57 respectivamente).

```
[36]: silhouette_score(listings_df[['lat', 'lon', 'bedrooms', 'bathrooms', 'living_area', 'rent_price']], listings_df['cluster'])

[36]: 0.36843463726188763

[39]: # Clustering by location, bedrooms, bathrooms, living area and rent price (5 clusters) with weights

      listings_df_weighted = listings_df

      listings_df_weighted.lat = listings_df.lat * 50
      listings_df_weighted.lon = listings_df.lon * 50
      listings_df_weighted.bedrooms = listings_df.bedrooms * 10
      listings_df_weighted.bathrooms = listings_df.bathrooms * 5
      listings_df_weighted.living_area = listings_df.living_area * 2
      listings_df_weighted.rent_price = listings_df.rent_price * 20

      kmeans = KMeans(n_clusters=5, random_state=0).fit(listings_df_weighted[['lat', 'lon', 'bedrooms', 'bathrooms', 'living_area', 'rent_price']])
      listings_df_weighted['cluster'] = kmeans.labels_
      print(listings_df_weighted.cluster.value_counts())

      4    2568
      0    1845
      1     498
      2     451
      3     270
      Name: cluster, dtype: int64

[40]: silhouette_score(listings_df_weighted[['lat', 'lon', 'bedrooms', 'bathrooms', 'living_area', 'rent_price']], listings_df_weighted['cluster'])

[40]: 0.42259905437720136

[41]: # Clustering by location, bedrooms, bathrooms, living area and rent price (10 clusters) with weights

      kmeans = KMeans(n_clusters=10, random_state=0).fit(
          listings_df_weighted[['lat', 'lon', 'bedrooms', 'bathrooms', 'living_area', 'rent_price']]
      )
      listings_df_weighted['cluster'] = kmeans.labels_
      print(listings_df_weighted.cluster.value_counts())

      7    926
      3    770
      1    687
      9    642
      4    515
      2    503
      6    487
      5    432
      8    401
      0    269
      Name: cluster, dtype: int64

[42]: silhouette_score(listings_df_weighted[['lat', 'lon', 'bedrooms', 'bathrooms', 'living_area', 'rent_price']], listings_df_weighted['cluster'])

[42]: 0.29176346843196993
```

Figura 56 – Clusterização com pesos

```
[43]: # Clustering by all columns (5 clusters)

      listings_df_filtered = listings_df.drop(columns=["coordinates", "coordinates_tuple", "transaction_type", "neighborhood_slug"])

      kmeans = KMeans(n_clusters=5, random_state=0).fit(listings_df_filtered)
      listings_df_filtered['cluster'] = kmeans.labels_
      print(listings_df_filtered.cluster.value_counts())

      1    2567
      3    1846
      4     498
      0     451
      2     270
      Name: cluster, dtype: int64

[44]: silhouette_score(listings_df_filtered, listings_df_filtered['cluster'])

[44]: 0.417198862442935

[54]: # Clustering by all columns (10 clusters)

      kmeans = KMeans(n_clusters=10, random_state=0).fit(listings_df_filtered)
      listings_df_filtered['cluster'] = kmeans.labels_
      print(listings_df_filtered.cluster.value_counts())

      4    1275
      8     782
      1     705
      2     680
      5     593
      6     436
      9     386
      3     269
      0     262
      7     244
      Name: cluster, dtype: int64

[55]: silhouette_score(listings_df_filtered, listings_df_filtered['cluster'])

[55]: 0.2833328340503023
```

Figura 57 – Clusterização com todos os dados disponíveis

E por último, foi feita uma análise visual dos diferentes resultados obtidos quando a quantidade de *clusters* é variada, fazendo uma comparação do coeficiente de silhueta.

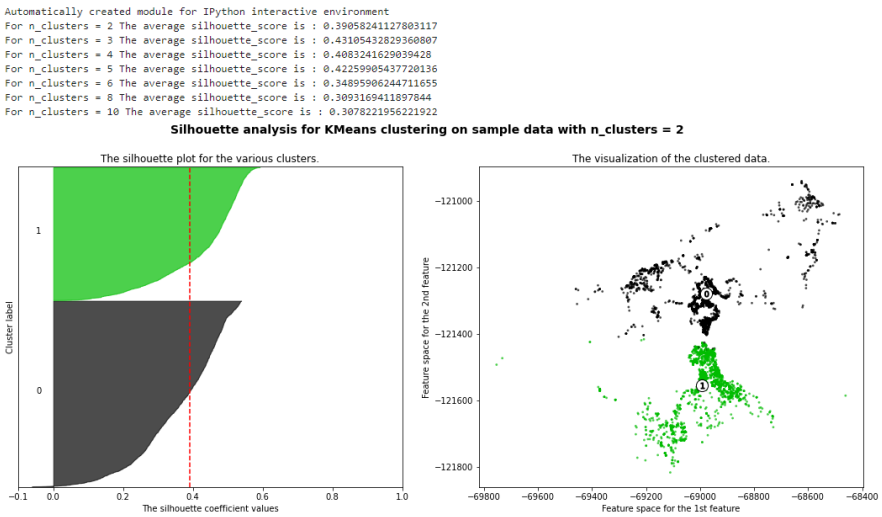

Figura 58 – Coeficientes de silhueta e visualização para n_clusters = 2

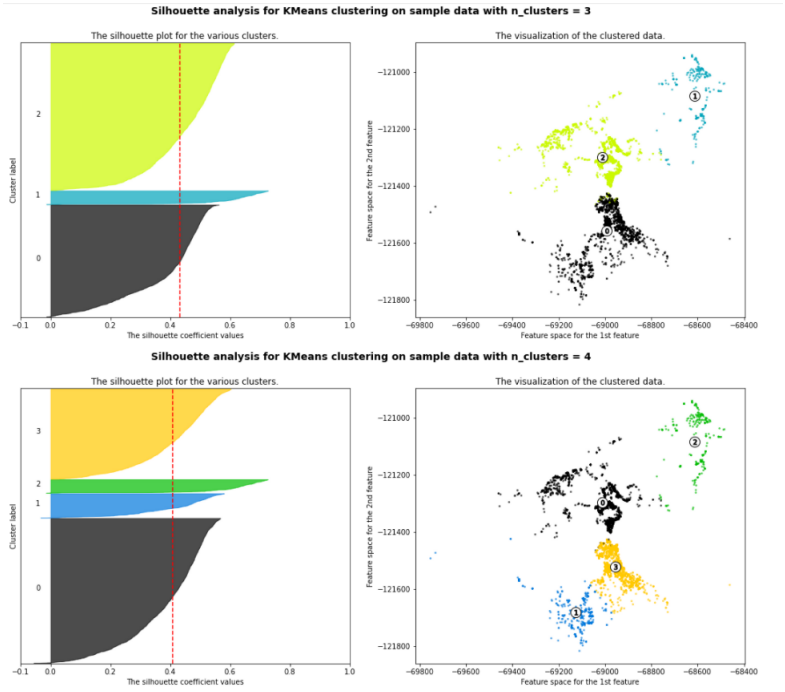

Figura 59 – Visualização para n_clusters = 3 e n_clusters = 4

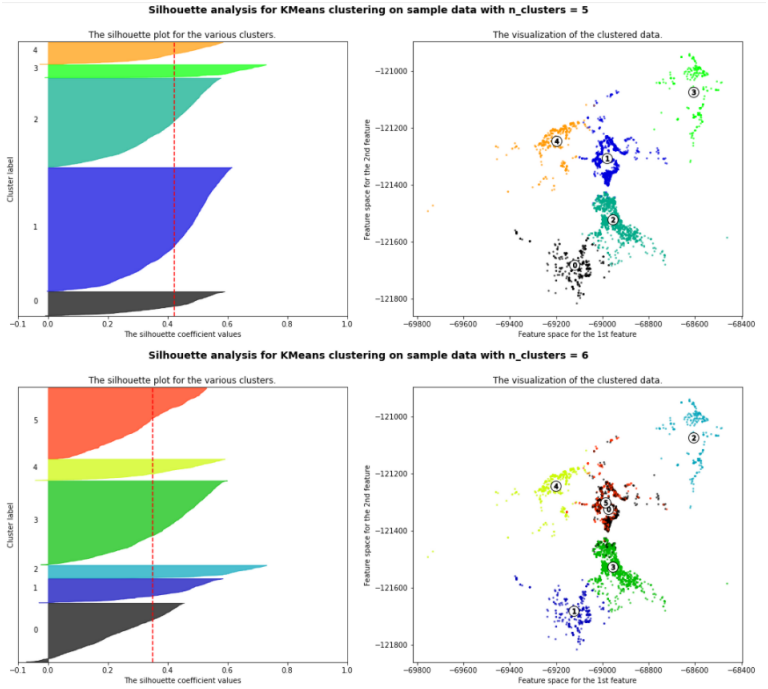

Figura 60 – Visualização para n_clusters = 5 e n_clusters = 6

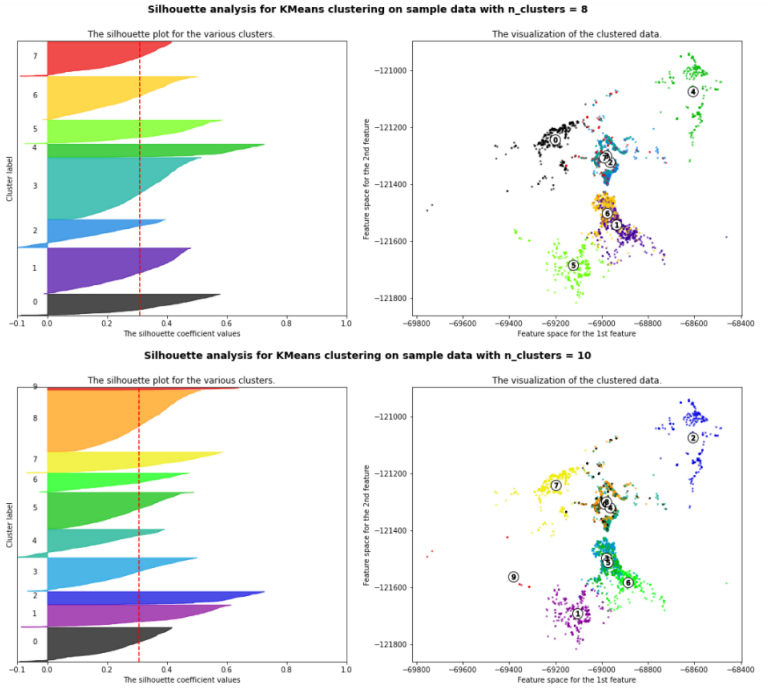

Figura 61 – Visualização para n_clusters = 8 e n_clusters = 10

Salienta-se ainda que, no sistema atual, a implementação do parâmetro de número de *clusters* está sendo feita manualmente. Isso pode ser empecilho para o sistema, visto que o número de *clusters* pode variar de acordo com vários fatores, como região da imobiliária.

6 RESULTADOS

Neste capítulo serão apresentados os resultados obtidos durante o desenvolvimento deste trabalho, analisando as respostas da implementação por *Collaborative Filtering* utilizando o ALS, e também as repostas da implementação da clusterização no sistema. Serão apresentadas algumas métricas e comparações do sistema de recomendações antes e depois da implementação das atividades do projeto.

6.1 FILTRAGEM COLABORATIVA

Primeiro, é importante citar que a modelagem do algoritmo ALS, que trata da parte de recomendações por filtragem colaborativa, não trouxe bons resultados. As métricas de precisão ficaram insatisfatórias como visto na seção 5.3.1 e deixaram a desejar. A hipótese do principal fator que levou a esse desempenho ruim é a falta de dados de interação.

Atualmente, tem-se acesso apenas às interações explícitas e com pouco volume, como as interações de *Liked by* e *Disliked by*, e portanto, seria necessário mais dados deste tipo, como cliques do usuário enquanto faz uma busca, tempo de visualização na página de um imóvel específico, tempo que gastou visualizando as fotos do imóvel, entre outros.

Dito isso, os resultados mais importantes e positivos que foram frutos deste projeto estão relacionados a *pipeline* genérica projetada do sistema de recomendações como um todo (Seção 4.2), pois com as atividades do projeto foi possível fazer a integração dessa *pipeline*. Daqui consegue-se tirar vários resultados qualitativos.

6.2 RESULTADOS QUALITATIVOS

Um resultado muito importante desse projeto foi a abertura de novas possibilidades de recomendações. Ou seja, antes só era possível fazer uma recomendação com filtros, ou seja fazendo uma busca colocando alguns filtros como entrada, como cidade, bairro, faixa de preço, etc, e tendo como resposta uma lista de imóveis com tais características. Hoje em dia, já é possível fazer outras recomendações, como recomendar baseado em um imóvel específico (Figura 62), baseado em imóveis favoritados, baseado em imóveis visualizados recentemente, baseados em imóveis mais populares, entre outras possibilidades.

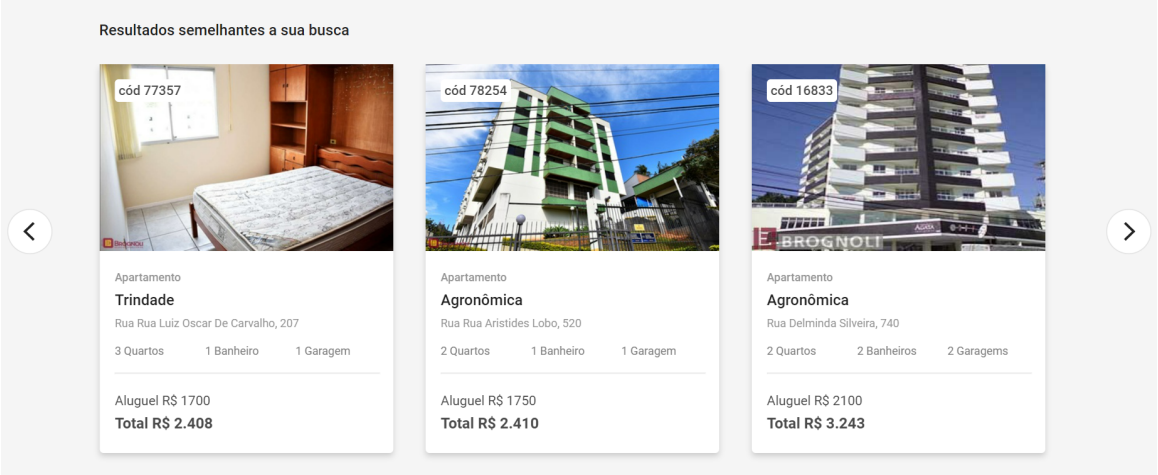

Figura 62 – Imóveis parecidos com o imóvel específico sendo visitado pelo usuário

Também relacionado a esse aspecto de novas possibilidades, é importante mencionar que agora o sistema de recomendação atende os casos que abrangem recomendação por filtro, por um imóvel e por n imóveis.

6.3 CLUSTERIZAÇÃO

Por último, foi possível fazer uma análise mais quantitativa fazendo um experimento com o agendamento de imóveis passados e observando os resultados da adição da clusterização na *pipeline* do sistema ao comparar com o sistema antigo.

O experimento teve a finalidade de validar as recomendações utilizando as métricas de *Ranking Precision* e *MAP* (Seção 3.3.4) e de cobertura.

Esse experimento utilizou os dados de 1 imobiliária, 574 imóveis, 179 agendamentos e 93 usuários com filtros de busca salvos.

Os dados de agendamentos são usados como gabarito para o que se considera uma boa recomendação. Ou seja, se dentre os imóveis recomendados, existe um imóvel com agendamento de visita, considera-se que a recomendação foi bem sucedida. E para cada usuário, fez-se recomendações usando ambos algoritmos antigo e novo.

As métricas de precisão (*precision_at_k* e *map_at_k*) são calculadas com um k=4, simulando uma recomendação que aconteceria através de um e-mail, onde, atualmente são enviados 4 imóveis para o usuário. E a métrica de cobertura é simplesmente a proporção de imóveis únicos recomendados dos 574 imóveis disponíveis da imobiliária.

Na Figura 63, observa-se como foi feito para chegar nos resultados. E nas tabelas 7 e 8, são exibidos os resultados do experimento.

```
[28]: precision = precision_at_k(
          rating_true=rating_true,
          rating_pred=rating_pred,
          col_user="user",
          col_item="listing",
          col_rating="booked",
          col_prediction="rec_simple",
          relevancy_method="top_k",
          k=4,
      )
      map_ = map_at_k(
          rating_true=rating_true,
          rating_pred=rating_pred,
          col_user="user",
          col_item="listing",
          col_rating="booked",
          col_prediction="rec_simple",
          relevancy_method="top_k",
          k=4,
      )
      print("Old Algorithm")
      print("Ranking Metrics")
      print(f"precision_at_k: {precision}")
      print(f"map_at_k: {map_}")
      print("\nNon accuracy based metrics")
      print(f"coverage: {rating_pred['listing'].nunique() / 574}")

      Old Algorithm
      Ranking Metrics
      precision_at_k: 0.05747126436781609
      map_at_k: 0.0803639846743295

      Non accuracy based metrics
      coverage: 0.2787456445993031
```

Figura 63 – *Snippet* do experimento realizado

Métricas de Ranking	**Algoritmo Antigo**	**Algoritmo Novo**
precision_at_k	0.05747126436781609	0.10674157303370786
map_at_k	0.0803639846743295	0.20168539325842696

Tabela 7 – Resultado das métricas de *Ranking*

Métrica de cobertura	**Algoritmo Antigo**	**Algoritmo Novo**
coverage	0.2787456445993031	0.3693379790940767

Tabela 8 – Resultado da métrica de cobertura

Outro resultado também interessante de citar foi a quantidade de envio por e-mail de recomendações realizadas. Foram enviados 111.413 e-mails durante o mês de Novembro de 2021 entre 15 imobiliárias, isso dá aproximadamente 3713 recomendações por dia (Figura 64).

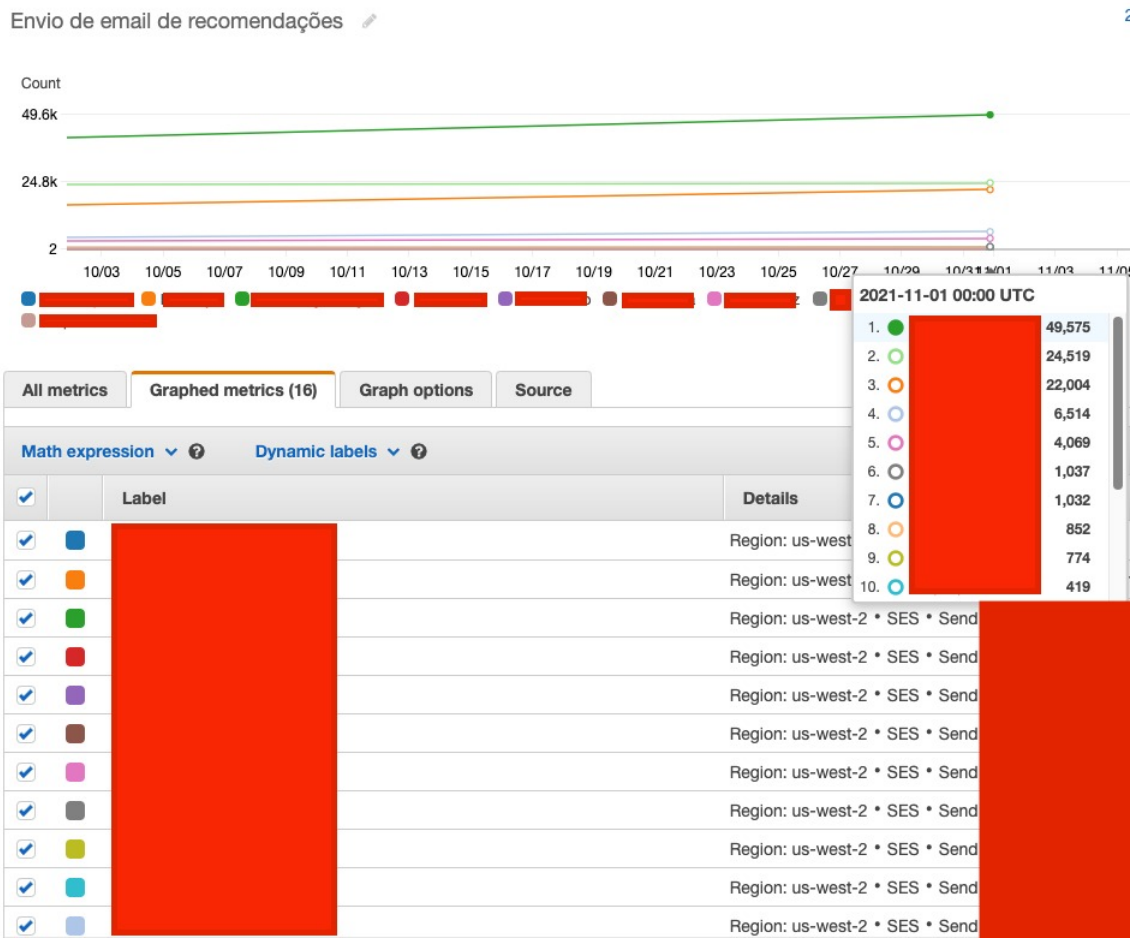

Figura 64 – Quantidade de e-mails enviados

7 CONCLUSÕES

O principal objetivo do trabalho foi desenvolver e aprimorar o sistema de recomendações da plataforma do Canal Digital utilizando de recursos, técnicas e algoritmos complexos e enraizados dentro da área de *Data Science*. Seguindo uma metodologia bem estruturada, foi possível partir de um sistema de recomendações simplificado, o qual era baseado apenas em um *score* calculado pela média ponderada dos filtros de entrada que o usuário selecionava, para um sistema com uma *pipeline* mais complexa e bem organizada, a qual utiliza um série de recursos e com uma heterogeneidade muito maior que anteriormente.

Quanto ao resultado, o projeto conseguiu trazer muitos pontos positivos, os principais foram consequência da implementação da clusterização no sistema. Essa técnica ajudou a agrupar mais imóveis por características similares e contribuiu para aumentar o repertório do sistema de recomendações, o qual era restrito a uma simples busca do usuário. Dessa forma, o produto ganha um leque maior de recomendações, o usuário pode ter como entrada um filtro, um imóvel específico ou uma quantidade n de imóveis, e mesmo assim a *pipeline* do sistema será capaz de lidar com a entrada e fazer uma recomendação.

Um ponto de melhoria em relação a clusterização é o cálculo automático do parâmetro de número de *clusters* com o auxílio do coeficiente de silhueta. Atualmente, esse cálculo é feito de forma manual e que tem potencial para ser aprimorado, dado que cada região é diferente e pode possuir um número de *clusters* distinto.

No que diz respeito aos próximos passos, é possível pontuar algumas atividades. A validação com auxílio do teste A/B em produção é algo que ainda não foi feito, porém instrumentação desse teste já está sendo elaborada. Na Figura 65, observa-se que os dados estão sendo coletados para analisar a taxa de conversão, possuindo informações de quantos e-mails são mandados, abertos e clicados.

Figura 65 – Instrumentos para realização do teste A/B

Outra conclusão que pode ser feita é sobre o experimento com o filtro colaborativo. Esse experimento ajudou a entender que a recomendação com os dados

de interação atuais não será satisfatória, e portanto, é necessário fazer mais aquisições quanto a esse tipo de dado. Com essa informação, um trabalho para o futuro seria aprimorar o desenvolvimento do algoritmo ALS que utiliza a abordagem do filtro colaborativo.

Mais um ponto de melhoria seria no ajuste fino de campos usados no treinamento de dados, principalmente os geográficos. Dado que regiões categóricas e com alta cardinalidade foram um problema de lidar durante a preparação dos dados, ficando restrito as coordenadas dos imóveis.

E por último, também é possível adicionar ao sistema dados de fontes externas para enriquecer o treinamento dos dados coletados. Por exemplo, dados socioeconômicos fornecidos pelo IBGE ou dados índices imobiliários.

REFERÊNCIAS

BELL, Robert M.; KOREN, Yehuda. Lessons from the Netflix Prize Challenge. **SIGKDD Explor. Newsl.**, Association for Computing Machinery, New York, NY, USA, v. 9, n. 2, p. 75–79, dez. 2007. ISSN 1931-0145. Disponível em: https://doi.org/10.1145/1345448.1345465.

BRUGMAN, Simon. **pandas-profiling: Exploratory Data Analysis for Python**. [*S.l.*: *s.n.*], 2019. Disponível em: https://github.com/pandas-profiling/pandas-profiling.

BURKE, Robin. Hybrid Recommender Systems: Survey and Experiments. **User Modeling and User-Adapted Interaction**, v. 12, nov. 2002. DOI: 10.1023/A:1021240730564.

CHAPMAN, Peter; CLINTON, Janet; KERBER, Randy; KHABAZA, Tom; REINARTZ, Thomas P.; SHEARER, Colin; WIRTH, Richard. **CRISP-DM 1.0: Step-by-step data mining guide**. [*S.l.*], 2000.

CROCKFORD, Douglas. **The application/json media type for javascript object notation (json)**. [*S.l.*], 2006.

GRAHAM, Scott; MIN, Jun-Ki; WU, Tao. Microsoft recommenders. **Proceedings of the 13th ACM Conference on Recommender Systems**, ACM, set. 2019. DOI: 10.1145/3298689.3346967. Disponível em: http://dx.doi.org/10.1145/3298689.3346967.

HARDESTY, LARRY. **The history of Amazon's recommendation algorithms**. [*S.l.*: *s.n.*], 2019. Disponível em: https://www.amazon.science/the-history-of-amazons-recommendation-algorithm.

HARRIS, Charles R. *et al.* Array programming with NumPy. **Nature**, Springer Science e Business Media LLC, v. 585, n. 7825, p. 357–362, set. 2020. DOI: 10.1038/s41586-020-2649-2. Disponível em: https://doi.org/10.1038/s41586-020-2649-2.

JUNGLE DEVS. **Jungle Devs**. [*S.l.*: *s.n.*]. Disponível em: https://jungledevs.com/.

JUPYTER TEAM. **Jupyter Notebook Documentaion**. [*S.l.*: *s.n.*]. Disponível em: https://jupyter-notebook.readthedocs.io/en/stable/notebook.html#.

KLUYVER, Thomas *et al.* Jupyter Notebooks – a publishing format for reproducible computational workflows. *In*: LOIZIDES, F.; SCHMIDT, B. (Ed.). **Positioning and Power in Academic Publishing: Players, Agents and Agendas**. [*S.l.*: *s.n.*], 2016. IOS Press, p. 87–90.

KONSTAN, Joseph A; EKSTRAND, Michael D. **Recommender Systems Specialization**. [*S.l.*: *s.n.*]. Disponível em: https://www.coursera.org/specializations/recommender-systems.

NARAYAN, Sriram. **Agile IT Organization Design: For Digital Transformation and Continuous Delivery**. 1st. [*S.l.*]: Addison-Wesley Professional, 2015. ISBN 0133903354.

PEDREGOSA, F. *et al.* Scikit-learn: Machine Learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

PORIYA, Anil; BHAGAT, Tanvi; PATEL, Neev; SHARMA, Rekha. Non-personalized recommender systems and user-based collaborative recommender systems. **Int. J. Appl. Inf. Syst**, v. 6, n. 9, p. 22–27, 2014.

PROVOST, Foster; FAWCETT, Tom. **Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking**. 1st. [*S.l.*]: O'Reilly Media, Inc., 2013. ISBN 1449361323.

RICCI, Francesco; ROKACH, Lior; SHAPIRA, Bracha; KANTOR, Paul B. **Recommender Systems Handbook**. 1st. Berlin, Heidelberg: Springer-Verlag, 2010. ISBN 0387858199.

TEAM, The pandas development. **pandas-dev/pandas: Pandas**. [*S.l.*]: Zenodo, fev. 2020. DOI: 10.5281/zenodo.3509134. Disponível em: https://doi.org/10.5281/zenodo.3509134.

TORVALDS, Linus; HAMANO, Junio. **Git**. [*S.l.*: *s.n.*], 2005. Disponível em: https://git-scm.com/.

VAN ROSSUM, Guido; DRAKE JR, Fred L. **Python tutorial**. [*S.l.*]: Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

WOLIVER. **Woliver API documentation**. [*S.l.*: *s.n.*]. Disponível em: `https://api.woliver.net/docs/`.

ZILLOW. **Introduction to Recommendations at Zillow**. [*S.l.*: *s.n.*], 2017. Disponível em: `https://www.zillow.com/tech/introduction-recommendations-zillow/`.