UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Luan Lázaro Vieira

ANÁLISE DE DADOS DO CORPO DE BOMBEIROS DE FLORIANÓPOLIS E IMPLEMENTAÇÃO DE UM MAPA INFORMATIVO DE OCORRÊNCIAS

Florianópolis
2019

LUAN LAZARO VIEIRA

ANÁLISE DE DADOS DO CORPO DE BOMBEIROS DE FLORIANÓPOLIS E IMPLEMENTAÇÃO DE UM MAPA INFORMATIVO DE OCORRÊNCIAS

Trabalho de Conclusão de Curso submetido à Universidade Federal de Santa Catarina, como requisito necessário para obtenção do grau de Bacharel em Ciências da Computação

Florianópolis, 12 de junho de 2019

UNIVERSIDADE FEDERAL DE SANTA CATARINA

LUAN LÁZARO VIEIRA

Esta Monografia foi julgada adequada para a obtenção do título de Bacharel em Ciências da Computação, sendo aprovada em sua forma final pela banca examinadora:

Orientador(a): Profa. Vania Bogorny
Universidade Federal de Santa Catarina - UFSC

Prof. Dr. Renato Fileto
Universidade Federal de Santa Catarina - UFSC

Prof. Dr. Luis Otavio Campos Alvares
Universidade Federal de Santa Catarina - UFSC

Florianópolis, 12 de junho de 2019

Agradecimentos

Agradeço a professora Vânia Bogorny pela orientação, pelo apoio, por acreditar em mim durante o projeto, sempre me trazendo uma nova perspectiva para cada dificuldade encontrada e também pelo esforço extra no final do semestre para melhorar meu texto. Muito obrigado! Sem a sua ajuda eu não chegaria aqui. Também agradeço ao prof. Luis Otavio Alvares, pelas sugestões e conversas, me senti muito acolhido. Agradeço ao Corpo de Bombeiros de Santa Catarina, pelos dados fornecidos, por me apresentarem seu ambiente de trabalho e pela atenção quando tive dificuldades de entender o funcionamento da corporação.

Agradeço a minha namorada Georgia Vieira, por conseguir me acalmar, me alegrar nos momentos difíceis e pelas sugestões no texto, sem ela tenho certeza que não teria motivação o suficiente para continuar. Agradeço também à minha psicóloga e psiquiatra, que acompanharam e deram suporte para manter minha saúde mental para completar o projeto e também à minha família que foi atenciosa comigo durante o projeto. Muito obrigado a todos!

A ilusão é só uma realidade que você deixou de acreditar.

Resumo

O corpo de bombeiros de Santa Catarina armazena dados das ocorrências desde 2007. Esses dados possuem informações que podem ser relevantes não só para um levantamento estatístico das ocorrências, mas, também, para identificar características entre as ocorrências. Por exemplo, existem horários do dia em que mais acontece um determinado tipo de ocorrência, época do ano e dia da semana? Condições climáticas ou estações do ano tem alguma correlação com as ocorrências. Existe algum padrão espacial, ou seja, regiões na cidade onde tenha maior incidência de ocorrências? Visando responder estas perguntas, este trabalho tem como objetivo a implementação de uma ferramenta para a visualização geográfica das ocorrências e a busca por padrões através de técnicas de data mining com o uso dos algoritmos de Regras de associação e clustering.

Palavras-chave: emergências corpo de bombeiros, mineração de dados, análise de dados, associação e clusterização.

Sumário

Lista de ilustrações

1 Introdução

Os algoritmos de mineração de dados são antigos. É difícil marcar uma data inicial para a sua concepção, visto que eles nasceram de um crescimento natural de estudos estatísticos e com o próprio avanço da computação. Ainda assim, há pouco tempo, o mundo vivia uma época onde havia muita informação armazenada e os próprios portadores dessa informação, sejam eles empresas ou instituições, eram incapazes de analisá-las. Hoje, apesar do volume de dados continuar crescendo em uma proporção maior, a realidade já não é mais a mesma. As técnicas para interpretar os dados de forma útil estão sendo cada vez mais disseminadas, tanto na área científica como na área tecnológica. Existe ainda um substancial volume de informação inutilizada, porém, muitas das informações que nos eram inúteis tornaram-se essenciais e com elas tornou-se possível prever comportamentos, identificar tendências e encontrar correlações através do processo de mineração de dados, com o objetivo de extrair novos conhecimentos de forma mais acessível aos seus usuários [Pang-ning, Michael e Vipin 2009].

Um exemplo de informação disponível para ser minerada são registros das ocorrências do corpo de Bombeiros de Santa Catarina . As informações registradas pela corporação abrangem diversos tipos de ocorrências como incêndios, resgates, salvamentos, entre outros. Nesses registros, é possível encontrar informações como cidade, horário do dia, data, tipo da ocorrência e logradouro. Utilizando essas informações, é possível encontrar características sobre as ocorrências atendidas. O estudo [Matt et al. 2015] mostra que, nos Estados Unidos, incêndios florestais acontecem com maior frequência em dias sem chuvas e secos, contudo, considerando toda a dinâmica da sociedade, o quanto que cada uma dessas características, como a ausência de precipitação e de umidade no ar, são um indicativo de um provável incêndio em Florianópolis? Será que algum dia da semana pode ser um fator de uma ocorrência? Ou talvez uma época do ano? Será que é possível agrupar as ocorrências em relação a comportamentos comuns entre elas? É difícil afirmar que essas características são as que realmente levaram à um incidente, mas podem ser indicadores de onde pode ser útil realizar análises.

O objetivo deste trabalho é analisar os dados das ocorrências registradas pela unidade de Corpo de Bombeiros de Santa Catarina. A base disponibilizada possui informações relacionadas a diversos tipos de ocorrência, a qual varia de atendimentos pré hospitalares até controle de insetos e pragas. Neste trabalho serão explorados dois tipos de análise : (i) analise espacial para visualizar a distribuição geográfica das ocorrências e (ii) mineração de dados.

1.1 Objetivos

Este trabalho apresenta uma análise dos atendimentos das ocorrências de Florianópolis no período de fevereiro de 2007 até abril de 2018. Utilizando a base de dados disponibilizada, a análise tem o propósito de identificar características comuns de cada tipo de emergência. Além da análise, será disponibilizada uma ferramenta de visualização, onde será possível visualizar todas as ocorrências atendidas em Florianópolis em um dado período. Sendo assim, os objetivos específicos incluem:

1. Identificar locais como bairros, logradouros, praias ou praças que possuam a maior parte das ocorrências em relação a cada tipo de emergência e seu respectivo total de ocorrências.
2. Dado um tipo de emergência, identificar as características mais comuns referentes às informações climáticas, dias da semana, época do ano, entre outras, em relação a cada um dos logradouros com maior frequência de ocorrências e exibir suas distribuições.
3. Dado um tipo de emergência, identificar a variação do número de ocorrências de bairros com maior número de ocorrências em relação ao ano anterior.
4. Criar uma ferramenta que permita visualizar espacialmente a distribuição das ocorrências atendidas pelo corpo de bombeiros para todos os bairros, logradouros, praias e praças disponibilizados e suas respectivas informações climáticas.
5. Aplicar diferentes técnicas de *data mining* sobre os dados visando a descoberta de padrões.

1.2 Metodologia

A metodologia de pesquisa e implementação utilizado neste trabalho foi dividida nas seguintes etapas:

1. Explorar os dados fornecidos pela corporação para reconhecimento de padrões;
2. Busca de informações que poderiam ser agrupadas com os dados originais;
3. Definir quais dados serão utilizados no trabalho;
4. Construção de uma estrutura que permita visualizar geograficamente as informações da base de dados;
5. Pesquisar técnicas de mineração que sejam compatíveis com os dados fornecidos;
6. Analisar os dados e padrões descobertos;
7. Contextualizar e exibir os padrões mineradas;

1.3 Escopo e organização do trabalho

O escopo deste trabalho inclui a exibição, através de uma ferramenta de visualização dos dados das ocorrências fornecidas pelo corpo de bombeiros assim como a extração, análise e exibição das ocorrências mais comuns. O presente trabalho está organizado da seguinte forma:

Capítulo 2: Conceitos básicos e contextualização dos dados;

Capítulo 3: Ferramenta de visualização;

Capítulo 4: Mineração dos dados;

Capítulo 5: Conclusão;

2 Conceitos básicos e contextualização dos dados

Este capítulo descreve as ferramentas, dados e técnicas utilizadas neste trabalho. Na seção 2.1 são descritas as características e também a motivação do uso dos dados utilizados neste trabalho. Na seção 2.2 são apresentadas algumas técnicas de modelagem assim como a metodologia de mineração de dados utilizada. Por último, na seção 2.3, são apresentadas as tecnologias utilizadas para o desenvolvimento do projeto.

2.1 Dados

Foram utilizados três conjuntos de dados diferentes para a elaboração deste trabalho. O conjunto de dados da seção 2.1.1 referem-se aos dados da tabela "Ocorrências"disponibilizada pela corporação, os dados da seção 2.1.2 fazem referência aos dados climáticos e o conjunto de dados da seção 2.1.3 fazem referência ao mapeamento geográfico realizado a partir dos nomes dos bairros e logradouros.

2.1.1 Dados das ocorrências

A base de dados disponibilizada pelo Corpo de Bombeiros de Santa Catarina possui registros de ocorrências de fevereiro de 2007 até abril de 2018. A tabela das ocorrências possui 1.814.563 registros, com atributos como a descrição inicial da ocorrência, tipo da emergência, data de inicio, data de finalização e um logradouro vinculado a ocorrência. Em relação à consistência dos atributos, apenas os atributos do logradouro, data da ocorrência, horário e tipo da emergência estão registrados em praticamente todas as ocorrências, enquanto os demais atributos possuem informações incompletas, vazias e de texto aberto. Visto que o volume de dados para o estado inteiro é muito grande, e principalmente porque os padrões de ocorrências podem variar de uma região para outra no estado, optou-se por analisar apenas a cidade de Florianópolis. Neste conjunto de dados observou-se que praticamente todos os dias alguma ocorrência foi registrada.

Definidas as informações que serão utilizadas, é necessário destacar os tipos de emergência que foram atendidos pela corporação, visto que as análises deste trabalho irão partir desses tipos de emergência. A base de dados contém os tipos de emergência listados na tabela 1.

Incêndio
Auxílios e apoios
Produtos perigosos
Salvamento, busca e resgate
Atendimento pré-hospitalar
Ocorrência não atendida
Diversos
Acidente de trânsito
Ações preventivas
Averiguação e corte de árvore
Averiguação e manejo de inseto

Tabela 1: Tipos de emergências.

2.1.2 Dados climáticos

Com o intuito de extrair informações que possam ser relevantes em relação as ocorrências, utilizou-se dados externos aos dados disponibilizados. Levantou-se a hipótese de que as características meteorológicas da data da ocorrência poderiam ser fatores relevantes para o acontecimento da ocorrência como, por exemplo, um maior número de acidentes de trânsito em dias mais chuvosos, incêndios em dias mais quentes e secos, afogamentos em épocas de veraneio, ataques por insetos em épocas de maior calor e umidade, etc. Por isso, utilizou-se registros climáticos diários para correlacioná-los com as ocorrências atendidas. Os dados utilizados foram extraídos do Banco de Dados Meteorológicos para Ensino e Pesquisa (BDMEP), sendo importante ressaltar que cada registro do BMEP é referente a um período de 24 horas. Através do BMEP, foram coletadas as informações climáticas de Florianópolis no período de fevereiro de 2007 até abril de 2018 e as características climáticas extraídas estão listadas na tabela 2.

Insolação
Temperatura mínima
Temperatura máxima
Precipitação
Temperatura média
Umidade relativa do ar média

Tabela 2: Características climáticas utilizadas.

2.1.3 Dados das ocorrências com geolocalização

Os dados disponibilizados inicialmente pelo corpo de bombeiros possuíam apenas o nome dos logradouros, sem a localização geográfica.Para obter a localização geográfica

aproximada, foi utilizado o Open Street Maps [OpenStreetMap 2019], fazendo um *matching* entre os nomes dos logradouros da base de dados das ocorrências e os logradouros do Open Street Maps, buscando o centroide do logradouro. Optou-se por extrair o centroide do logradouro porque a base de dados fornecida inicialmente não possui a informacao do numero da rua onde aconteceu a ocorrencia. Para definir as delimitações dos bairros utilizou-se a mesma *api* para encontrar os polígonos referentes a geometria dos bairros.

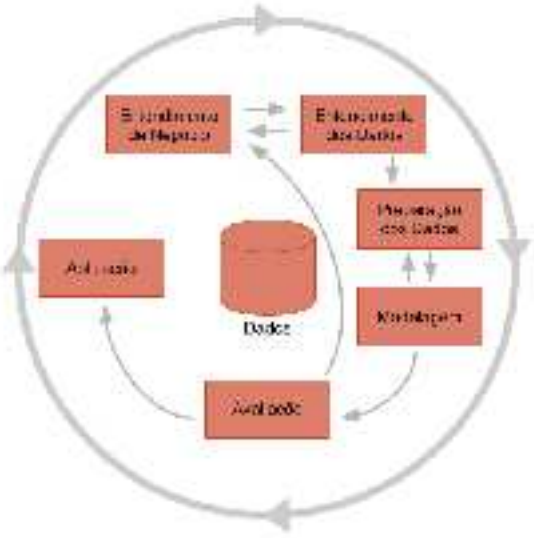
2.2 Mineração de dados e descoberta de conhecimento

A mineração de dados é uma técnica que visa analisar dados buscando extrair conhecimento. O processo de mineração de dados utiliza métodos inteligentes, como aprendizado de máquina e estatísticas, para realizar análises e extrair conhecimento, transformando o grande volume de dados em padrões, geralmente compreensíveis. Segundo [Han, Kamber e Jian 2012] *Data mining* é também popularmente conhecido como descoberta de conhecimento a partir de bases de dados *(Knowledge Discovery in Databases*), ou KDD, é o processo de extração de padrões implícitos escondidos em grandes bases de dados, *data warehouses*, na internet, etc.

A mineração de dados é composta por várias etapas que são descritas pelo processo de descoberta de conhecimento (KDD - *Knowledge Discovery in Databases*). O processo de descoberta de conhecimento foi idealizado por [Usama, Gregory e Padhraic 1996] e se resume em completar uma série de etapas que compõe desde a seleção, pré-processamento e transformação dos dados até, finalmente, a mineração destes dados e a interpretação do conhecimento gerado. Durante a etapa de seleção, são obtidos dados que se encontram geralmente brutos e necessitam passar por um pré-processamento, que é a etapa seguinte do processo. Durante o pré-processamento é realizada a limpeza de registros que estão incompletos, redundantes ou que não sejam úteis para o dado objetivo. Já na etapa da transformação, novos dados são gerados a partir dos dados pré-processados. Por fim, na etapa de *data mining* é feita a busca por padrões nos dados gerados de forma a descobrir conhecimento para ajudar nas tomadas de decisões.

Uma das formas mais recomendadas de elaborar um trabalho de mineração de dados é através da metodologia CRISP-DM (Processo Padrão Inter-Indústrias para Mineração de Dados). O CRISP-DM foi criado com o objetivo de desenvolver processos capazes de padronizar a elaboração de projetos de mineração de dados. O guia do CRISP-DM [Chapman et al. 2000] define que o processo de mineração é dividido em seis fases de forma cíclica, sendo elas: Entendimento do Negócio, Entendimento dos Dados, Preparação dos Dados, Modelagem, Avaliação e Disponibilização. A figura 1 ilustra o ciclo das fases do CRISP-DM. As atividades relacionadas com cada fase do CRISP-DM serão descritas a seguir em um formato similar ao que foi publicado pelo livro *Data Mining for Dummies* [Brown 2014].

Figura 1 – Ciclo do processo CRISP-DM



2.2.1 Entendimento do Negócio

O entendimento do negócio é a primeira fase do processo de mineração. O foco nessa etapa é procurar compreender os requisitos do domínio. Dessa forma, esta etapa é dividida nas seguintes atividades: (i) Identificar objetivos do negócio, (ii) avaliar a situação atual e (iii) definir os objetivos de mineração dos dados. Na primeira atividade, procura-se identificar os objetivos que o negócio almeja, contextualizando o cenário atual do negócio, definindo os meios que capacitem alcançar o mais cedo possível os critérios de sucesso. Na etapa de avaliar a situação atual, deve-se identificar os recursos disponíveis para a realização do projeto, sendo eles o tempo, custos, tecnologias ou até pessoas. Na etapa de definição dos objetivos da mineração dos dados deve-se correlacionar os principais objetivos do projeto de mineração. Após a compreensão nas características do domínio, é definido a direção que o projeto irá seguir com a finalidade de solucionar um certo problema ou atingir um dado objetivo. Estudar e compreender trabalhos similares são passos necessários para descobrir aspectos que influenciam ou estão relacionados ao projeto.

2.2.2 Entendimento dos Dados

Na segunda etapa do processo de metodologia CRISP-DM, é marcada do Entendimento dos Dados, é onde os dados serão coletados e, também, verificados em relação a sua qualidade. É comum nesta etapa a percepção de que os dados coletados possam não ser ideais para o propósito do projeto, sendo este o momento mais adequado para avaliar os objetivos e buscar novos dados ou até voltar para a etapa anterior e refinar o Entendimento do Negócio. Sendo assim, esta etapa engloba as atividades de (i) obtenção de dados, (ii) descrição dos dados coletados, (iii) exploração dos dados e (iv) verificação da qualidade dos dados. Na atividade de coleta de dados é realizada a checagem da disponibilidade dos

dados, a obtenção prática desses dados, descrevendo como foram obtidos, sejam os dados disponibilizados por alguma entidade ou através de um processo de coleta. Na atividade seguinte, chamada de exploração dos dados, deve-se analisar os tipos coletados, descrever os seus formatos, sendo eles inteiros, texto-livre, ou outro formato. A próxima atividade é a verificação dos dados. O objetivo dessa atividade é familiarizar-se com os dados, identificar problemas e começar a idealização da forma em que deverão ser utilizados na etapa de Preparação dos Dados.

2.2.3 Preparação dos Dados

A preparação dos Dados é a etapa onde os dados são prepararados para que eles possam ser usados em algum técnica de modelagem. Um modelo de *data mining* é criado através da aplicação de algoritmos de mineração. O termo padrão no contexto de mineração de dados refere-se a um conjunto de dados que se repetem numa frequência maior do que esperava-se pelo acaso, destacando que esse conjunto de dados possa ter algum comportamento padronizado ou dependente. Esta etapa é citada por alguns autores como a mais trabalhosa de todo o processo de mineração de dados. Segundo [Adriaans, Zantinge e (Firm) 1997] cerca de 60% do esforço do projeto é utilizado para as atividades dessa etapa. As atividades a serem realizadas para completar a Preparação dos Dados são as seguintes: (i) Seleção, (ii) limpeza, (iii) integração e (iv) formatação. A primeira atividade, seleção dos dados, é onde serão selecionados os dados que serão utilizados para o processo de mineração. Isso significa que deverão ser selecionados os dados que forem considerados mais apropriados depois da coleta, podendo assim, ser utilizado todo o conjunto de dados coletados ou somente uma parte. A segunda atividade, limpeza dos dados, foca em identificar dados que não estejam aptos para serem usados, sejam eles informações faltantes, conteúdo ambíguo ou inapropriados. A atividade seguinte, a integração dos dados, tem o objetivo de agrupar os dados selecionados e é finalizada no momento em que todos os dados que pretende-se utilizar estão unidos em um ou mais arquivos. Por fim, a atividade de formatação de dados visa transformar os dados em formatos que sejam compatíveis para o uso dos algoritmos de mineração. Após a finalização da atividade de formatação, os dados estarão prontos para a etapa seguinte.

2.2.4 Modelagem

Após a conclusão das etapas anteriores, os dados devem estar aptos para serem processados pelos algoritmos de mineração de dados, assim começando o processo de reconhecimento de padrões. As atividades que são realizadas durante a Modelagem são as seguintes: (i) Selecionar uma técnica de modelagem para mineração de dados, (ii) planejar testes, (iii) construir o modelo e a (iv) avaliação do modelo. A atividade de seleção da técnica de modelagem consiste em escolher alguma técnica já criada em trabalhos

anteriores buscando encontrar aquelas que possam ser utilizadas nos dados que foram tratados. Existem diversas tarefas de mineração de dados já desenvolvidas, cada uma delas possui seus algoritmos, seus dados de entrada e seus resultados. A atividade seguinte, chamada de planejamento de testes, visa selecionar dados que irão ser utilizados como base para testes e quais dados serão usados para treinamento. Essa atividade refere-se apenas para a tarefa de classificação, que é detalhada na próxima seção. Dessa forma, ainda durante o planejamento de testes, espera-se que sejam definidos critérios que avaliem a qualidade do modelo gerado. Em seguida, durante a atividade de construção do modelo, deve-se realizar a descrição do modelo que será utilizado e configurar os parâmetros que serão usados na modelagem em si. A seguir são descritas as tarefas de Classificação, Regras de associação e *Clustering*, também chamado de Agrupamento.

2.2.4.1 Classificação

[Pang-ning, Michael e Vipin 2009] define classificação sendo a atividade de mineração de dados que foca em organiza-los em categorias definidas previamente. Por exemplo, classificar cogumelos em comestíveis e venenosos, a previsão climática, grupos de animais, etc. O grupo de dados deve ser classificado anteriormente. Para realizar a classificação é necessário identificar cada registro como sendo uma dupla (x,y), sendo o x um conjunto de atributos de entrada e y o atributo considerado como classe, que representa os valores que os atributos de entrada possam assumir, assim permitindo a classificação dos registros e a geração do modelo. O modelo resultante da técnica de classificação é capaz de indicar a classe de registros que não foram classificados anteriormente. Alguns dos tipos de classificadores que se destacam por sua frequência de uso, são os chamados de classificadores espertos e os classificadores preguiçosos. Os classificadores espertos separam os arquivos em dois conjuntos: um conjunto de treinamento e um conjunto de testes. O conjunto de treinamento é utilizado para gerar as classes pré-definidas e o conjunto de testes avalia o modelo de classificação gerado. Exemplos de algoritmos de classificação espertos são : árvores de decisão e redes neurais. Já os classificadores preguiçosos não utilizam dados para treinamento e classificam os dados na medida que novos registros chegam, designando a classe desse novo registro a partir de todos os dados que já foram classificados previamente. Exemplo desse classificador é: *K Nearest Neighbors Algorithm (KNN)*

2.2.4.2 Agrupamento ou *clustering*

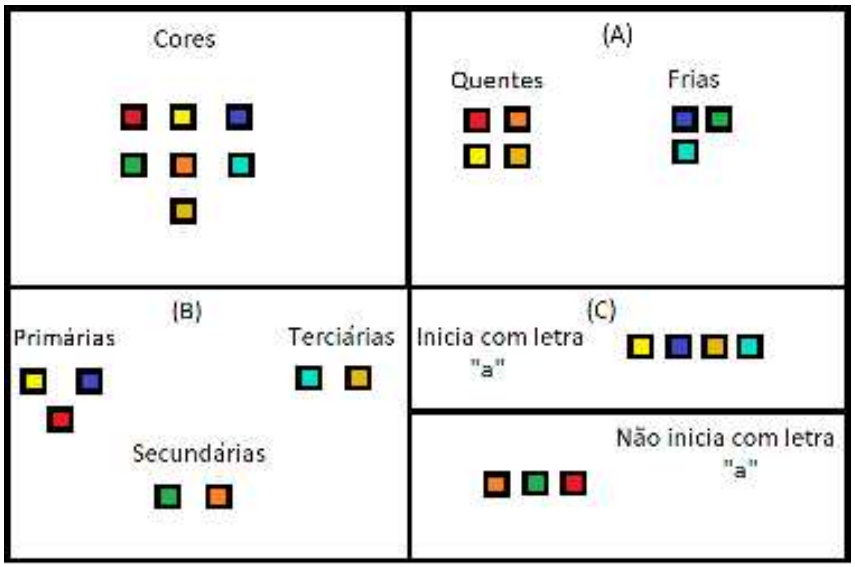
A técnica de agrupamento é citada por [Pang-ning, Michael e Vipin 2009] como uma técnica capaz de unir registros baseando-se nos valores dos atributos dos registros e seus relacionamentos. A união dos registros, chamada de agrupamento, é citada por Charu C. Aggarwal [C. e K. 2013] como um modelo conciso onde os dados podem ser interpretados no formato de um resumo. Outra forma de identificar um problema de

agrupamento, de acordo com Charu C. Aggarwal, pode ser descrita da seguinte forma: Dado um conjunto de registros, particione-os em grupos que sejam o mais semelhantes possível entre si. O objetivo do uso de técnicas de agrupamento é o de identificar grupos similares e destacá-los dos outros conjuntos. Segundo [Pang-ning, Michael e Vipin 2009] a qualidade do agrupamento gerado depende da semelhança dos dados dentro do seu próprio grupo e a diferença entre os dados de grupos diferentes. Quanto maior a semelhança dentro de um grupo e quanto maior a diferença entre os outros grupos, melhor a qualidade dos *clusters*. Sendo assim, dependendo do critério de agrupamento, um conjunto de dados pode ser agrupado de diversas formas. Para exemplificar as diversas formas que um dado conjunto pode ser agrupado, a tabela 3 mostra um exemplo de cores e a figura 2 mostra um conjunto (A) onde as cores foram agrupadas por sua temperatura, um conjunto (B) onde as cores foram agrupadas por seu tipo de cor e um conjunto (C) as cores foram agrupadas pelo fato de possuir a letra inicial "a"ou não.

Cor	Temperatura	Tipo de cor
Vermelho	Quente	Primária
Amarelo	Quente	Primária
Azul	Fria	Primária
Verde	Fria	Secundária
Laranja	Quente	Secundária
Azul-esverdeado	Fria	Terciária
Amarelo-alaranjado	Quente	Terciária

Tabela 3: Cores e algumas características.

Figura 2 – Exemplo de agrupamento



Um dos algoritmos mais conhecidos de agrupamento é o K-means . Segundo [Pang-ning, Michael e Vipin 2009], o K-means é dito como uma técnica de agrupamento particional, ou seja, um dado registro só é vinculado a um único grupo. O k-means utiliza como

parâmetro o número de grupos que deseja gerar, sendo este número o valor "k" de clusters. O número de grupos muitas vezes é difícil de estimar. O algoritmo inicia selecionando aleatoriamente "k" registros da base de dados como sendo o centroide de cada "k" *clusters* na primeira iteração. As iterações seguintes identificam os registros que foram atrelados a cada um dos *clusters* gerados e atualiza o valor do novo centroide baseando-se nos registros agrupados na iteração anterior. Como o centroide foi modificado na iteração atual, ele verifica os registros que agora estão mais próximos do centroide atual para gerar o novo cluster. Esse procedimento é repetido até que o centroide não seja modificado entre as iterações. Cada atributo do grupo será representado por um ponto central que irá representar o atributo no grupo, sendo esta a origem do nome do algoritmo onde o *means* significa média. O fato de utilizar a média como métrica torna o agrupamento problemático em alguns casos. Segundo [Junjie 2012], a chegada do *big data* tem dificultado o uso do k-means por si só, pelo fato das informações possuírem muitos fatores, muitas dimensões e, também, ruído.

O Panda é um algoritmo criado para mineração de itens frequentes [Claudio, Salvatore e Raffaele 2013]. Ele gera conjuntos de itens comprados em conjunto nas mesmas transações. O PaNDa+ trabalha com uma matriz binária onde as linhas são as transações e as colunas são os itens. Este algoritmo realiza diversas iterações tentando encontrar o melhor conjunto de características (itens) que estão presentes em um conjunto de transações. Este algoritmo será usado pela primeira vez para fazer *clustering*, pois ele agrupa linhas e colunas e permite que a mesma linha faça parte de mais de um grupo e que cada grupo possa conter diferentes colunas. Desta maneira, o Panda, se utilizado para clustering, permite gerar grupos heterogênos do ponto de vista do número de atributos, ou seja, grupos que contenham atributos diferentes. Sob este aspecto, o Panda deve gerar clusters melhores do que o K-means, o qual gera grupos onde todos os atributos participam de um cluster, fazendo a média das distâncias de todos os atributos.

O Panda utiliza uma função de custo, que é um dos parâmetros utilizados para medir a qualidade dos padrões gerados e como critério de parada para encontrar o melhor conjunto de características que caracterize os dados. Os critérios de parada mais utilizados para esse algoritmo são um número pré-definido de padrões ou quando a função de custo não for modificada em relação as iterações anteriores. O ruído nesse algoritmo é tratado como "margem de erro", de forma que cada conjunto de características aceita uma tolerância de erro. Esse percentual pode ser definido nas configurações do algoritmo, mas ele também é calculado dinamicamente através da função de custo, otimizando assim a busca dos melhores padrões.

2.2.4.3 Regras de associação

Regras de associação são técnicas de aprendizado de máquina que possuem o objetivo de gerar regras a partir das relações entre os atributos de um conjunto de dados. A

técnica foi criada para mineração de conjunto de itens de supermercados e foi introduzido por [Agrawal e Srikant 1994] ao descobrir relações entre compras de produtos nos registros das vendas de supermercados. O algoritmo de regras de Associação mais conhecido é o Apriori proposto em 1994 por Agrawal e Skrikant, definindo um problema de regra de associação como sendo:

- Seja I = $\{i_1, i_2 ..., i_n\}$ um conjunto de n atributos chamado de itens.
- Seja D = $\{t_1, t_2 ..., t_m\}$ um conjunto de m transações chamado de base de dados.
- Cada transação no conjunto D possui seu identificador e contém um subconjunto de itens. Uma regra é definida como sendo uma implicação da forma:

$$X \Rightarrow Y, \text{ onde } XeY \subseteq I \tag{2.1}$$

- Cada regra é composta por dois grupos diferentes de itens, também conhecidos como *itemsets*, X e Y, onde X é chamado de antecedente e Y é chamado de consequente. Os algoritmos de regra de associação geram muitas regras e possuem formas para avaliar-las de forma a identificar regras fortes a partir de critérios de interesse. Alguns deles são:
- Suporte: O suporte é uma indicação de quão frequente um *itemset* aparece no conjunto de dados. Considerando **X e Y** como *itemsets*, $X \Rightarrow Y$ como uma regra de associação e **D** sendo um conjunto de transações de uma base de dados fornecida, o suporte de **X** em relação a **D** é definido como a proporção de transações **t** no conjunto de dados que contém o *itemset* **X**. E é definido formalmente por:

$$\text{supp}(X) = \frac{|X|}{|D|} \tag{2.2}$$

- As regras de associação são geradas a partir de *itemsets* frequentes. Uma medida usada para avaliar as regras é a confiança. A confiança é interpretada como uma estimativa da probabilidade condicional $P(E_Y|E_X)$, sendo esta a probabilidade de encontrar o consequente da regra em transações sob a condição de que essas transações também contenham o antecedente. O suporte de uma regra $X \Rightarrow Y$, em relação ao conjunto de transações **D**, é a proporção de transações que contém **X** e também contém **Y**. A confiança é definida por:

$$\text{conf}(X \longrightarrow Y) = \text{supp}(X \cup Y) / \text{supp}(X) \tag{2.3}$$

- Lift: O lift mede a importância das regras levando em consideração a frequência do consequente geradas utilizando a confiança da regra dividida pelo suporte do

consequente. Quanto maior o lift, melhor a qualidade da regra. O lift é definido por:

$$Lift = \frac{c(X \longrightarrow Y)}{supp(Y)} \tag{2.4}$$

2.2.5 Avaliação

Após a etapa de modelagem, ou seja, da mineração de dados, inicia a etapa de avaliação dos resultados. A etapa anterior do processo CRISP-DM gera resultados que podem ou não ser úteis. Esta etapa possui as seguintes atividades como parte do processo: (i) Avaliação dos resultados, (ii) revisão do processo, (iii) determinação dos passos seguintes. Na atividade de avaliação dos resultados, espera-se analisar os resultados obtidos de forma a verificar a sua relevância e sua coerência. Para isso, o conhecimento adquirido na etapa de Entendimento do Negócio torna-se essencial e somente conhecendo o domínio do projeto será possível efetivar essa avaliação sendo comum especialistas do domínio acompanharem a avaliação dos resultados. A atividade seguinte, revisão do processo, busca por possíveis falhas de execução e planejamento. É nessa atividade onde será reavaliado se o processo utilizado realmente alcançou os resultados esperados ou se existe alguma etapa que pode ser refinada. A revisão do processo utilizado é essencial para o próprio aprimoramento do processo de mineração de dados.

2.2.6 Disponibilização dos padrões descobertos

Esta etapa é dividida nas seguintes atividades: (i) Planejamento de implementação, (ii) planejamento de manutenção, (iii) exposição dos resultados e (iv) revisão dos resultados. A atividade de planejamento de implementação foca na utilização das descobertas encontradas para tomar alguma ação. Na parte de planejamento de manutenção, procura-se manter as análises atualizadas para que seja possível manter o refino contínuo das decisões. Para realizar as tomadas de ações, é necessário apresentar os resultados coletados de forma acessível, visto que a maioria das pessoas que realizam a mineração de dados não são as mesmas que tomam as decisões de ações, portanto, é necessário exposição dos resultados de forma compreensível. Finalmente, deve-se revisar os resultados do processo. Esse é o momento de conclusão onde todos os envolvidos no processo devem comunicar o que foi efetivo, trabalhoso e qualquer outra experiência que possa agregar para trabalhos posteriores.

2.3 Tecnologias utilizadas

Além das análises dos dados, o trabalho também tem o objetivo de criar uma ferramenta que facilite a visualização das ocorrências. Dessa forma, tornou-se necessário escolher algumas tecnologias para o desenvolvimento dessa ferramenta. Esta seção possui

a descrição e a motivação da escolha das tecnologias utilizadas para o desenvolvimento da ferramenta de visualização das ocorrências.

2.3.1 Linguagem utilizada: Python

A linguagem [Python 2019] foi escolhida por oferecer várias bibliotecas e frameworks bem documentados para a utilização de recursos que são essenciais para a ferramenta de visualização desenvolvida. Desses recursos, destaca-se banco de dados para aplicação, framework para desenvolvimento de páginas web, o pyplot para gerar histogramas e bibliotecas do próprio google maps para extração de geolocalização e suas respectivas exibições através do google maps.

2.3.2 Web *framework* : Flask

Para este projeto, foi utilizado o web framework [Flask 2019]. Um web framework contém ferramentas, bibliotecas e tecnologias que permitem a construção e desenvolvimento de aplicações web. A aplicação web desenvolvida utilizará páginas web para que o usuário da aplicação possa visualizar os bairros e logradouros das ocorrências e suas características. O flask foi escolhido por ser um framework minimalista, sendo o modo de configuração e utilização enxuto, permitindo a codificação de páginas webs sem se preocupar tanto com configurações. Por este motivo, o Flask é considerado um micro-framework, visto que ele possui pouquíssimas dependências com outras bibliotecas. Suas dependêcias são: o Werkzeug, que é o seu Web Server Gateway Interface(WSGI) e o Jinja2 que carrega os dados do código em Python para o respectivo template de página HTML.

2.3.3 Armazenamento dos dados: Dataset

Para evitar armazenagem de dados desnecessários para a aplicação, decidiu-se utilizar um banco de dados interno para gerenciar os dados. O [Dataset 2019] é uma biblioteca capaz de realizar uma abstração simples de operações básicas de banco de dados, de forma a tornar acessível a armazenagem, extração e manipulação de dados. O Dataset pode ser configurado de diversas formas, mas a configuração utilizada neste projeto foi de um banco de dados SQLite. Ainda assim, como ferramenta interessante para o projeto, o Dataset consegue registrar e extrair informações de modo trivial de alguns formatos populares como o formato JSON.

3 Ferramenta de visualização

Este capítulo apresenta as funcionalidades da ferramenta de visualização desenvolvida e as análises realizadas através dela. A seção 3.1 descreve a motivação do desenvolvimento da ferramenta, a seção 3.2 descreve a arquitetura utilizada para desenvolvimento da ferramenta, a seção 3.3 descreve as telas da ferramenta de visualização e as informações disponibilizadas através das mesmas e a seção 3.4 mostra algumas análises realizadas através da ferramenta desenvolvida.

3.1 Motivação

Os dados disponibilizados não possuem uma forma natural de serem visualizados. Para entender a dinâmica dos dados é necessário analisá-los e também utilizar técnicas para destacar as informações que os dados possam ter. Ainda assim, uma das formas mais práticas de analisar informação é através de imagens. Representar os dados em gráficos ou mapas torna o entendimento dos dados uma tarefa muito mais eficiente e também confortável. Por essa razão, a motivação para o desenvolvimento de uma ferramenta visual, que é o mapa.

3.2 Arquitetura

A ferramenta desenvolvida utilizou como *web framework* o Flask, que é conhecido como um framework minimalista para desenvolvimento de páginas web em Python. O flask possui apenas duas dependências, o Werkzeug e o Jinja2. Também foi utilizado o *virtualenv* para encapsular todas as dependências necessárias para o desenvolvimento.

A ferramenta também conta com diversos histogramas que são geradas a partir da biblioteca Plotly, uma biblioteca rica para criação de diversos tipos de gráficos. Para a exibição das ocorrências no mapa foram utilizadas as bibliotecas do Google Maps, as quais também possuem uma adaptação simplificada para o Flask. Para a coleta dos centroides dos logradouros e também para a identificação da geometria dos bairros, foi utilizada a API do OpenStreetMaps.

3.3 Telas e informações disponibilizadas

Foram desenvolvidas três telas: Uma interface para a visualização das ocorrências em relação a cidade de Florianópolis, onde é possível visualizar um mapa de calor que identifica os bairros que possuem mais ocorrências, outra interface para a visualização

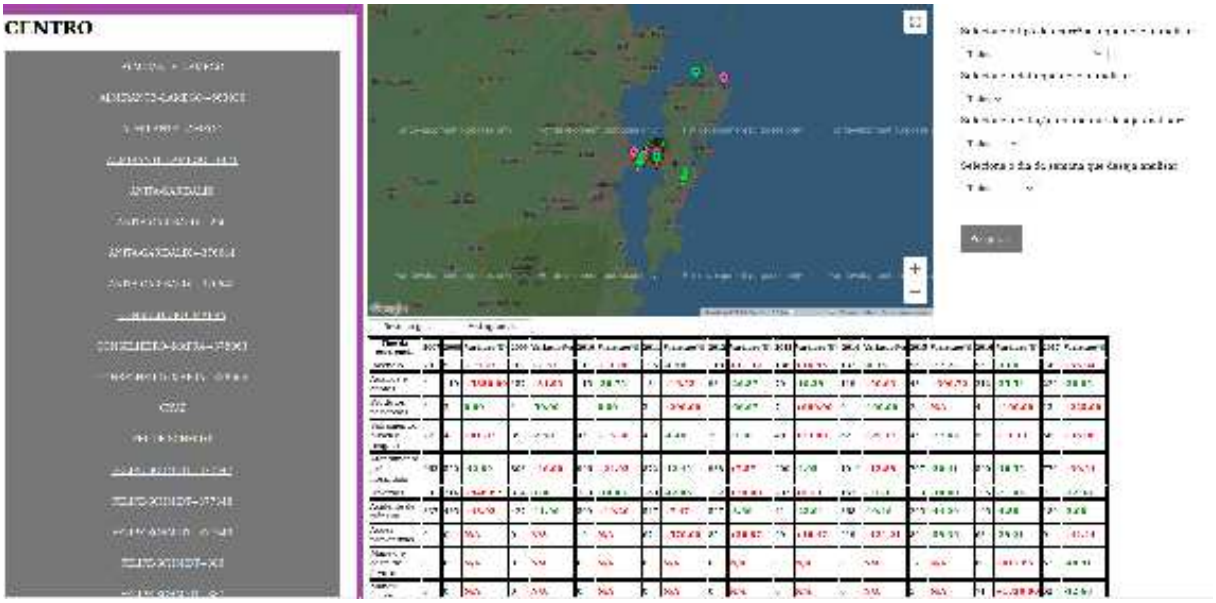
das ocorrências por bairro selecionado, identificando as ocorrências pelos centroides dos logradouros atrelados ao bairro e, a última tela, para a visualização das ocorrências por logradouro selecionado e suas respectivas informações de atendimento. Cada tela será descrita nesta seção e será especificado quais informações são exibidas através de cada interface.

3.3.1 Tela da visualização de Florianópolis

A tela inicial da aplicação é a mesma tela de visualização de Florianópolis. Na tela inicial, representada pela figura 3, é exibido um mapa de calor destacado em azul, uma tabela de resumo geral, campos para seleção de bairro, os filtros e um botão para seleção dos histogramas. A tela inicial, que na prática é a mesma tela de visualização de Florianópolis, mostra em seu respectivo mapa de calor a densidade de todas as ocorrências da cidade. O mapa de calor é calculado a partir do bairro que possuí o maior número de ocorrências e as suas demarcações referem-se aos polígonos encontrados desses bairros. O bairro é destacado no mapa de calor com o valor máximo de 100% referente ao bairro com o maior número de ocorrências. Os demais bairros possuirão uma intensidade relativa ao valor máximo, variando de 0% até 100%. Ao selecionar a opção "Todos" entre os bairros, o usuário será redirecionado para a mesma tela. Ao ser redirecionado, a nova tela exibida irá respeitar os filtros selecionados, onde será possível visualizar mapas de calores distintos e gerar novos histogramas, os quais respeitarão os respectivos filtros. Ao selecionar um bairro diferente da opção "Todos", o usuário será redirecionado para uma tela diferente, a tela de visualização por bairro.

Figura 3 – Tela da visualização de Florianópolis.

Figura 4 – Tela de visualização por bairro do "Centro".



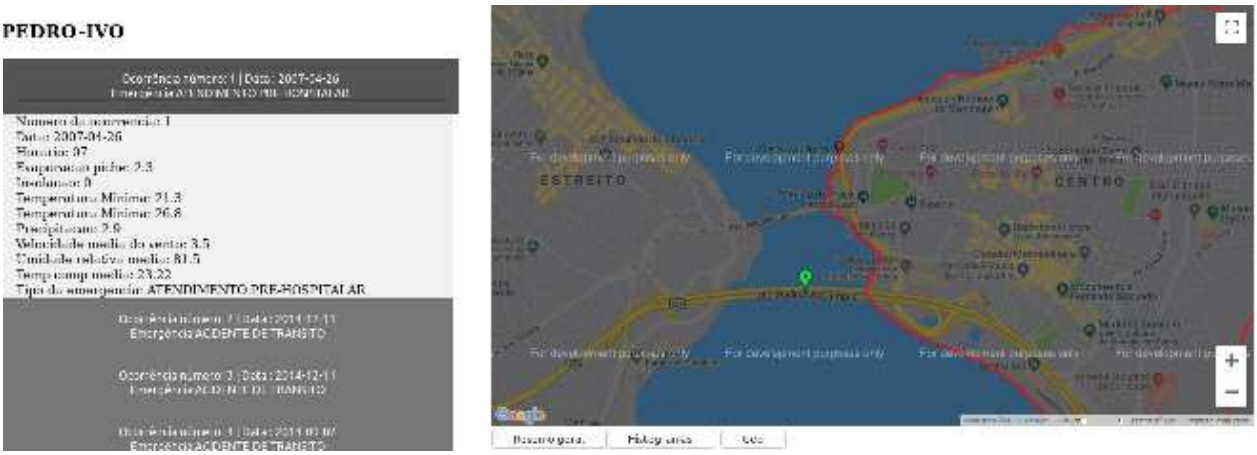
3.3.2 Tela da visualização por bairro

A tela de visualização por bairro, representada pela figura 4, exibe a lista de todos os logradouros atrelados ao bairro(independente dos filtros) destacada em roxo, exibe no mapa os logradouros que tiveram algum atendimento, a sua tabela de resumo geral e um botão para seleção dos histogramas. É importante ressaltar que foi usado o centroide do logradouro ao perceber que muitos logradouros foram cadastrados de forma incorreta. Algumas ocorrências foram cadastradas como "Logradouro X do bairro Y", mas, na verdade, a ocorrência foi atendida no "Logradouro X do bairro Z". Para manter a consistência e a coesão das informações disponibilizadas, utilizou-se o centroide do logradouro para referenciar somente a rua na qual a ocorrência foi cadastrada, sem garantir a sua posição em relação ao bairro. A figura 10 mostra uma ocorrência atendida na rua "Ícaro", que pertence ao bairro Centro, mas está vinculada com o bairro "Lagoa da conceição". O logradouro "Ícaro" não pertence a Lagoa da Conceição, mas, como foi vinculado à ela, será exibido na sua respectiva visualização do bairro indicando a posição real do centroide logradouro. Ao clicar em um elemento da lista de logradouros, o usuário será transportado para a tela da visualização por centroide do logradouro.

3.3.3 Tela da visualização por logradouro

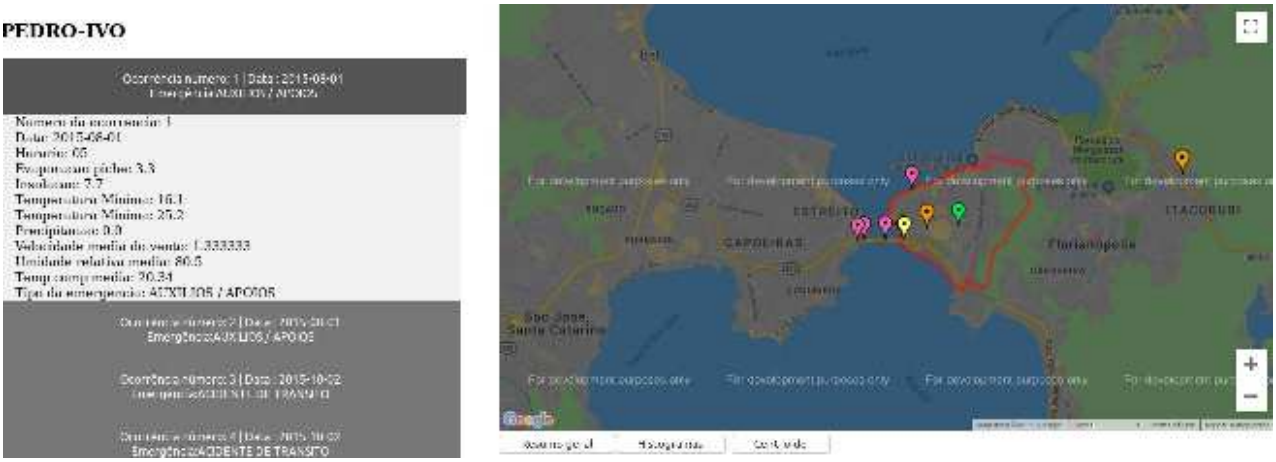
Esta tela possuía o objetivo de mostrar a geolocalização do atendimento ao longo da via, destacando, assim, pontos onde a densidade de ocorrências era maior. Para isso, tornou-se necessário ter a posição geográfica exata do atendimento ao invés de apenas o centroide do logradouro. Ao final deste trabalho, a corporação disponibilizou uma segunda base de dados de ocorrências com atributos nomeados de "geolocalização aproximada".

Figura 5 – Visualização da tela da visualização por centroide por logradouro do logradouro "Pedro Ivo".



Como foram fornecidas duas bases de dados distintas, dividiu-se a tela de visualização da distribuição dos logradouros em duas telas, que serão chamadas de tela de visualização por geolocalização aproximada do logradouro e tela da visualização por centroide do logradouro. A divisão foi realizada pois os dados com geolocalização aproximada possuem registros de 2015 a 2018, impossibilitando a visualização de estatísticas no período anterior a 2015. Como a base de dados fornecida inicialmente possui registros de 2007 até 2018, manteve-se ela disponível para visualização na tela da visualização por centroide do logradouro. Ambas as telas possuem a sua tabela de resumo geral, informações do atendimento, um botão para modificar a visualização entre centroide e geolocalização aproximada, um botão para seleção dos histogramas e a diferença entre elas refere-se à exibição das ocorrências no mapa. A tela da visualização por centroide do logradouro, representada pela figura 5, mostra um único marcador na posição do centroide do logradouro, esse marcador representa o tipo de emergência que mais foi atendida. A tela da visualização por geolocalização aproximada do logradouro, representada pela figura 6, exibe um marcador em cada geolocalização onde

Figura 6 – Visualização da tela da visualização por geolocalização aproximada por logradouro do logradouro "Pedro Ivo".



houve ocorrências, sendo estes marcadores identificados pelo tipo de emergência atendida naquele local.

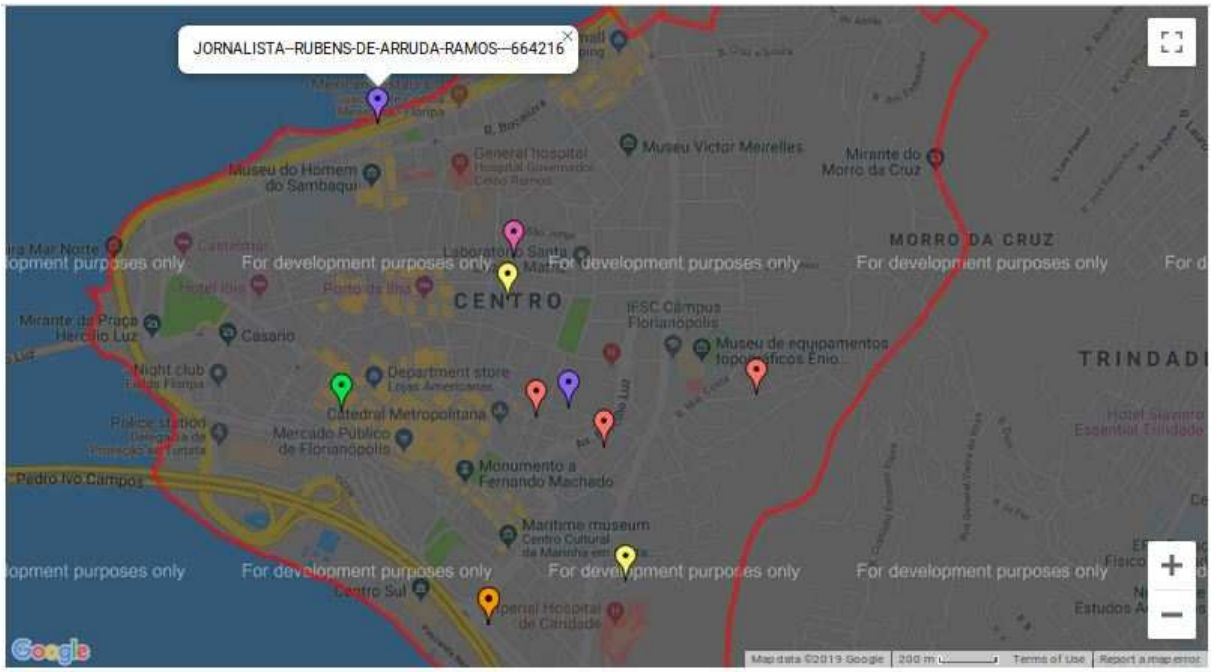
3.3.4 Filtros

Todas as telas possuem filtros para refinar a exibição das ocorrências no mapa. Os filtros disponibilizados são: tipo de emergência, ano, estação do ano e dia da semana. Os filtros são importantes pois é esperado que grande parte dos logradouros tenham alguma ocorrência ao longo dos 11 anos registrados, e, com a utilização dos filtros, é possível identificar ocorrências mais específicas como as da figura 7, que mostra os logradouros em que houve alguma ocorrência atendida no bairro "Centro" durante os domingos do outono de 2014. Cada logradouro exibido no mapa é identificado com um marcador, sendo o parâmetro de seleção deste marcador o tipo de ocorrência que foi registrada mais vezes naquele logradouro, totalizando 11 marcadores distintos. Então, para visualizar as ocorrências de um dado bairro, basta escolher os parâmetros de pesquisa e o bairro que se deseja visualizar. O resultado da visualização é um marcador posicionado no centroide de cada logradouro que possui alguma ocorrência, representando o tipo de ocorrência que mais foi atendida naquele logradouro.

3.3.5 Estatísticas disponíveis

Em todas as telas pode-se visualizar uma tabela de resumo geral das ocorrências e histogramas em relação aos dados climáticos. Essa tabela de resumo geral das ocorrências

Figura 7 – Ocorrências atendidas no bairro "Centro" durante os domingos do outono de 2014.



é similar à apresentada na tabela da figura 8, onde é possível ver o total de cada tipo de emergência atendida em Florianópolis, dividida por ano e mostrando a variação do total de ocorrências atendidas em relação ao ano anterior. A variação é indicada com cor verde quando houver uma diminuição da incidência, em vermelho quando tiver um aumento de incidência e em preto quando não houver atendimentos. O resumo geral é criado independentemente dos filtros e serve para destacar a distribuição das ocorrências ao longo dos anos, sem a necessidade de executar outras pesquisas para encontrar os principais anos ou principais ocorrências atendidas, seja na cidade, no bairro ou no logradouro. Para a tela de logradouro, é possível clicar no número do ano da tabela e visualizar o resumo anual daquele logradouro, tabela similar ao resumo geral, onde é mostrado o número de ocorrências em relação a cada mês do ano selecionado.

Todas as telas possuem um botão, localizado abaixo do mapa, para gerar histogramas. Os histogramas são exibidos em relação ao número de ocorrências e os seguintes atributos: Ano, hora do dia, tipo de emergência, umidade relativa, temperatura mínima, temperatura máxima, insolação, estação do ano e dia da semana. Todos os histogramas respeitam os filtros selecionados e podem ser gerados em todas as telas com exceção da tela inicial da aplicação. O primeiro histograma da figura 9 representa a quantidade de ocorrências por dia da semana que foram atendidas no bairro "Canto" durante o verão, enquanto o segundo histograma da mesma figura representa as ocorrências em relação à umidade relativa do ar do mesmo período.

3.4 Análises

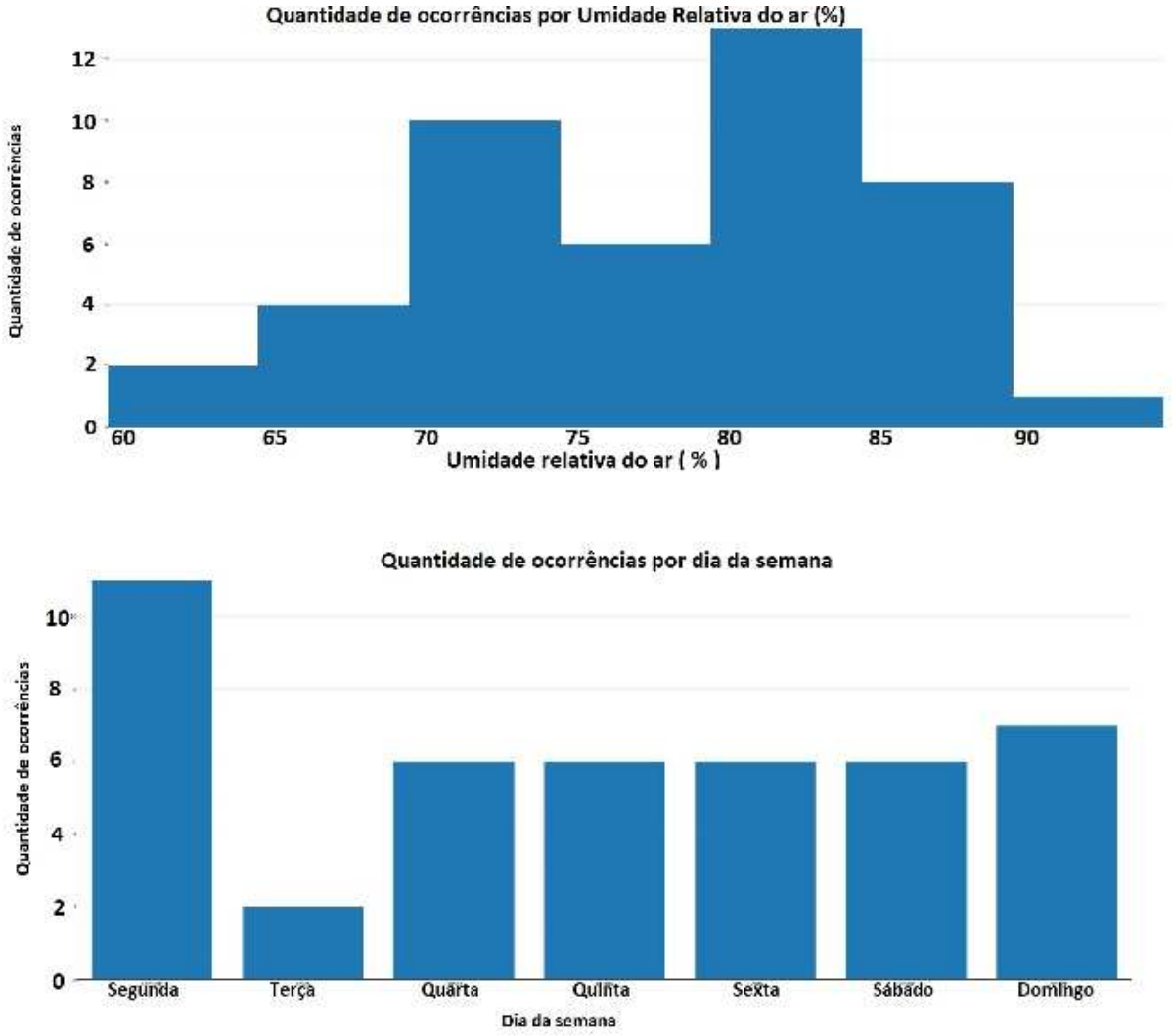
As análises foram feitas na ferramenta de visualização utilizando consultas ao banco de dados das ocorrências e também através das informações encontradas durante o processo de *data mining*.

Figura 8 – Tabela resumo geral da cidade.

Tipo da ocorrencia	2007	2008	Variacao(%)	2009	Variacao(%)	2010	Variacao(%)	2011	Variacao(%)	2012	Variacao(%)
Incêndio	223	[illegible]	+144.84	723	132.42	726	14.39	[illegible]	-14.99	820	+33.30
Auxílios e apoios	23	[illegible]	+2117.39	764	133.73	427	-36.01	[illegible]	+18.31	641	-25.13
Produtos perigosos	18	14	-22.22	10	-28.57	4	-60.00	5	+50.00	[illegible]	-33.33
Salvamento busca e resgate	72	[illegible]	+51.11	305	+107.74	470	+56.72	[illegible]	-60.04	156	-13.08
Atendimento pré-hospitalar	[illegible]	2878	-44.01	[illegible]	124.29	3272	-0.14	2162	-11.48	[illegible]	+6.01
Diversos	149	[illegible]	+1050.87	[illegible]	-1.42	[illegible]	-23.24	[illegible]	9.89	[illegible]	-13.77
Acidente de trânsito	[illegible]	[illegible]	+83.90	[illegible]	-4.05	3204	19.94	2222	+3.30	[illegible]	-8.41
Ações preventivas	0	5	N/A	0	N/A	75	N/A	271	+261.00	[illegible]	+13.55
Manejo e corte de árvore	0	5	N/A	0	N/A	0	N/A	[illegible]	N/A	0	N/A
Manejo animal	0	5	N/A	0	N/A	0	N/A	[illegible]	N/A	0	N/A

Tipo da ocorrencia	2013	Variacao(%)	2014	Variacao(%)	2015	Variacao(%)	2016	Variacao(%)	2017	Variacao(%)
Incêndio	927	10.55	1061	+14.40	[illegible]	-30.19	920	122.27	[illegible]	123.30
Auxílios e apoios	523	17.69	[illegible]	+49.31	2240	+458.71	[illegible]	-34.88	[illegible]	-27.43
Produtos perigosos	17	+325.00	[illegible]	-64.71	4	-33.33	9	+125.00	43	+377.78
Salvamento busca e resgate	[illegible]	+22.00	[illegible]	+13.31	274	+18.61	325	-3.28	314	+18.49
Atendimento pré-hospitalar	[illegible]	110.37	[illegible]	+13.85	2240	-20.84	[illegible]	-3.82	[illegible]	49.81
Diversos	[illegible]	111.21	[illegible]	-11.01	[illegible]	-2.18	[illegible]	-40.44	[illegible]	-21.05
Acidente de trânsito	[illegible]	-13.06	[illegible]	-0.19	[illegible]	-42.90	[illegible]	42.78	2640	15.19
Ações preventivas	373	+20.00	765	+106.43	[illegible]	-41.00	330	-21.36	[illegible]	+11.54
Manejo e corte de árvore	0	N/A	[illegible]	N/A	85	N/A	[illegible]	+1732.31	[illegible]	-18.87
Manejo animal	0	N/A	[illegible]	N/A	71	N/A	659	+828.17	[illegible]	0.00

Figura 9 – Exemplos de histogramas possíveis de serem gerados



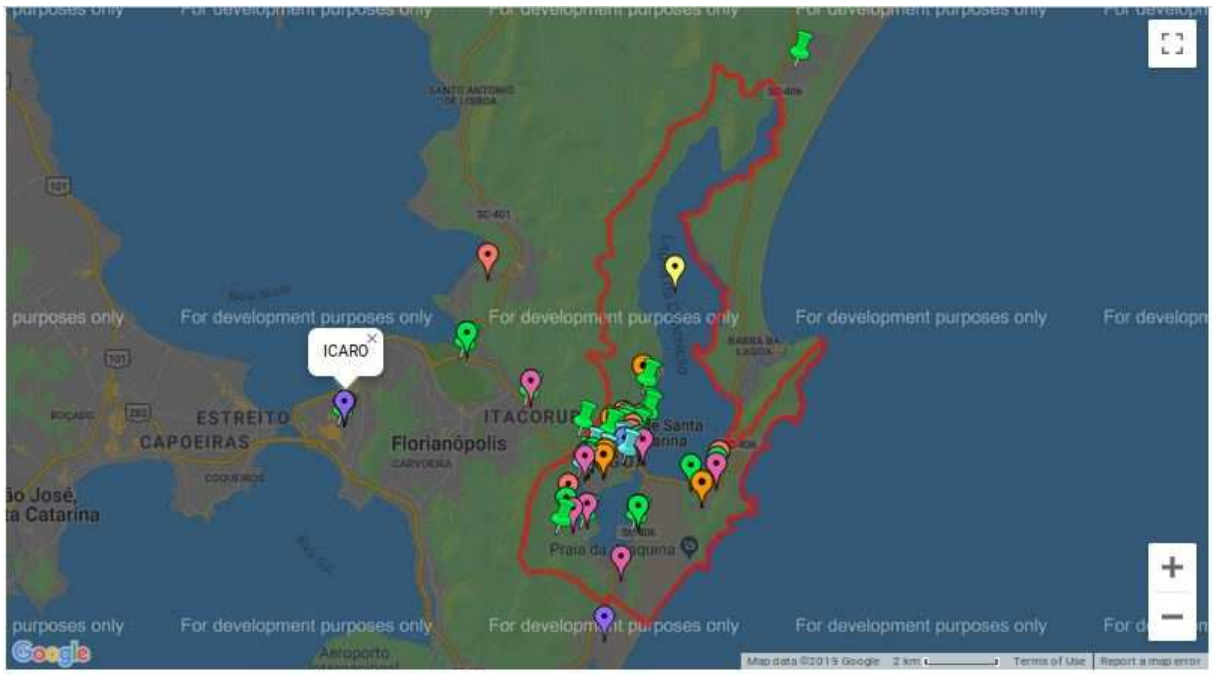
3.4.1 Densidade das ocorrências de Florianópolis

Para verificar a relevância dos principais bairros que possuem ocorrências em Florianópolis, utilizou-se a tela de visualização da cidade. A partir daí, visualizou-se a densidade das ocorrências em relação aos bairros, de forma a destaca-los ao comparar o número de ocorrências atendidas em cada bairro em relação ao bairro com o maior número de ocorrências. O mapa de calor da figura 11 mostra os atendimentos de "Averiguação e corte de árvore" da cidade de Florianópolis.

É possível notar que o corte de árvores acontece de forma distribuída ao longo da cidade, diferente da distribuição de "Acidentes de trânsito", indicados pela figura 12, onde é possível perceber que a grande maioria dos acidentes acontecem na região do "Centro" da cidade.

Através da ferramenta, percebe-se que não somente os acidentes de trânsito estão concentrados majoritariamente no centro da cidade e, sim, o total de ocorrências prevalecem no centro durante todo o período de 2007 a 2017, conforme mostra a tabela 4, que também

Figura 10 – Logradouro "Ícaro" cadastrado e exibido pelo visualização do bairro "Lagoa da Conceição".



ilustra os 5 bairros com o maior número de ocorrências atendidas em relação ao período. Para dar ênfase na análise realizada, a tabela 4 mostra os valores em porcentagem em relação à intensidade do mapa de calor mostrado no mapa. O mapa de calor varia sua intensidade entre 0% e 100% sendo 100% o valor do bairro que possuir o maior número de ocorrências e os demais bairros irão posuir uma porcentagem de intensidade que varia em relação ao bairro que obteve o maior número de ocorrências, que sempre será representado como 100%. Dessa forma, dizer que um bairro possuiu uma intensidade de 60% indica que ele registrou 60% do número de ocorrências do valor máximo registrado de ocorrências no bairro que tiver registrado o valor 100%, ou seja, do bairro que teve o maior número de ocorrências atendidas. A numeração no topo da tabela 4 indica, em ordem decrescente, os bairros com maiores densidades de ocorrências. É possível notar que o bairro "Centro", além de ter o maior número de ocorrências durante todos os anos, possui de 2008 até 2014 mais ocorrências que os outros 4 bairros da tabela somados. É interessante destacar que o bairro "Lagoa da conceição" não aparece na tabela até o ano de 2015, quando ele surge pela primeira vez possuindo uma densidade de 77% no mapa de calor, portanto indicando que ele registrou 77% de ocorrências em relação ao bairro com mais atendimentos daquele ano, ou seja, o bairro "Centro". Esse valor é muito maior que qualquer outra densidade dos principais bairros dos anos anteriores, com exceção do próprio "Centro". Isso destaca o fato de algo ter acontecido neste período para que a quantidade de ocorrências nesse bairro tenha crescido tanto. Outra análise realizada na ferramenta diz respeito ao veraneio em Florianópolis. Durante essa época, a população de Florianópolis aumenta, chegando a triplicar em datas festivas como o Revellion, segundo a [Casan 2016]. Por isso, buscou-se a

Figura 11 Mapa de calor dos atendimentos de "Averiguação e corte de árvores" de Florianópolis

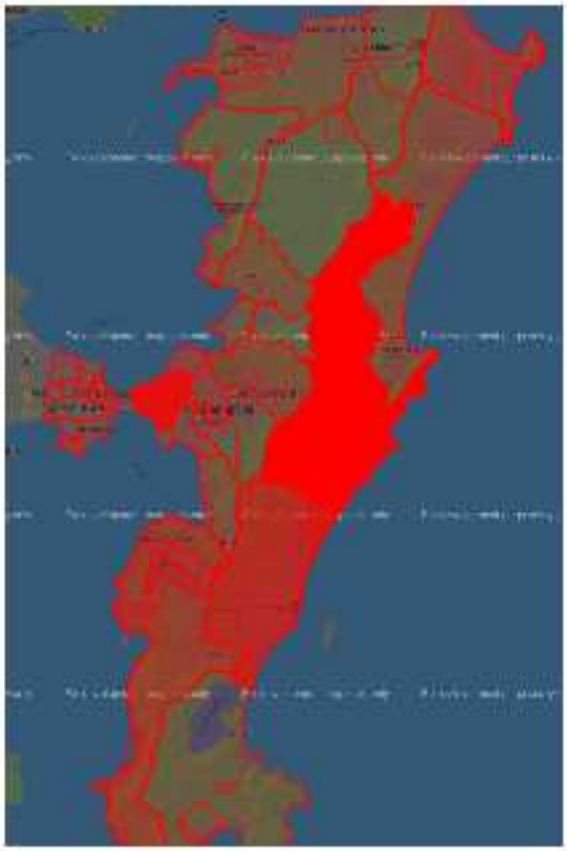


Figura 12 Mapa de calor dos atendimentos de "Acidentes de trânsito" de Florianópolis

densidade de ocorrências em relação as estações do ano na cidade. Para isso, comparou-se a densidade de todas as ocorrências em relação ao verão e as demais estações do ano, conforme mostra a tabela 5. A numeração no topo da tabela indica, em ordem decrescente, os bairros com maiores densidades de ocorrências. Nela é possível perceber que o bairro "Canasvieiras" aparece somente durante o verão, indicando que as ocorrências na região aumentam durante essa época, fato que não ocorre com a mesma intensidade nos demais bairros.

Ano	1°maior	2°maior	3°maior	4° maior	5°maior	(Soma do 2° ao 5°)
2007	CENTRO(100%)	ESTREITO(37%)	CAPOEIRAS(33%)	TRINDADE(19%)	AGRONOMICA(18%)	107,00%
2008	CENTRO(100%)	ESTREITO(29%)	CAPOEIRAS(28%)	TRINDADE(19%)	AGRONOMICA(18%)	95,00%
2009	CENTRO(100%)	ESTREITO(28%)	CAPOEIRAS(26%)	CANASVIEIRAS(21%)	TRINDADE(20%)	95,00%
2010	CENTRO(100%)	ESTREITO(22%)	CAPOEIRAS(21%)	LAGOA DA CONCEIÇÃO(17%)	TRINDADE(17%)	77,00%
2011	CENTRO(100%)	ESTREITO(26%)	CAPOEIRAS(22%)	TRINDADE(21%)	CANASVIEIRAS(16%)	85,00%
2012	CENTRO(100%)	ESTREITO(24%)	CAPOEIRAS(21%)	CANASVIEIRAS(18%)	TRINDADE(17%)	80,00%
2013	CENTRO(100%)	ESTREITO(24%)	CANASVIEIRAS(22%)	CAPOEIRAS(20%)	TRINDADE(20%)	86,00%
2014	CENTRO(100%)	ESTREITO(26%)	CAPOEIRAS(22%)	BARRA DA LAGOA(21%)	TRINDADE(18%)	88,00%
2015	CENTRO(100%)	LAGOA DA CONCEIÇÃO(77%)	ESTREITO(27%)	BARRA DA LAGOA(24%)	CANASVIEIRAS(23%)	151,00%
2016	CENTRO(100%)	LAGOA DA CONCEIÇÃO(73%)	ESTREITO(30%)	CAPOEIRAS(26%)	AGRONOMICA(24%)	152,00%
2017	CENTRO(100%)	LAGOA DA CONCEIÇÃO(56%)	ESTREITO(41%)	BARRA DA LAGOA(26%)	AGRONOMICA(26%)	149,00%

Tabela 4: Bairros com os maiores índices de ocorrências por ano.

	1°maior	2° maior	3° maior	4° maior	5°maior
Todos	CENTRO (100%)	LAGOA DA CONCEIÇÃO(30%)	ESTREITO(29%)	CAPOEIRAS(24%)	TRINDADE(20%)
Verão	CENTRO (100%)	LAGOA DA CONCEIÇÃO(32%)	CANASVIEIRAS(30%)	ESTREITO(28%)	BARRA DA LAGOA(24%)
Outono	CENTRO (100%)	ESTREITO(30%)	LAGOA DA CONCEIÇÃO(25%)	CAPOEIRAS(25%)	TRINDADE(22%)
Inverno	CENTRO (100%)	ESTREITO(28%)	LAGOA DA CONCEIÇÃO(25%)	CAPOEIRAS(24%)	TRINDADE(20%)
Primavera	CENTRO (100%)	LAGOA DA CONCEIÇÃO(33%)	ESTREITO(31%)	CAPOEIRAS(27%)	TRINDADE(21%)

Tabela 5: Bairros com os maiores índices de ocorrências por estação.

3.4.2 Distribuição das ocorrências em relação aos bairros

Após identificar a densidade das ocorrências, analisou-se mais detalhadamente as ocorrências em relação aos bairros e seus tipos de emergência.

Analisando as ocorrências em relação aos tipos de emergência "Atendimento pré-hospitalar" e "Acidentes de trânsito" dos bairros "Centro", "Capoeiras", "Estreito", "Trindade", "Agronômica" e "Barra da Lagoa", percebeu-se que ambos tipos de emergência tiveram oscilação de ano a ano, sem possuir um padrão de decréscimo ou de aumento. No ano de 2009, houve uma diminuição constante dos "Acidentes de trânsito" indicada pela tabela 6 no bairro de "Canasvieiras", no qual houveram 121 acidentes, finalizando em 2017 com apenas 48 acidentes. A tabela 7 mostra os atendimentos de "Acidentes de trânsito", "Atendimento pré-hospitalar"e "Auxílios e apoios" do bairro "Lagoa da Conceição" observa-se que a quantidade de "Acidentes de trânsito" manteve-se similar ao longo dos anos, com exceção do ano de 2016, onde houve o maior pico de acidentes da região. Em relação aos "Atendimento pré-hospitalares" observa-se um aumento de 219% durante o período de 2014 a 2015 e outro aumento de 57% de 2016 para 2017, crescendo o número de atendimentos de 93 em 2014 para 454 atendimentos em 2017. Além desse

aumento nos atendimentos pré-hospitalares, notou-se algo imprevisível: No ano de 2014 para 2015 houve um aumento de 2581% do número de ocorrências do tipo "Auxílios e Apoios"atendidos na "Lagoa da conceição", variando de 33 auxílios para 885. O número de auxílios e apoios na cidade toda é de 9390 ocorrências, sendo que 9.4% dessas ocorrências pertencem à "Lagoa da Conceição" no ano de 2015. Levando em consideração o aumento de "Atendimento pré-hospitalar" e "Auxílios e apoios", torna-se visível o motivo da "Lagoa da Conceição" aparecer com uma densidade de ocorrências muito maior em 2015.

Tipo da ocorrencia	2007	2008	Variacao(%)	2009	Variacao(%)	2010	Variacao(%)	2011	Variacao(%)	2012	Variacao(%)
Acidente de trânsito	38	87	103.64	121	39.08	117	-4.96	107	-8.70	95	-9.62
	2012	2013	Variacao(%)	2014	Variacao(%)	2015	Variacao(%)	2016	Variacao(%)	2017	Variacao(%)
Acidente de trânsito	95	77	-17.89	83	-19.33	53	-36.60	51	-2.83	49	-39.73

Tabela 6: Tabela dos acidentes de trânsito de "Canasvieiras" de 2007 a 2017.

Tipo da ocorrencia	2007	2008	Variacao(%)	2009	Variacao(%)	2010	Variacao(%)	2011	Variacao(%)	2012	Variacao(%)
Auxilios e apoios	2	11	450.00	21	90.91	17	-19.05	20	17.65	11	-45.00
Atendimento pré-hospitalar	150	59	-60.67	91	86.10	107	9.38	97	-12.99	115	39.34
Acidente de trânsito	40	92	130.00	100	8.70	121	21.00	99	-18.18	85	-14.14
	2012	2013	Variacao(%)	2014	Variacao(%)	2015	Variacao(%)	2016	Variacao(%)	2017	Variacao(%)
Auxilios e apoios	11	8	-27.27	33	312.50	885	2581.82	461	-47.91	203	-64.66
Atendimento pré-hospitalar	115	92	-20.00	91	38.70	297	219.35	288	-3.03	454	57.64
Acidente de trânsito	85	75	-14.12	63	-13.70	104	65.08	153	47.12	121	-20.92

Tabela 7: Tabela principais ocorrências "Lagoa da Conceição".

3.4.3 Distribuição das ocorrências em relação ao logradouros

Conforme descoberto durante a mineração de dados, que será descrita com mais detalhes na seção 4.3, percebeu-se que grande parte das ocorrências acontecem nos mesmos logradouros. Analisando as ocorrências do bairro "Lagoa da conceição"para identificar a geolocalização aproximada de suas ocorrências, visualizou-se os atendimento no logradouro "Renato Ântônio de Souza", mostrada a figura 13. Através da figura, percebe-se que a geolocalização aproximada das ocorrências fazem referência a muitos lugares distantes da "Lagoa da Conceição". Explorando os principais logradouros, pode-se perceber que existe um padrão bem definido para as ocorrências de acidente de trânsito em relação ao horário. A grande maioria dos acidentes acontecem entre as 17 e as 18 horas, enquanto acidentes entre as 02 até 05 horas são incomuns. Para ilustrar, a figura 14 mostra os histogramas em relação ao horário de acidentes de trânsito dos logradouros "Paulo Fontes" e "Jornalista Rubens de Arruda Ramos" .

Figura 13 – Visualização com geolocalização aproximada das ocorrências de logradouro "Renato Antônio de Souza".

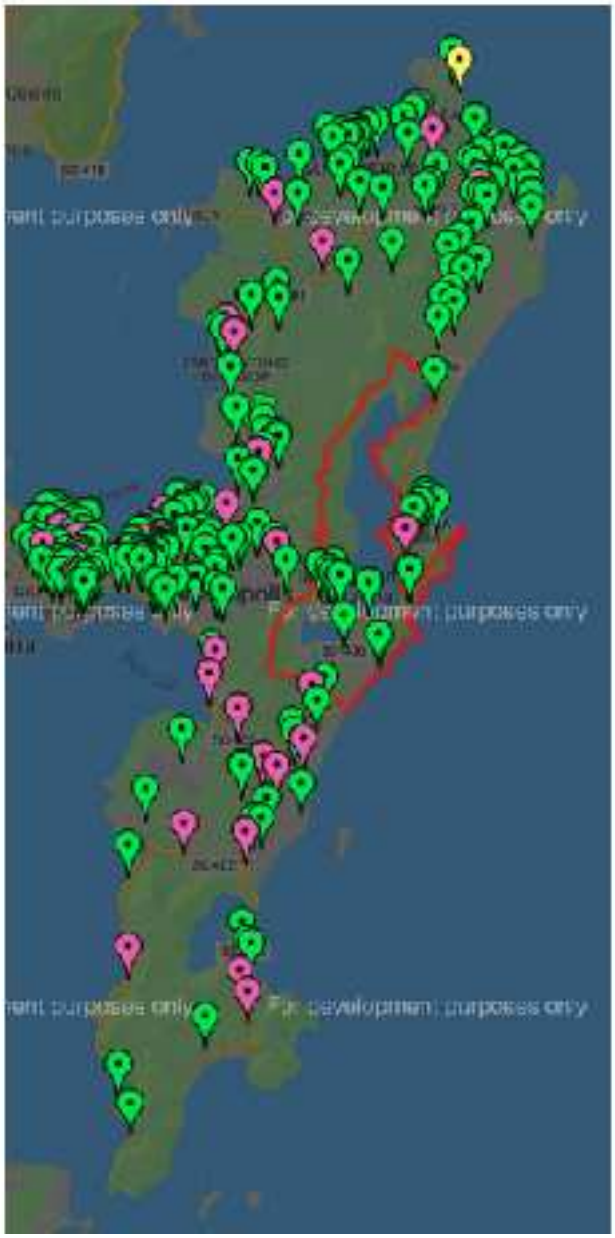
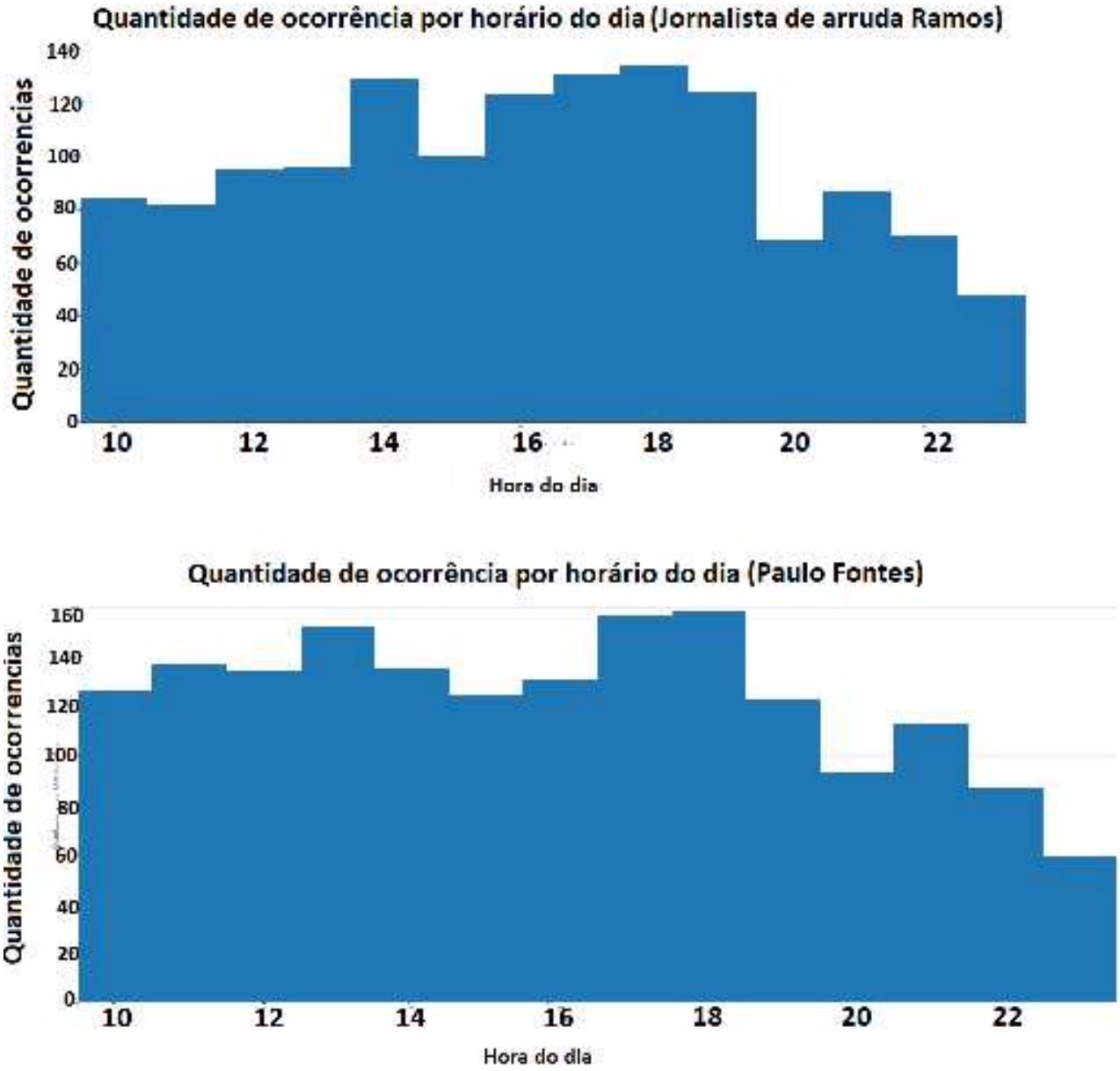


Figura 14 – Visualização dos horários mais comuns dos acidentes de trânsito .



4 Mineração dos dados

Este capítulo aborda a mineração de dados realizada e as seções deste capítulo referem-se a cada etapa do ciclo CRISP-DM. A seção 4.1 descreve a etapa de entendimento do negócio, a seção 4.2 descreve a etapa de entendimento dos dados, a seção 4.3 descreve a etapa de preparação dos dados, a seção 4.4 descreve a etapa de modelagem e avaliação e a seção 4.5 descreve a etapa de disponibilização.

4.1 Fase 1: Entendimento do Negócio

O domínio do projeto são os registros do corpo de bombeiros que foram introduzidos na seção 2.1. A primeira atividade do entendimento do negócio é identificar os objetivos do corpo de bombeiros. Para ilustrar os objetivos gerais do corpo de bombeiros será especificado as atividades que são realizadas pelo Corpo de Bombeiros Militares de Santa Catarina (CBMSC), a descrição dessas atividades foram extraídas diretamente do próprio site da corporação.

- Prevenção a Sinistros (Atividades Técnicas): Além de buscar oferecer serviços de excelência na resposta a sinistros, o CBMSC desenvolve ações paralelas para que estes episódios adversos tornem-se cada vez mais raros e, com isso, um número maior de vidas e bens sejam preservados. Neste sentido, desde a década de 1990 o CBMSC oferece serviços técnicos de prevenção a sinistros através da análise de projetos e vistorias.

- Educação Pública: Visando estimular a consciência dos riscos de sinistros envolvidos nos ambientes nos quais a população convive no dia a dia, o CBMSC desenvolve programas educacionais e de capacitação voltados para as diferentes faixas etárias, buscando disseminar a consciência prevencionista da segurança com o objetivo de diminuir a ocorrência de incêndios e outros sinistros.

- Atendimento Pré-Hospitalar (APH): O atendimento especializado de vítimas de trauma e emergências médicas ocorridas fora do hospital também é um dos serviços prestados pelo CBMSC à população no Estado. Com o emprego de ambulâncias tripuladas por bombeiros socorristas a instituição tornou-se referência na área, sendo responsável pela estabilização clínica e o transporte com segurança e rapidez de vítimas até os centros hospitalares - onde recebem o atendimento definitivo.

- Busca e Resgate: A localização e o socorro de pessoas e animais em perigo, perdidos ou em locais nos quais eles não podem sair por meios próprios – seja em ambiente

terrestre ou aquático - são atividades realizadas pela instituição em todo o território catarinense. Para cumprir esta abrangente missão a Corporação mantém seus integrantes capacitados através de treinamentos específicos e dispõe de diferentes recursos, mobilizados conforme a natureza da ocorrência. Conheça a relação das principais modalidades do serviço.

- Combate a Incêndio: Atividade histórica que deu origem à Corporação. Complexos, diferentes e com níveis diversos de dificuldade de combate, o serviço de extinção de incêndios envolve risco elevado, requerendo para seu sucesso treinamento específico, além do emprego de equipamentos e técnicas conforme as características do sinistro, seja o incêndio Urbano, Florestal ou Especial.

- Emergências com Produtos Perigosos: O uso de substâncias químicas, biológicas e radiológicas é cada vez mais comum e, por isso, cresce o risco de acidentes envolvendo a manipulação, estocagem ou o transporte desses materiais. A Corporação mantém equipes treinadas em seus quartéis para agirem na identificação, isolamento e contenção destas substâncias com o intuito de reduzir os riscos de dano às pessoas e ao meio ambiente.

- Operações Aéreas: A exemplo das maiores corporações de bombeiros no mundo, o CBMSC também salva vidas pelo ar através do serviço de operações aéreas. O uso de helicóptero (Arcanjo 01) e avião (Arcanjo 02) proporcionam atualmente, além do resgate e salvamento de vítimas em locais de difícil acesso, a agilidade necessária para o atendimento de pessoas que precisam da rápida remoção até o hospital.

- Ajuda Humanitária: O trabalho operacional do CBMSC em benefício da comunidade é complementado pela atividade de Ajuda Humanitária, com a prestação de assistência material ou logísticas para fins humanitários em resposta a situações de crise, como desastres naturais.

Segue-se para a atividade de avaliação da situação atual da corporação. O CBMSC é uma instituição que tem uma atuação abrangente em relação a sociedade e definir sua situação atual torna-se inviável sem a presença de alguém da própria corporação. Para este trabalho de conclusão de curso, a situação atual será dada pela forma como os registros fornecidos pela corporação já vem sido utilizados. A corporação hoje possui aplicações para gerar relatórios das ocorrências como quais e quantas ocorrências foram atendidas por quais unidades de Florianópolis. Acredita-se que os logradouros e os bairros da cidade são fatores importantes para o acontecimento de sinistros e, os sinistros, são descritos neste trabalho como "ocorrências"e "atendimentos". Após avaliar a situação atual do negócio, são definidos os objetivos da mineração dos dados. A extração das informações relacionadas aos bairros foi definida como objetivo principal da mineração de dados, identificando características dos lugares onde os incidentes mais ocorrem.

Bairro	Ocorrências
CENTRO	19066
LAGOA DA CONCEIÇÃO	5847
ESTREITO	5501
CAPOEIRAS	4661
TRINDADE	4347
BARRA DA LAGOA	3588
CANASVIEIRAS	3514
AGRONOMICA	3134
COQUEIROS	2668
RIO VERMELHO	2564
Total:	54890

Tabela 8 : Bairros com os maiores índices de ocorrências.

4.2 Fase 2: Entendimento dos Dados

A primeira atividade da etapa de entendimento de dados é a obtenção dos dados. Utilizou-se os dados fornecidos pela Unidade de Tecnologia e Informação do corpo de bombeiros e os dados climáticos do Banco de Dados Meteorológicos para Ensino e Pesquisa. A atividade seguinte consiste na descrição dos dados obtidos. Foram selecionadas algumas informações para ilustrar as informações disponibilizadas pelo corpo de bombeiros:

- O histórico das ocorrências atendidas possui 1.814.563 registros. Alguns dos atributos mais relevantes são os locais onde o incidente foi atendido e os identificadores do tipo da ocorrência, na qual define a atividade realizada durante o atendimento.
- Os lugares registrados nas ocorrências são relativos aos nomes dos logradouros. Estes logradouros podem estar vinculados com apenas um bairro e uma única cidade. Dessa forma, se um logradouro cruzar dois ou mais barros, ele terá dois ou mais identificadores diferentes registrados.
- Existem 104.425 logradouros, 8.118 bairros e 295 cidades registradas.
- As ocorrências de Florianópolis somam 109.279 registros.
- Existem 124 bairros vinculados a Florianópolis.
- Existem 5049 logradouros vinculados a Florianópolis.

A exploração dos dados é a próxima atividade desta etapa. Nessa atividade os dados obtidos são analisados para identificar quais atributos serão úteis para atingir os objetivos. Como foram utilizados dados climáticos e registros de ocorrências, cada um dos conjunto de dados serão descritos separadamente. Os dados climáticos obtidos possuem qualidade boa, visto que a maioria dos registros estão devidamente cadastrados. Contudo, eles se relacionam ao dia e não as horas do dia. Além dos atributos dos dados climáticos estarem cadastrados de forma consistente, esses registros são todos do tipo

numérico provindos de medidores de umidade, velocidade do vento, e seus respectivos sensores. A hipótese levantada supõe que as características climáticas possam ser um dos fatores que influenciam as ocorrências. Mudando a perspectiva dos dados climáticos para os registros das ocorrências, pode-se ver através da tabela 8 e da figura 15 que 54.890 de todas as ocorrências de Florianópolis, totalizando 50,23% do total de ocorrências da cidade, estão vinculadas com somente 10 bairros. Conforme mostra a tabela 9, quinze dos 5.049 logradouros possuem 17,48% das ocorrências, indicando os principais logradouros. Também é possível notar que alguns logradouros fazem referência ao próprio bairro, visto que na lista dos nomes dos logradouros com maior número de ocorrências encontram-se os nomes "FLORIANOPOLIS-CENTRO", "CANASVIEIRAS"e "BARRA DA LAGOA".

Logradouro	Quantidade de ocorrências	Porcentagem sobre total
RENATO ANTÔNIO DE SOUZA - 378665	2324	2,13%
PAULO FONTES	2097	1,92%
FLORIANOPOLIS-CENTRO	2031	1,86%
JORNALISTA RUBENS DE ARRUDA RAMOS	1770	1,62%
LAURO LINHARES	1360	1,24%
SANTOS SARAIVA	1273	1,16%
SC - 405	1136	1,04%
GOVERNADOR GUSTAVO RICHARD	1093	1,00%
BR 282	1083	0,99%
IVO SILVEIRA	999	0,91%
MAURO RAMOS	924	0,85%
JOÃO GUALBERTO SOARES - 1231	883	0,81%
CANASVIEIRAS	714	0,65%
BARRA DA LAGOA	712	0,65%
BALDICERO FILOMENO	707	0,65%
Total:	19106	17,48%

Tabela 9: Distribuição das ocorrências em relação aos principais logradouros.

Tipo de emergencia	Ocorrências	Representação do total
ATENDIMENTO PRÉ-HOSPITALAR	43353	39,67%
ACIDENTE DE TRÂNSITO	25660	23,48%
DIVERSOS	12189	11,15%
AUXÍLIOS / APOIOS	9390	8,59%
INCÊNDIO	8555	7,83%
AÇÕES PREVENTIVAS	3110	2,85%
SALVAMENTO / BUSCA / RESGATE	2124	1,94%
AVERIGUAÇÃO / CORTE DE ÁRVORE	2085	1,91%
AVERIGUAÇÃO / MANEJO DE INSETO	1673	1,53%
OCORRÊNCIA NÃO ATENDIDA	980	0,90%
PRODUTOS PERIGOSOS	160	0,15%

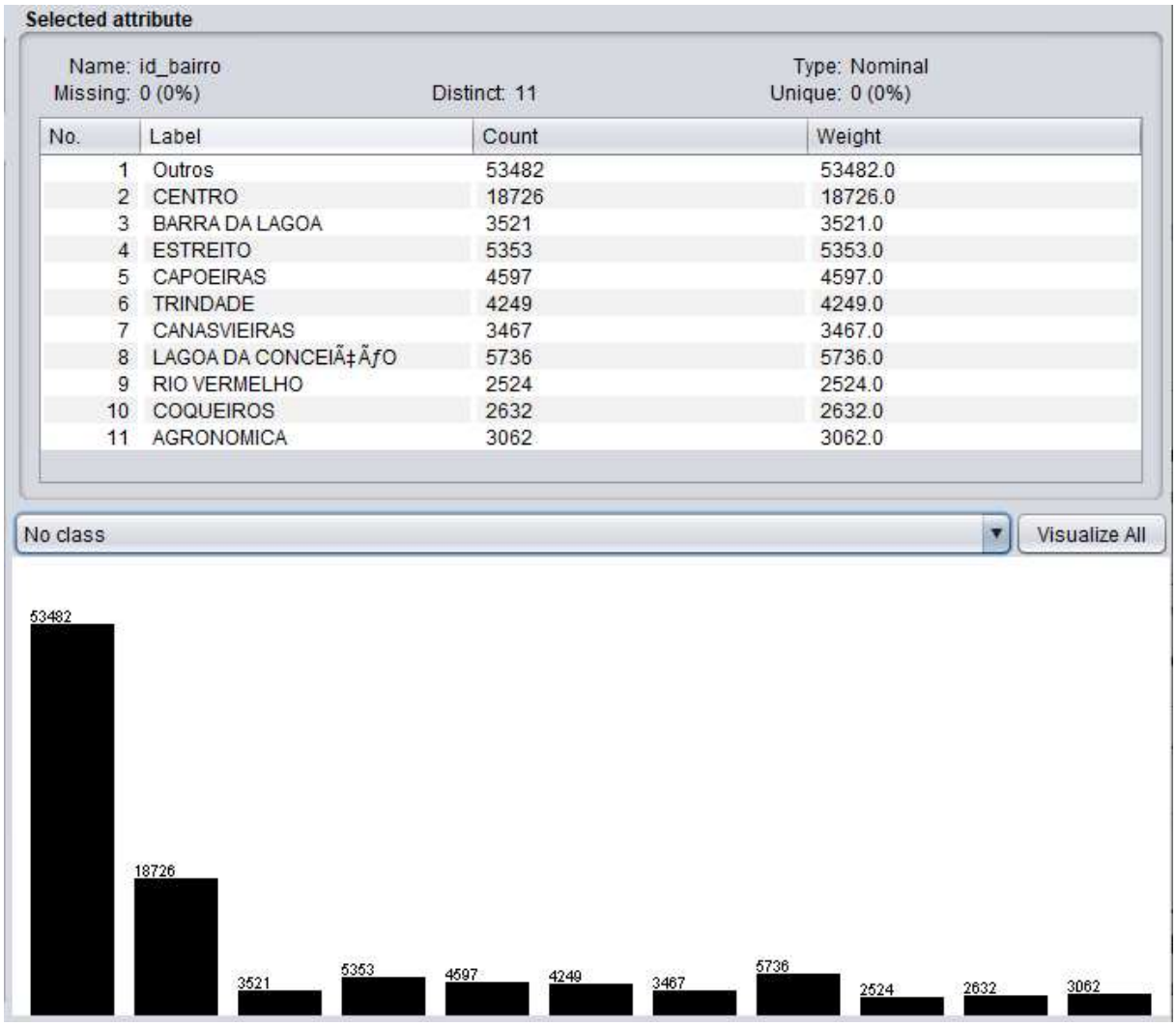
Tabela 10: Tipo de emergência por quantidade de ocorrências.

Ainda durante a exploração de dados, é possível perceber que os tipos de emergência não são distribuídas igualitariamente e existem ocorrências atendidas mais frequentemente

que outras conforme mostra a tabela 10. É possível notar nessa tabela que as atividades de "ATENDIMENTO PRÉ-HOSPITALAR" e "ACIDENTE DE TRÂNSITO" correspondem a 63,15% do total de ocorrências da cidade.

A última etapa desta fase é referente a averiguação da qualidade dos dados. Todos as características climáticas obtidas serão utilizadas para averiguar a hipótese levantada, pois possuem qualidade boa. Já, para os registros das ocorrências, grande parte dos atributos registrados pelos bombeiros são nulos e alguns outros são de texto aberto, indicando que nem todos os atributos são úteis para atingir os objetivos.

Figura 15 – Distribuição das ocorrências ao longo dos bairros.



4.3 Fase 3: Preparação dos dados

A primeira atividade desta fase consiste em escolher definitivamente os dados que serão utilizados. Foram utilizados apenas os atributos "id logradouro", "id tp emergencia", "dt ocorrencia", "tm ocorrencia"dos registros de ocorrências da cidade de Florianópolis para realizar a mineração, pois estes dados eram os únicos cadastrados de forma consistente.

Foi considerado consistente os atributos que estavam devidamente cadastrados em mais de 75% dos registros. A limpeza dos dados, que é a atividade seguinte, foi realizada nos dados climáticos, pois existiam valores muito distantes dos demais, como ventos com velocidade acima de 200 km/h. No caso dos dados das ocorrências, foram retiradas 1.930 ocorrências por não possuírem o "id logradouro", totalizando 107.349 registros para os algoritmos de mineração. Depois de efetuada a limpeza, foi realizada a integração dos dados através do atributo "dt ocorrencia" para vincular os dados das ocorrências com os registros climáticos. Para a finalização da fase de preparação de dados, realizou-se a transformação dos dados. Os dados climáticos foram discretizados de acordo com a tabela 12 e os dados das ocorrências tiveram um processo diferente de transformação. A partir das datas e horários das ocorrências, foram extraídos a sua estação do ano, o dia da semana, o dia do mês e a faixa de horário em que a ocorrência foi registrada conforme a tabela 13. Contudo, o algoritmo paNDa+ requer transformações aos dados que vão além das mostradas pelas tabelas 12 e 13. Para adequar os dados ao paNDa+ transformou-se cada instância de cada atributo em um identificador único, de forma que cada transação(ocorrência) será dada por um *array* de identificadores dos respectivos valores de seus atributos. Essa informação será utilizada para montar a matriz binária que é utilizada pelo algoritmo. Para ilustrar a transformação realizada, a informação "Hora:00" recebeu o identificador "69" e "DiaDaSemana:Segunda-feira" recebeu o valor "62". Além disso, para o PaNDa+, foram removidas as informações da maioria dos dados climáticos, preservando somente a precipitação e, também, foram adicionados as informações "Tipo do dia da ocorrência" que foram categorizadas entre "Dia Útil" e "Final de semana" e "Tipo de época do ano"que foram categorizadas entre "Alta temporada" e "Comum" .

Atributo	Transformação
Insolação	Entre 0 e 5: Baixa; Acima de 5: Alta
Temperatura mínima	Abaixo de 15: Baixa; Acima de 15 e menor que 25: Normal; Acima de 25 : Alta
Temperatura máxima	Abaixo de 18: Baixa; Acima de 18 e menor que 30: Normal; Acima de 30: Alta
Precipitação	Entre 0 e 0.1: Sem chuva; Acima de 0.1 : Com chuva
Temperatura média	Abaixo de 17: Baixa; Acima de 15 e menor que 25: Normal; Acima de 25 : Alta
Umidade relativa do ar média	Abaixo de 75: Baixa; Acima de 75 e menor que 90:Normal, Acima de 90: Alta

Tabela 12: Transformação dos dados climáticos.

4.4 Fase 4 e 5: Modelagem e Avaliação

A fase de modelagem inicia-se com a seleção do algoritmo para mineração de dados. As técnicas de associação foram utilizadas para encontrar regras entre as características das ocorrências e os atendimentos realizados. Também foi escolhido *clustering* para encontrar características que descrevem as ocorrências. O algoritmo de Regras de Associação Apriori e o K-means foram aplicados por serem algoritmos já bem estudados. Além deles, o algoritmo paNDa+ que gera grupos sem ter necessariamente todos os atributos em cada

transação. Visto que cada algoritmo processa os dados de forma diferente, as subseções seguintes irão descrever as atividades de Planejamento de Testes, construção do modelo e avaliação do modelo para seus respectivos algoritmos.

Atributo	Exemplos de valores
id logradouro	4714, 433, 3135
id tp emergencia	ATENDIMENTO PRÉ-HOSPITALAR, ACIDENTE DE TRÂNSITO
Estacao do ano	Primavera, Verão, Outono, Inverno
Horario da ocorrencia	0,1,2,3,4,5...23
Dia do mês	1,2,3,4...31
Bairro	CENTRO, ESTREITO, 32

Tabela 13: Dados das ocorrências utilizados e exemplos de valores.

4.4.1 Modelagem e Avaliação: Associação Apriori

A regra de Associação Apriori tem o objetivo de gerar regras de forma a identificar padrões entre um dado de conjunto de itens, os antecedentes e o consequente. Foram realizados diversos testes através da ferramenta Weka com este algoritmo, sendo que o primeiro teste gerou regras sem utilizar uma classe como consequente e gerou 4000 regras. Outros testes foram realizados utilizando somente as ocorrências dos principais bairros e logradouros, com o objetivo de encontrar características específicas à estes locais. Em ambos os testes, foram analisados os *lifts* das regras que possuíam como consequente algum tipo de emergência, mas, além de poucas regras com esse consequente terem sido geradas, todos os *lifts* gerados eram menores que "1" e, por isso, utilizou-se o tipo de emergência como parâmetro de classe do consequente do algoritmo.

Foram utilizadas todas as ocorrências de Florianópolis para o teste disponibilizado e como parâmetros de entrada utilizou-se o suporte mínimo de 0.05 (cinco porcento), confiança de 30% (trinta porcento), número de regras máximo de 1000 regras e selecionou-se como classe o atributo "id tp emergencia". O modelo gerou 407 regras distintas e, como nem todas as regras geradas são relevantes, selecionou-se as regras mais interessantes levando em consideração o seu suporte e confiança. Um detalhe adicional refere-se ao lift, no qual não é gerado quando se utiliza um atributo de classe como parâmetro para o algoritmo. As seguinte regras foram selecionadas:

- Regra 43. umidade relativa media=Normal (Entre 75% e 90%) suporte:61,7% ==> emergencia="Atendimento pré-hospitalar"confiança:(39.5%). Essa regra possui a maior confiança em relação ao suporte das regras geradas. Indica que a maior parte dos acidentes pré-hospitalar acontecem em dias em que a umidade relativa media esta entre 75 e 90%.

- Regra 70. temperatura maxima=Normal (Entre 18ºC e 30ºC) umidade relativa media=Normal (Entre 75% e 90%) suporte: 47,8% ==> emergencia="Atendimento pré-hospitalar"confiança:39,4%. A regra 43 engloba as transações desta mesma regra, mas esta traz a temperatura máxima.

- Regra 18. temperatura maxima=Normal (Entre 18ºC e 30ºC) temperatura minima=Normal (Entre 15ºC e 25ºC) estacaoDoAno=Verao suporte:14,4% ==> emergencia="Atendimento pré-hospitalar"confiança:39,4%. Esta é a regra com maior suporte e com a informação de estação do ano. Ela é um indicativo de que a estação do ano não tem grande impacto sobre as ocorrências.

- Regra 9. diaDaSemana=Domingo suporte:13.2% ==> emergencia="Atendimento pré-hospitalar"conf:40.2%. É a primeira regra gerada em relação ao dia da semana. Destaca que a maior parte das ocorrências desse tipo acontecem no domingo, mas a quantidade não é tão maior assim em relação aos demais dias. Comparando-a com as regras geradas nos outros dias da semana, o suporte das outras regras fica entre 11% e 14%, tendo uma confiança que varia entre 37% até 39%.

- Regra 374. precipitacao=Com chuva temperatura minima=Normal (Entre 15ºC e 25ºC) suporte: 38,2% ==> emergencia="Acidente de trânsito"conf:23.19%. Essa é a primeira regra onde a presença de chuva aparece como antecedente, mas o interessante em relação as regras geradas com presença de chuva antecedente possuem a emergência "Acidente de Trânsito" como consequente.

4.4.2 Modelagem e Avaliação: K-means

O K-means possuí como objetivo gerar *clusters* de forma a alocar todas as transações(ocorrências) em somente um dos padrões gerados. Foram realizados diversos testes através da ferramenta Weka com este algoritmo.

Inicialmente utilizou-se como dado de entrada somente as ocorrências dos principais logradouros e poucos *clusters* variando entre 3 a 5 dependendo do número de ocorrências do logradouro. Os resultados em relação aos *clusters* gerados nos testes foram similares entre si, e, por isso, realizou-se um outro teste utilizando 11 bairros diferentes, sendo 10 deles os principais bairros mostrados na tabela 10 e o outro bairro sendo "Outros". O resultado foi similar ao encontrado nos *clusters* gerados pelo teste anterior que utilizou os logradouros. Por este motivo, utilizou-se todas as ocorrências de Florianópolis para o teste exibido neste trabalho.

O algoritmo recebeu como parâmetro 11 *clusters* para todas as ocorrências. O motivo de utilizar essa quantidade de grupos é dada pelo fato de possuírem 11 tipos de emergência diferentes e, também, pelo fato de que 10 bairros somados possuírem mais do que 50% de todas as ocorrências.

Característica	Valor	Grupos com o mesmo valor
id bairro	Centro	1,2,3,5,6,8,9,10
id logradouro	3281(FLORIANOPOLIS-CENTRO)	Nenhum
id tp emergencia	5(Atendimento pré-hospitalar)	1,2,4,5,6,8
Precipitacao	Sem chuva	Todos
Temperatura máxima	Alta	Nenhum
Temperatura mínima	Normal	1,2,3,4,6,7,8,9,10
Insolacao	Alta	1,4,5,6,8,10
Temperatura compensada media	Alta	1,4,6
Umidade relativa média	Normal	1,2,3,4,8,9
Velocidade vento média	Baixa	1,3,4,5,6,7,8,9,10
Numero do dia	23	9
Dia da semana	Friday(Sexta-feira)	5,6
Hora	15	8,9
Estacao do ano	Verao	3,6
mes	2(Fevereiro)	Nenhum

Tabela 14: Comparação das características do grupo 0 aos demais.

A figura 16 mostra os grupos gerados a partir de todos os dados das ocorrências. É possível notar, como se era esperado, que maior parte dos padrões gerados possuem a característica "Atendimento pré-hospitalar"(valor 5) em seu "id tp emergencia". Ainda assim, observa-se nos grupo 3,9 e 10 a presença do tipo emergência "Acidente de trânsito"(valor 8) e no grupo 7 o tipo de emergência "Auxílios e apoios"(valor 2). A figura 16 também mostra a quantidade de transações atrelada a cada grupo, sendo os grupos 0 e 1 os mais predominantes possuindo, respectivamente, 15.7% e 19.1% de todas as transações(ocorrências). A tabela 14 mostra as características identificadas no grupo 0 e em quais *clusters* a mesma característica foi identificada, enquanto a tabela 15 mostra as respectivas informações para o grupo 1.

Característica	Valor	Grupos com o mesmo valor
id bairro	Centro	0,2,3,5,6,8,9,10
id logradouro	6631(PAULO FONTES)	3
id tp emergencia	5(Atendimento pré-hospitalar)	0,2,4,5,6,8
Precipitacao	Sem chuva	Todos
Temperatura máxima	Normal	2,3,4,5,6,7,8,9,10
Temperatura mínima	Normal	0,2,3,4,6,7,8,9,10
Insolacao	Alta	0,4,5,6,8,10
Temperatura compensada media	Normal	0,4,6
Umidade relativa média	Normal	0,2,3,4,8,9
Velocidade vento média	Baixa	0,3,4,5,6,7,8,9,10
Numero do dia	12	Nenhum
Dia da semana	Saturday(Sábado)	0,10
Hora	18	3
Estacao do ano	Outono	5
mes	4(Maio)	Nenhum

Tabela 15: Comparação das características do grupo 1 aos demais.

O resultado obtido destaca as características mais comuns da cidade. O K-means utiliza a distância em relação ao centroide como critério para agrupar os atributos e,

como é possível observar nos padrões gerados, as características mais comuns identificadas são referentes às características comuns dos dias na cidade de Florianópolis. Ou seja, o uso dos dados climáticos como característica adicional das ocorrências só destacou as características climáticas mais comuns entre todos os dias da cidade, ao invés de destacar características incomuns que potencializassem as ocorrências, indicando assim, que essas características climáticas diárias não possuem grande influência em relação às ocorrências. É importante lembrar que essas características climáticas referem-se ao dia da ocorrência e não ao momento (hora) em que ela foi atendida. É possível que as informações climáticas ainda sejam fatores importantes para a ocorrência, mas essa informação só poderá ser confirmada realizando outras análises ou talvez utilizando as características da condição climática do instante em que a ocorrência. Os atributos que são vinculados a época da ocorrência não trazem muitas informações também, com exceção da hora. Os padrões gerados possuem horários que variam entre as 09 horas e as 18 horas, destacando o período em que a maioria das ocorrências são atendidas que, ademais, é a faixa de horário onde a maioria das pessoas está fora de casa executando as atividades do dia-a-dia.

4.4.3 Modelagem e Avaliação: PaNDa+

O PaNDa+ possui o objetivo gerar padrões de forma a alocar as transações mais relevantes considerando uma margem de erro.

Foram realizados diversos testes com o algoritmo. Os testes variaram em relação à margem de erro configurada como parâmetro e os dados de entrada, os quais variaram entre todas as ocorrências e ocorrências relativas a tipos específicos de emergências. Quanto maior a margem de erro configurada, mais atributos eram encontrados nos padrões considerados relevantes, enquanto quando a margem de erro era configurada como "0", os padrões possuíam poucas transações vinculadas a eles, mas com atributos sempre presentes nas transações vinculadas. Em outros testes, foram utilizados os dados climáticos no PaNDa+, mas estes apontaram as mesmas características comuns agrupadas pelos *clusters* do K-means. Por este motivo, removeu-se a maioria os dados climáticos para o testes disponibilizado utilizando o PaNDa+, deixando apenas a presença da chuva, e foram adicionado mais informações em relação as datas das ocorrências. Por não possuir métrica para definir a margem de erro para os dados, utilizou-se para disponibilização deste trabalho apenas os testes que usaram as configurações padrão do algoritmo. A configuração padrão do algoritmo utiliza uma margem de erro dinâmica baseada na função de custo, que é calculada a cada iteração do algoritmo. Os resultados do PaNDa+ não foram melhores do que o do K-means. Pelas características que foram utilizadas para as ocorrências, esperava-se encontrar padrões sólidos que indicassem momentos onde as ocorrências acontecessem, como horários do dia ou meses do ano. Foram gerados 3 modelos e os parâmetros e os dados de entrada, para cada modelo, são os seguintes:

Figura 16 – Grupos gerados a partir de todos os dados com k-means.

		Cluster#				
Attribute	Full Data	0	1	2	3	4
	(97754.0)	(13395.0)	(17645.0)	(10130.0)	(11253.0)	(5932.0)
id_bairro	CENTRO	CENTRO	CENTRO	CENTRO	CENTRO	CENTRO
id_logradouro	6631	6631	6631	6073	6073	6631
id_tp_emergencia	5	5	5	5	5	1
precipitacao	Sem chuva	Sem chuva	Com chuva	Sem chuva	Sem chuva	Sem chuva
temp_maxima	Normal	Normal	Normal	Alta	Normal	Normal
temp_minima	Normal	Baixa	Normal	Normal	Normal	Normal
insolacao	Alta	Alta	Baixa	Alta	Alta	Alta
temp_comp_media	Normal	Normal	Normal	Alta	Normal	Normal
umidade_relativa_media	Normal	Normal	Normal	Normal	Normal	Baixa
velocidade_vento_media	Baixa	Normal	Normal	Baixa	Normal	Normal
numeroDia	1	24	8	9	26	7
diaDaSemana	Saturday	Sunday	Tuesday	Thursday	Saturday	Monday
hora	14	14	10	11	10	20
estacaoDoAno	Verao	Inverno	Primavera	Verao	Outono	Inverno
ano	2017	2014	2009	2017	2008	2017
mes	3	8	12	2	4	8

Attribute	5	6	7	8	9	10
	(5912.0)	(11520.0)	(5045.0)	(6371.0)	(3643.0)	(6908.0)
id_bairro	CENTRO	CENTRO	CENTRO	CENTRO	CENTRO	CENTRO
id_logradouro	1127	6073	4331	4331	6631	4726
id_tp_emergencia	5	5	8	8	5	5
precipitacao	Com chuva	Com chuva	Sem chuva	Com chuva	Sem chuva	Sem chuva
temp_maxima	Alta	Normal	Normal	Normal	Alta	Normal
temp_minima	Normal	Normal	Normal	Normal	Normal	Baixa
insolacao	Alta	Baixa	Alta	Alta	Alta	Alta
temp_comp_media	Alta	Normal	Normal	Normal	Alta	Normal
umidade_relativa_media	Normal	Normal	Normal	Baixa	Normal	Baixa
velocidade_vento_media	Normal	Baixa	Normal	Baixa	Normal	Baixa
numeroDia	18	2	26	5	4	6
diaDaSemana	Wednesday	Saturday	Saturday	Sunday	Wednesday	Tuesday
hora	10	17	18	19	15	14
estacaoDoAno	Verao	Primavera	Inverno	Outono	Verao	Outono
ano	2017	2013	2010	2012	2014	2017
mes	1	11	6	4	1	6

4.4.3.1 Modelo 1 : Todas as ocorrências

O primeiro modelo recebeu como parâmetro o padrão do algoritmo e os dados de entrada desse modelo são todas as ocorrências atendidas. O modelo possuí o objetivo de identificar as características mais comuns das ocorrências com a sua respectiva proporção

em relação a todas as ocorrências. O algoritmo também aloca a mesma transação em agrupamentos diferentes. Dessa forma, a soma das transações alocadas em cada grupo pode ultrapassar 100%, já que a mesma transação pode aparecer em um ou mais grupos gerados. Essa característica é uma das mais importantes ao comparar o Panda+ com o K-means, no qual obriga que cada transação esteja em apenas um dos grupos gerados. A tabela 16 mostram os padrões encontrados no primeiro modelo em relação às informações de todas as ocorrências e, através dela, pode-se notar o seguinte:

- O padrão "1" destacou que 70.28% das transações podem ser vinculados às características "Temporada:Comum", "TipoDoDIa:DiaUtil"e "Precipitacao: Sem chuva". Que é uma informação muito similar ao que foi encontrado em relação ao K-means, o fato de que as ocorrências acontecem independente das características selecionadas, e sim, pelo fato de que a maioria dos dias em que as ocorrências foram atendidas são dias comuns.

- O padrão "2" destacou que 15.61% das ocorrências possuem a característica de ser um atendimentos pré-hospitalar e possuir um dia com chuva.

- O padrão "3" destacou que 19.66% das ocorrências possuem a característica de estarem na "Estacao:Verao", que é praticamente o mesmo período da "Temporada:AltaTemporada"de Florianópolis, na qual o verão faz referência ao meses de "Dezembro, Janeiro e Fevereiro"enquanto a "AltaTemporada"só abrange os meses de "Janeiro"e "Fevereiro". A informação do "TipoDoDia:DiaUtil"só indica que o final de semana não é um fator que aumenta o número de ocorrências.

4.4.3.2 Modelo 2 : Ocorrências do tipo "Atendimento pré-hospitalar"

O segundo modelo recebeu como parâmetro o padrão do algoritmo e os dados de entrada desse modelo são todas as ocorrências do tipo "Atendimento pré-hospitalar". O modelo possuí o objetivo de identificar as características mais comuns em relação ao tipo de ocorrência selecionada, indicando a sua respectiva proporção em relação a todas as ocorrências. A tabela 17, mostra os padrões encontrados em relação ao segundo modelo em relação aos atendimentos pré-hospitalar da cidade:

- O padrão "1" destacou que 46.61% das transações podem ser vinculados às características "ATENDIMENTO PRÉ-HOSPITALAR"e "Precipitacao: Com chuva". Como todos os dados usados para esse teste são do tipo "Atendimento pré-hospitalar", não é uma característica relevante para esse agrupamento. Por outro lado, a informação "Precipitacao: Com chuva"reforça a descoberta feita no teste com todas as ocorrências, indicando que a chuva realmente aumenta o número de atendimentos pré-hospitalares.

- O padrão "2" destacou que 8.87% das ocorrências possuem a característica de ser "TipoDoDia:FinalDeSemana"e "Estacao:Verao", permitindo a interpretação de que durante o verão as ocorrências de atendimento pré-hospitalar acontecem mais durante os finais de semana comparado-se aos dias úteis. Fato que também é reforçado pelo padrão "3", dessa vez para o inverno.

4.4.3.3 Modelo 3 : Ocorrências do tipo "Acidente de trânsito"

O terceiro modelo recebeu como parâmetro o padrão do algoritmo e os dados de entrada desse modelo são todas as ocorrências do tipo "Acidente de trânsito".O modelo possuí o objetivo de identificar as características mais comuns em relação ao tipo de ocorrência selecionada, indicando a sua respectiva proporção em relação a todas as ocorrências. A tabela 18, mostra os padrões encontrados ao terceiro modelo em relação aos acidentes de trânsito da cidade:

- O padrão "1" destacou que 25.41% das transações podem ser vinculados às características "ACIDENTE DE TRÂNSITO" e "Precipitacao: Com chuva". Como todos os dados usados para esse teste são do tipo "ACIDENTE DE TRÂNSITO", não é uma característica relevante para esse agrupamento. Contudo, a informação "Precipitacao: Com chuva", indica que essa ocorrência é potencializada pela presença de chuva.

Numero do padrão	Característica 1	Característica 2	Característica 3	Característica 4	Quantidade agrupada
1	Temporada:Comum	TipoDoDia:DiaUtil	Precipitacao:Sem chuva		70.28%
2	Precipitacao:Com chuva	ATENDIMENTO PRÉ-HOSPITALAR			15.61%
3	Estacao:Verao	Temporada:AltaTemporada	TipoDoDia:DiaUtil		19.66%
4	Precipitacao:Com chuva	Estacao:Outono	Temporada:Comum		13.16%
5	TipoDoDia:FinalDeSemana	Estacao:Inverno	DiaDaSemana:Sabado	Temporada:Comum	12.36%

Tabela 16: Padrões gerados pelo PaNDa+ relação a todas as ocorrências.

Numero do padrão	Característica 1	Característica 2	Característica 3	Quantidade agrupada
1	ATENDIMENTO PRÉ-HOSPITAL	Precipitacao: Com chuva		46.61%
2	TipoDoDia:FinalDeSemana	Estacao:Verao	ATENDIMENTO PRÉ-HOSPITALAR	8.87%
3	Estacao:Inverno	TipoDoDia:FinalDeSemana	TipoDoDia:DiaUtil	5.09%

Tabela 17: Padrões gerados pelo PaNDa+ relação aos atendimentos pré-hospitalar.

Numero do padrão	Característica 1	Característica 2	Característica 3	Quantidade agrupada
1	ACIDENTE DE TRÂNSITO	Precipitacao: Com chuva		25.41%
2	TipoDoDia:FinalDeSemana	Estacao:Outono	ACIDENTE DE TRÂNSITO	4.1%

Tabela 18: Padrões gerados pelo PaNDa+ relação aos atendimentos de acidentes de trânsito.

4.5 Fase 6: Disponibilização

Esta etapa faz referência a Disponibilização dos resultados obtidos e foi feita através da produção deste trabalho de conclusão de curso. Utilizou-se tanto a aplicação desenvolvida para verificar e investigar as informações encontradas durante todo o processo de *data mining*.

5 Conclusão e Resultados

Este trabalho teve como objetivo realizar a coleta de dados e estudar as ocorrências atendidas pelo corpo de bombeiros através do processo de mineração de dados para a descoberta de padrões. A partir de dados como: registros dos atendimentos, características climáticas das datas das ocorrências, informações referente a estação do ano e horário do dia; foi feito o entendimento dos dados, levantando informações relevantes que pudessem auxiliar no processo de *data mining*, assim como informações importantes sobre os lugares onde as ocorrências mais ocorrem, como a identificação dos principais bairros e logradouros. Em seguida foi realizada a preparação dos dados para que pudessem ser utilizados pelos algoritmos K-means, Associação Apriori e PaNDa+.

O projeto enfrentou muitas dificuldades em etapas diferentes. O primeiro desafio foi em relação aos próprios registros dos bairros e logradouros. Muitos estavam duplicados, com erros de digitação e até inexistentes. Visto que muitos dos logradouros fazem referência a áreas abrangentes como bairros inteiros e praças, para não desperdiçar as informações das ocorrências atendidas nesses locais, dedicou-se muito tempo para ajustar esses dados, principalmente em relação aos bairros, onde foram desenhados polígonos para os que não foram encontrados por api. Depois de finalizar a aplicação e notar a incoerência de muitos dados, desejou-se verificar com os próprios fornecedores informações em relação ao cadastro de ocorrências. Observou-se que os dados fornecidos são generalizados. Por exemplo, "Salvamento e Resgates"na base oficial é detalhada como "Afogamento", "Alagamento", "Colisão/Choque", entre outros.

O principal objetivo de verificar a origem dos dados foi obter a geolocalização das ocorrências atendidas fornecidas em um segundo momento através de 14 planilhas diferentes as quais tiveram de ser unidas. Cada planilha fazia referência a um batalhão da cidade de Florianóplis. Esse exercício teve que ser feito com cuidado pois cada planilha tinha um conjunto diferente de informações. A dificuldade seguinte foi em relação a modelagem das técnicas utilizadas para a mineração de dados. Foram utilizadas diversas técnicas diferentes de classificação, *clustering*, entre outras para encontrar padrões dentro dos dados disponibilizados. A grande maioria dos resultados obtidos foram infrutíferos, provavelmente pela granularidade muito generalizada. Portanto as informações mais relevantes extraídas durante o projeto foram durante na etapa de exploração de dados, etapa revisitada muitas vezes. Em conclusão, as principais descobertas do trabalho são as seguintes:

- O bairro "Centro" é a região que mais atende chamados de emergência durante o ano todo.
- O bairro "Canasvieiras" é o único bairro com aumento significativo de ocorrências durante o Verão.

- Os dez bairros com mais ocorrências sâo: Centro, Lagoa da Conceição, Estreito, Capoeiras, Trindade, Barra da Lagoa, Canasvieiras, Agronômica, Coqueiros e Rio Vermelho;
- A rua "Paulo Fontes"possui o maior número de ocorrências do tipo "Acidente de trânsito".
- O logradouro "Renato Ântônio de Souza"possui o maior número de ocorrências do tipo "Auxílios e Apoios";
- Os logradouros com maior número de ocorrências são: Renato Antônio de Souza, Paulo Fontes, Florianópolis-Centro, Jornalista Rubens de Arruda Ramos, Lauro Linhares, Santos Saraiva, SC-405, Governador GUstavo Richard, BR-282, Ivo Silveira, Mauro Ramos, João GUalberto Soares, Canasvieiras, Barra da Lagoa, Baldicero Filomeno.
- A maior parte dos atendimentos são do tipo "Atendimento pré-hospitalar" e "Acidente de trânsito". A corporação deve estar preparada para atender uma demanda maior de ocorrências desse tipo do que das demais.
- A utilização dos dados climáticos do dia para atendimentos das emergências do Corpo de Bombeiros de Santa Catarina não é recomendada para trabalhos posteriores.
- A maior parte dos acidentes de trânsito ocorrem as 10 e 22 horas, sendo mais comuns entre as 17 e 18 horas.

Referências

ADRIAANS, P.; ZANTINGE, D.; (FIRM), S. Book; Book/Illustrated. *Data mining*. [S.l.]: Harlow, England ; Reading, Mass. : Addison-Wesley, 1997. Includes index. ISBN 0201403803. Citado na página 23.

AGRAWAL, R.; SRIKANT, R. Fast algorithms for mining association rules. In: *Proc. of 20th Intl. Conf. on VLDB*. [S.l.: s.n.], 1994. p. 487–499. Citado na página 27.

BROWN, M. S. *Data Mining For Dummies*. [S.l.]: For Dummies, 2014. v. 382. Citado na página 21.

C., A. C.; K., R. C. *Data Clustering: Algorithms and Applications*. [S.l.]: Chapman e Hall/CRC, 2013. v. 652. ISBN 1466558210 9781466558212. Citado na página 24.

CASAN. *População de Florianópolis quase triplicou no Réveillon*. 2016. <http://rcnonline.com.br/populaç˜ao-de-florianópolis-quase-triplicou-no-réveillon-aponta-casan-1.1955463>. Accessed: 2019-05-11. Citado na página 38.

CHAPMAN, P. et al. *CRISP-DM 1.0 Step-by-step data mining guide*. [S.l.], 2000. Disponível em: <http://www.crisp-dm.org/CRISPWP-0800.pdf>. Citado na página 21.

CLAUDIO, L.; SALVATORE, O.; RAFFAELE, P. A unifying framework for greedy mining approximate top-k binary patterns and their evaluation. ISTI-CNR, v. 14, p. 27–34, 2013. Citado na página 26.

DATASET. *Flask-Dataset*. 2019. <https://flask-dataset.readthedocs.io/en/latest/>. Accessed: 2019-05-11. Citado na página 29.

FLASK. *Welcome | Flask(A Python Microframework)*. 2019. <http://flask.pocoo.org/>. Accessed: 2019-05-11. Citado na página 29.

HAN, J.; KAMBER, M.; JIAN, P. *Data Mining Concepts and Techniques*. [S.l.]: Elsevier, 2012. v. 703. Citado na página 21.

JUNJIE, W. *Advances in K-means Clustering: A Data Mining Thinking*. [S.l.]: Springer Publishing Company, 2012. v. 194. Citado na página 26.

MATT, J. W. et al. Climate-induced variations in global wildfire danger from 1979 to 2013. Nature Communications, v. 6, n. 7537, p. 387–404, 2015. Citado na página 15.

OPENSTREETMAP. *OpenStreetMap*. 2019. <https://www.openstreetmap.org>. Accessed: 2019-05-11. Citado na página 21.

PANG-NING, T.; MICHAEL, S.; VIPIN, K. *Introdução ao Data Mining. Rio*. [S.l.]: Ciência Moderna, 2009. v. 900. Citado 3 vezes nas páginas 15, 24 e 25.

PYTHON. *Welcome to Python.org*. 2019. <https://www.python.org/>. Accessed: 2019-05-11. Citado na página 29.

USAMA, F.; GREGORY, P.-S.; PADHRAIC, S. The kdd process for extracting useful knowledge from volumes of data of legal services. Acm, v. 39, n. 11, p. 27–34, 1996. Citado na página 21.

Apêndices

APÊNDICE A – ARTIGO DO TRABALHO DE CONCLUSÃO DE CURSO

Análise De Dados Do Corpo De Bombeiros De Florianópolis E Implementação De Um Mapa Informativo De Ocorrências

Luan L. Vieira[1]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88040-900 – Florianópolis – SC – Brasil

Resumo. *Este artigo pretende analisar os registros de dados das ocorrências atendidas pelo corpo de bombeiros da cidade de Florianópolis-SC. Para isto foram coletados dados referentes as condições climáticas das datas das ocorrências junto com os registros das ocorrências. O objetivo desta coleta é estudar as ocorrências através do processo de mineração de dados para a descoberta de padrões entre as ocorrências e, também, a implementação de um mapa informativo para visualização das ocorrências. As técnicas de mineração de dados utilizadas neste artigo foram a clusterização(agrupamento) e regras de associação, para que se possa obter como resultado padrões que possam influenciar no acontecimento das ocorrências.*

1. Introdução e Motivação

Os algoritmos de mineração de dados são antigos. É difícil marcar uma data inicial para a sua concepção, visto que eles nasceram de um crescimento natural de estudos estatísticos e com o próprio avanço da computação. Ainda assim, há pouco tempo, o mundo vivia uma época onde havia muita informação armazenada e os próprios portadores dessa informação, sejam eles empresas ou instituições, eram incapazes de analisá-las. Hoje, apesar do volume de dados continuar crescendo em uma proporção maior, a realidade já não é mais a mesma. As técnicas para interpretar os dados de forma útil estão sendo cada vez mais disseminadas, tanto na área científica como na área tecnológica. Existe ainda um substancial volume de informação inutilizada, porém, muitas das informações que nos eram inúteis tornaram-se essenciais e com elas tornou-se possível prever comportamentos, identificar tendências e encontrar correlações através do processo de mineração de dados, com o objetivo de extrair novos conhecimentos de forma mais acessível aos seus usuários[Pang-ning et al. 2009].

Um exemplo de informação disponível para ser minerada são registros das ocorrências do corpo de Bombeiros de Santa Catarina . As informações registradas pela corporação abrangem diversos tipos de ocorrências como incêndios, resgates, salvamentos, entre outros. Nesses registros, é possível encontrar informações como cidade, horário do dia, data, tipo da ocorrência e logradouro. Utilizando essas informações, é possível encontrar características sobre as ocorrências atendidas. O estudo [Matt et al. 2015] mostra que, nos Estados Unidos, incêndios florestais acontecem com maior frequência em dias sem chuvas e secos, contudo, considerando toda a dinâmica da sociedade, o quanto que cada uma dessas características, como a ausência de precipitação e de umidade no ar, são um indicativo de um provável incêndio em Florianópolis? Será que algum dia da semana pode ser um fator de uma ocorrência? Ou talvez uma época do ano? Será que é possível agrupar as ocorrências em relação a comportamentos comuns entre elas? É difícil afirmar que essas características são as que realmente levaram à um incidente, mas podem ser indicadores de onde pode ser útil realizar análises.

O objetivo deste trabalho é analisar os dados das ocorrências registradas pela unidade de Corpo de Bombeiros de Santa Catarina. A base disponibilizada possui informações relacionadas a diversos tipos de ocorrência, a qual varia de atendimentos pré hospitalares até controle de insetos e pragas. Neste trabalho serão explorados dois tipos de análise : (i) analise espacial para visualizar a distribuição geográfica das ocorrências e (ii) mineração de dados.

2. Conceitos básicos e contextualização dos dados

Esta seção é dedicada para a descrição dos dados e algoritmos de mineração utilizados na elaboração deste artigo.

2.1. Dados

Foram utilizados três conjuntos de dados diferentes para a elaboração deste trabalho. O conjunto de dados da seção 2.1.1 referem-se aos dados da tabela ”Ocorrências”disponibilizada pela corporação, os dados da seção 2.1.2 fazem referência aos dados climáticos e o conjunto de dados da seção 2.1.3 fazem referência ao mapeamento geográfico realizado a partir dos nomes dos bairros e logradouros.

2.1.1. Dados das ocorrências

A base de dados disponibilizada pelo Corpo de Bombeiros de Santa Catarina possui registros de ocorrências de fevereiro de 2007 até abril de 2018. A tabela das ocorrências possui 1.814.563 registros, com atributos como a descrição inicial da ocorrência, tipo da emergência, data de inicio, data de finalização e um logradouro vinculado a ocorrência. Em relação à consistência dos atributos, apenas os atributos do logradouro, data da ocorrência, horário e tipo da emergência estão registrados em praticamente todas as ocorrências, enquanto os demais atributos possuem informações incompletas, vazias e de texto aberto. Visto que o volume de dados para o estado inteiro é muito grande, e principalmente porque os padrões de ocorrências podem variar de uma região para outra no estado, optou-se por analisar apenas a cidade de Florianópolis. Neste conjunto de dados observou-se que praticamente todos os dias alguma ocorrência foi registrada.

Definidas as informações que serão utilizadas, é necessário destacar os tipos de emergência que foram atendidos pela corporação, visto que as análises deste trabalho irão partir desses tipos de emergência. A base de dados contém os tipos de emergência listados na tabela 1.

Incêndio
Auxílios e apoios
Produtos perigosos
Salvamento, busca e resgate
Atendimento pré-hospitalar
Ocorrência não atendida
Diversos
Acidente de trânsito
Ações preventivas
Averiguação e corte de árvore
Averiguação e manejo de inseto

Tabela 1: Tipos de emergências.

2.1.2. Dados climáticos

Com o intuito de extrair informações que possam ser relevantes em relação as ocorrências, utilizou-se dados externos aos dados disponibilizados. Levantou-se a hipótese de que as características meteorológicas da data da ocorrência poderiam ser fatores relevantes para o acontecimento da ocorrência como, por exemplo, um maior número de acidentes de trânsito em dias mais chuvosos, incêndios em dias mais quentes e secos, afogamentos em épocas de veraneio, ataques por insetos em épocas de maior calor e umidade, etc. Por isso, utilizou-se registros climáticos diários para correlacioná-los com as ocorrências atendidas. Os dados utilizados foram extraídos do Banco de Dados Meteorológicos para Ensino e Pesquisa (BDMEP), sendo importante ressaltar que cada registro do BMEP é referente a um período de 24 horas. Através do BMEP, foram coletadas as informações climáticas de Florianópolis no período de fevereiro de 2007 até abril de 2018 e as características climáticas extraídas estão listadas na tabela 2.

Insolação
Temperatura mínima
Temperatura máxima
Precipitação
Temperatura média
Umidade relativa do ar média

Tabela 2: Características climáticas utilizadas.

2.1.3. Dados das ocorrências com geolocalização

Os dados disponibilizados inicialmente pelo corpo de bombeiros possuíam apenas o nome dos logradouros, sem a localização geográfica.Para obter a localização geográfica aproximada, foi utilizado o Open Street Maps [OpenStreetMap 2019], fazendo um *matching* entre os nomes dos logradouros da base de dados das ocorrências e os logradouros do Open Street Maps, buscando o centroide do logradouro. Optou-se por extrair o centroide do logradouro porque a base de dados fornecida inicialmente não possui a informação do numero da rua onde aconteceu a ocorrencia. Para definir as delimitações dos bairros utilizou-se a mesma *api* para encontrar os polígonos referentes a geometria dos bairros.

2.2. Principais Técnicas de Mineração de Dados

Esta seção contém uma descrição das técnicas de mineração utilizada na elaboração deste artigo.

2.2.1. Agrupamento ou *clustering*

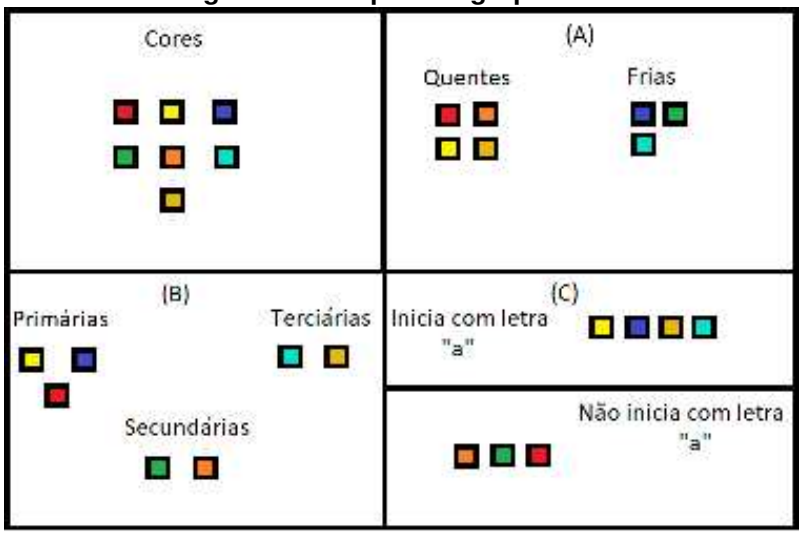
A técnica de agrupamento é citada por [Pang-ning et al. 2009] como uma técnica capaz de unir registros baseando-se nos valores dos atributos dos registros e seus relacionamentos. A união dos registros, chamada de agrupamento, é citada por Charu C. Aggarwal[Charu C. and Chandan K. 2013] como um modelo conciso onde os dados podem ser interpretados no formato de um resumo. Outra forma de identificar um problema de agrupamento, de acordo com Charu C. Aggarwal, pode ser descrita da seguinte forma: Dado um conjunto de registros, particione-os em grupos que sejam o mais semelhantes possível entre si.

O objetivo do uso de técnicas de agrupamento é o de identificar grupos similares e destacá-los dos outros conjuntos. Segundo [Pang-ning et al. 2009] a qualidade do agrupamento gerado depende da semelhança dos dados dentro do seu próprio grupo e a diferença entre os dados de grupos diferentes. Quanto maior a semelhança dentro de um grupo e quanto maior a diferença entre os outros grupos, melhor a qualidade dos *clusters*. Sendo assim, dependendo do critério de agrupamento, um conjunto de dados pode ser agrupado de diversas formas. Para exemplificar as diversas formas que um dado conjunto pode ser agrupado, a tabela 3 mostra um exemplo de cores e a figura 2 mostra um conjunto (A) onde as cores foram agrupadas por sua temperatura, um conjunto (B) onde as cores foram agrupadas por seu tipo de cor e um conjunto (C) as cores foram agrupadas pelo fato de possuir a letra inicial ”a”ou não.

Cor	Temperatura	Tipo de cor
Vermelho	Quente	Primária
Amarelo	Quente	Primária
Azul	Fria	Primária
Verde	Fria	Secundária
Laranja	Quente	Secundária
Azul-esverdeado	Fria	Terciária
Amarelo-alaranjado	Quente	Terciária

Tabela 3: Cores e algumas características.

Figura 1. Exemplo de agrupamento



Um dos algoritmos mais conhecidos de agrupamento é o K-means . Segundo [Pang-ning et al. 2009], o K-means é dito como uma técnica de agrupamento particional, ou seja, um dado registro só é vinculado a um único grupo. O k-means utiliza como parâmetro o número de grupos que deseja gerar, sendo este número o valor ”k” de clusters. O número de grupos muitas vezes é difícil de estimar. O algoritmo inicia selecionando aleatoriamente ”k” registros da base de dados como sendo o centroide de cada ”k” *clusters* na primeira iteração. As iterações seguintes identificam os registros que foram atrelados a cada um dos *clusters* gerados e atualiza o valor do novo centroide baseando-se nos registros agrupados na iteração anterior. Como o centroide foi modificado na iteração atual, ele verifica os registros que agora estão mais próximos do centroide atual para gerar o novo cluster. Esse procedimento é repetido até que o centroide não seja modificado entre as iterações. Cada atributo do grupo será representado por um ponto central que irá representar o atributo no grupo, sendo esta a origem do nome do algoritmo onde o *means* significa média. O fato de utilizar a média como métrica torna o agrupamento problemático em alguns casos. Segundo [Junjie 2012], a chegada do *big data* tem dificultado o uso do k-means por si só, pelo fato das informações possuírem muitos fatores, muitas dimensões e, também, ruído.

O Panda é um algoritmo criado para mineração de itens frequentes [Claudio et al. 2013]. Ele gera conjuntos de itens comprados em conjunto nas mesmas transações. O PaNDa+ trabalha com uma matriz binária onde as linhas são as transações e as colunas são os itens. Este algoritmo realiza diversas iterações tentando encontrar o melhor conjunto de características (itens) que estão presentes em um conjunto de transações.

Este algoritmo será usado pela primeira vez para fazer *clustering*, pois ele agrupa linhas e colunas e permite que a mesma linha faça parte de mais de um grupo e que cada grupo possa conter diferentes colunas. Desta maneira, o Panda, se utilizado para clustering, permite gerar grupos heterogêneos do ponto de vista do número de atributos, ou seja, grupos que contenham atributos diferentes. Sob este aspecto, o Panda deve gerar clusters melhores do que o K-means, o qual gera grupos onde todos os atributos participam de um cluster, fazendo a média das distâncias de todos os atributos.

O Panda utiliza uma função de custo, que é um dos parâmetros utilizados para medir a qualidade dos padrões gerados e como critério de parada para encontrar o melhor conjunto de características que caracterize os dados. Os critérios de parada mais utilizados para esse algoritmo são um número pré-definido de padrões ou quando a função de custo não for modificada em relação as iterações anteriores. O ruído nesse algoritmo é tratado como "margem de erro", de forma que cada conjunto de características aceita uma tolerância de erro. Esse percentual pode ser definido nas configurações do algoritmo, mas ele também é calculado dinamicamente através da função de custo, otimizando assim a busca dos melhores padrões.

2.2.2. Regras de associação

Regras de associação são técnicas de aprendizado de máquina que possuem o objetivo de gerar regras a partir das relações entre os atributos de um conjunto de dados. A técnica foi criada para mineração de conjunto de itens de supermercados e foi introduzido por[Agrawal and Srikant 1994] ao descobrir relações entre compras de produtos nos registros das vendas de supermercados. O algoritmo de regras de Associação mais conhecido é o Apriori proposto em 1994 por Agrawal e Skrikant, definindo um problema de regra de associação como sendo:

- Seja I = $\{i_1, i_2 ..., i_n\}$ um conjunto de n atributos chamado de itens.
- Seja D = $\{t_1, t_2 ..., t_m\}$ um conjunto de m transações chamado de base de dados.
- Cada transação no conjunto D possui seu identificador e contém um subconjunto de itens. Uma regra é definida como sendo uma implicação da forma:

$$X \Rightarrow Y, onde X e Y \subseteq I \tag{1}$$

- Cada regra é composta por dois grupos diferentes de itens, também conhecidos como *itemsets*, X e Y, onde X é chamado de antecedente e Y é chamado de consequente. Os algoritmos de regra de associação geram muitas regras e possuem formas para avaliar-las de forma a identificar regras fortes a partir de critérios de interesse. Alguns deles são:
- Suporte: O suporte é uma indicação de quão frequente um *itemset* aparece no conjunto de dados. Considerando **X e Y** como *itemsets*, $X \Rightarrow Y$ como uma regra de associação e **D** sendo um conjunto de transações de uma base de dados fornecida, o suporte de **X** em relação a **D** é definido como a proporção de transações **t** no conjunto de dados que contém o *itemset* **X**. E é definido formalmente por:

$$supp(X) = \frac{|X|}{|D|} \tag{2}$$

- As regras de associação são geradas a partir de *itemsets* frequentes. Uma medida usada para avaliar as regras é a confiança. A confiança é interpretada como uma estimativa da probabilidade condicional $P(E_Y|E_X)$, sendo esta a probabilidade de encontrar o consequente da regra em transações sob a condição de que essas transações também contenham o antecedente. O suporte de uma regra $X \Rightarrow Y$, em relação ao conjunto de transações **D**, é a proporção de transações que contém **X** e também contém **Y**. A confiança é definida por:

$$conf(X \longrightarrow Y) = supp(X \cup Y)/supp(X) \tag{3}$$

- Lift: O lift mede a importância das regras levando em consideração a frequência do consequente geradas utilizando a confiança da regra dividida pelo suporte do consequente. Quanto maior o lift, melhor a qualidade da regra. O lift é definido por:

$$Lift = \frac{c(X \longrightarrow Y)}{supp(Y)} \tag{4}$$

3. Materiais e Métodos Utilizados Para o Processo de Mineração de Dados

As subseções desta seção foram elaboradas com base do CRISP-DM. A etapa de Entendimento do Negócio é referenciada na introdução e motivação deste artigo e a continuidade das etapas será descrita nas subseções seguintes.

3.1. Entendimento dos Dados

Utilizou-se os dados fornecidos pela Unidade de Tecnologia e Informação do corpo de bombeiros e os dados climáticos do Banco de Dados Meteorológicos para Ensino e Pesquisa. Foram selecionadas algumas informações para ilustrar as informações disponibilizadas pelo corpo de bombeiros:

- O histórico das ocorrências atendidas possui 1.814.563 registros. Alguns dos atributos mais relevantes são os locais onde o incidente foi atendido e os identificadores do tipo da ocorrência, na qual define a atividade realizada durante o atendimento.
- Os lugares registrados nas ocorrências são relativos aos nomes dos logradouros. Estes logradouros podem estar vinculados com apenas um bairro e uma única cidade. Dessa forma, se um logradouro cruzar dois ou mais barros, ele terá dois ou mais identificadores diferentes registrados.
- Existem 104.425 logradouros, 8.118 bairros e 295 cidades registradas.
- As ocorrências de Florianópolis somam 109.279 registros.
- Existem 124 bairros vinculados a Florianópolis.
- Existem 5049 logradouros vinculados a Florianópolis.

Na atividade de exploração dos dados foram analisados os registros para identificar quais atributos serão úteis. Como foram utilizados dados climáticos e registros de ocorrências, cada conjunto de dados será descrito separadamente. Os dados climáticos obtidos possuem qualidade boa, visto que a maioria dos registros estão devidamente cadastrados. Contudo, eles se relacionam ao dia e não as horas do dia. Além dos atributos

Bairro	Ocorrências
CENTRO	19066
LAGOA DA CONCEIÇÃO	5847
ESTREITO	5501
CAPOEIRAS	4661
TRINDADE	4347
BARRA DA LAGOA	3588
CANASVIEIRAS	3514
AGRONOMICA	3134
COQUEIROS	2668
RIO VERMELHO	2564
Total:	54890

Tabela 4 : Bairros com os maiores índices de ocorrências.

dos dados climáticos estarem cadastrados de forma consistente, esses registros são todos do tipo numérico provindos de medidores de umidade, velocidade do vento, e seus respectivos sensores. A hipótese levantada supõe que as características climáticas possam ser um dos fatores que influenciam as ocorrências. Mudando a perspectiva dos dados climáticos para os registros das ocorrências, pode-se ver através da tabela 4 que 54.890 de todas as ocorrências de Florianópolis, totalizando 50,23% do total de ocorrências da cidade, estão vinculadas com somente 10 bairros. Conforme mostra a tabela 5, quinze dos 5.049 logradouros possuem 17,48% das ocorrências, indicando os principais logradouros. Também é possível notar que alguns logradouros fazem referência ao próprio bairro, visto que na lista dos nomes dos logradouros com maior número de ocorrências encontram-se os nomes ”FLORIANOPOLIS-CENTRO”, ”CANASVIEIRAS”e ”BARRA DA LAGOA”.

Logradouro	Quantidade de ocorrências	Porcentagem sobre total
RENATO ANTÔNIO DE SOUZA - 378665	2324	2,13%
PAULO FONTES	2097	1,92%
FLORIANOPOLIS-CENTRO	2031	1,86%
JORNALISTA RUBENS DE ARRUDA RAMOS	1770	1,62%
LAURO LINHARES	1360	1,24%
SANTOS SARAIVA	1273	1,16%
SC - 405	1136	1,04%
GOVERNADOR GUSTAVO RICHARD	1093	1,00%
BR 282	1083	0,99%
IVO SILVEIRA	999	0,91%
MAURO RAMOS	924	0,85%
JOÃO GUALBERTO SOARES - 1231	883	0,81%
CANASVIEIRAS	714	0,65%
BARRA DA LAGOA	712	0,65%
BALDICERO FILOMENO	707	0,65%
Total:	19106	17,48%

Tabela 5: Distribuição das ocorrências em relação aos principais logradouros.

Ainda durante a exploração de dados, é possível perceber que os tipos de emergência não são distribuídas igualitariamente e existem ocorrências atendidas mais frequentemente que outras conforme mostra a tabela 6. É possível notar nessa tabela que as atividades de ”ATENDIMENTO PRÉ-HOSPITALAR” e ”ACIDENTE DE TRÂNSITO” correspondem a 63,15% do total de ocorrências da cidade.

Todas as características climáticas obtidas serão utilizadas para averiguar a

Tipo de emergencia	Ocorrências	Representação do total
ATENDIMENTO PRÉ-HOSPITALAR	43353	39,67%
ACIDENTE DE TRÂNSITO	25660	23,48%
DIVERSOS	12189	11,15%
AUXÍLIOS / APOIOS	9390	8,59%
INCÊNDIO	8555	7,83%
AÇÕES PREVENTIVAS	3110	2,85%
SALVAMENTO / BUSCA / RESGATE	2124	1,94%
AVERIGUAÇÃO / CORTE DE ÁRVORE	2085	1,91%
AVERIGUAÇÃO / MANEJO DE INSETO	1673	1,53%
OCORRÊNCIA NÃO ATENDIDA	980	0,90%
PRODUTOS PERIGOSOS	160	0,15%

Tabela 6: Tipo de emergência por quantidade de ocorrências.

hipótese levantada, pois possuem qualidade boa. Já, para os registros das ocorrências, grande parte dos atributos registrados pelos bombeiros são nulos e alguns outros são de texto aberto, indicando que nem todos os atributos são úteis para atingir os objetivos.

3.2. Preparação dos dados

Foram utilizados apenas os atributos ”id logradouro”, ”id tp emergencia”, ”dt ocorrencia”, ”tm ocorrencia”dos registros de ocorrências da cidade de Florianópolis para realizar a mineração, pois estes dados eram os únicos cadastrados de forma consistente. Foi considerado consistente os atributos que estavam devidamente cadastrados em mais de 75% dos registros.

A limpeza dos dados, que é a atividade seguinte, foi realizada nos dados climáticos, pois existiam valores muito distantes dos demais, como ventos com velocidade acima de 200 km/h. No caso dos dados das ocorrências, foram retiradas 1.930 ocorrências por não possuírem o ”id logradouro”, totalizando 107.349 registros para os algoritmos de mineração. Depois de efetuada a limpeza, foi realizada a integração dos dados através do atributo ”dt ocorrencia” para vincular os dados das ocorrências com os registros climáticos.

Para a finalização da fase de preparação de dados, realizou-se a transformação dos dados. Os dados climáticos foram discretizados de acordo com a tabela e os dados das ocorrências tiveram um processo diferente de transformação. A partir das datas e horários das ocorrências, foram extraídos a sua estação do ano, o dia da semana, o dia do mês e a faixa de horário em que a ocorrência foi registrada conforme a tabela 8. Contudo, o algoritmo paNDa+ requer transformações aos dados que vão além das mostradas pelas tabelas 7 e 8. Para adequar os dados ao paNDa+ transformou-se cada instância de cada atributo em um identificador único, de forma que cada transação(ocorrência) será dada por um *array* de identificadores dos respectivos valores de seus atributos. Essa informação será utilizada para montar a matriz binária que é utilizada pelo algoritmo. Para ilustrar a transformação realizada, a informação ”Hora:00” recebeu o identificador ”69” e ”DiaDaSemana:Segunda-feira” recebeu o valor ”62”. Além disso, para o PaNDa+, foram removidas as informações da maioria dos dados climáticos, preservando somente a precipitação e, também, foram adicionados as informações ”Tipo do dia da ocorrência” que foram categorizadas entre ”Dia Útil” e ”Final de semana” e ”Tipo de época do ano”que foram categorizadas entre ”Alta temporada” e ”Comum” .

Atributo	Transformação
Insolação	Entre 0 e 5: Baixa; Acima de 5: Alta
Temperatura mínima	Abaixo de 15: Baixa; Acima de 15 e menor que 25: Normal; Acima de 25 : Alta
Temperatura máxima	Abaixo de 18: Baixa; Acima de 18 e menor que 30: Normal; Acima de 30: Alta
Precipitação	Entre 0 e 0.1: Sem chuva; Acima de 0.1 : Com chuva
Temperatura média	Abaixo de 17: Baixa; Acima de 15 e menor que 25: Normal; Acima de 25 : Alta
Umidade relativa do ar média	Abaixo de 75: Baixa; Acima de 75 e menor que 90:Normal, Acima de 90: Alta

Tabela 7: Transformação dos dados climáticos.

Atributo	Exemplos de valores
id logradouro	4714, 433, 3135
id tp emergencia	ATENDIMENTO PRÉ-HOSPITALAR, ACIDENTE DE TRÂNSITO
Estacao do ano	Primavera, Verão, Outono, Inverno
Horario da ocorrencia	0,1,2,3,4,5...23
Dia do mês	1,2,3,4...31
Bairro	CENTRO, ESTREITO, 32

Tabela 8: Dados das ocorrências utilizados e exemplos de valores.

3.3. Modelagem e Avaliação: Associação Apriori

A regra de Associação Apriori tem o objetivo de gerar regras de forma a identificar padrões entre um dado de conjunto de itens, os antecedentes e o consequente. Foram utilizadas todas as ocorrências de Florianópolis para o teste disponibilizado e como parâmetros de entrada utilizou-se o suporte mínimo de 0.05 (cinco porcento), confiança de 30% (trinta porcento), número de regras máximo de 1000 regras e selecionou-se como classe o atributo ”id tp emergencia”. O modelo gerou 407 regras distintas e, como nem todas as regras geradas são relevantes, selecionou-se as regras mais interessantes levando em consideração o seu suporte e confiança. Um detalhe adicional refere-se ao lift, no qual não é gerado quando se utiliza um atributo de classe como parâmetro para o algoritmo. As seguinte regras foram selecionadas:

- Regra 43. umidade relativa media=Normal (Entre 75% e 90%) suporte:61,7% ==¿ emergencia=”Atendimento pré-hospitalar”confiança:(39.5%). Essa regra possui a maior confiança em relação ao suporte das regras geradas. Indica que a maior parte dos acidentes pré-hospitalar acontecem em dias em que a umidade relativa media esta entre 75 e 90%.
- Regra 70. temperatura maxima=Normal (Entre 18ºC e 30ºC) umidade relativa media=Normal (Entre 75% e 90%) suporte: 47,8% ==¿ emergencia=”Atendimento pré-hospitalar”confiança:39,4%. A regra 43 engloba as transações desta mesma regra, mas esta traz a temperatura máxima.
- Regra 18. temperatura maxima=Normal (Entre 18ºC e 30ºC) temperatura minima=Normal (Entre 15ºC e 25ºC) estacaoDoAno=Verao suporte:14,4% ==¿ emergencia=”Atendimento pré-hospitalar”confiança:39,4%. Esta é a regra com maior suporte e com a informação de estação do ano. Ela é um indicativo de que a estação do ano não tem grande impacto sobre as ocorrências.
- Regra 9. diaDaSemana=Domingo suporte:13.2% ==¿ emergencia=”Atendimento pré-hospitalar”conf:40.2%. É a primeira regra gerada em relação ao dia da semana. Destaca que a maior parte das ocorrências desse tipo acontecem no domingo, mas a quantidade não é tão maior assim em relação aos demais dias. Comparando-a com as regras geradas nos outros dias da semana, o suporte das outras regras fica entre 11% e 14%, tendo uma confiança que varia entre 37% até 39%.

- Regra 374. precipitacao=Com chuva temperatura minima=Normal (Entre 15ºC e 25ºC) suporte: 38,2% ==¿ emergencia=”Acidente de trânsito”conf:23.19%. Essa é a primeira regra onde a presença de chuva aparece como antecedente, mas o interessante em relação as regras geradas com presença de chuva antecedente possuem a emergência ”Acidente de Trânsito” como consequente.

3.4. Modelagem e Avaliação: K-means

O K-means possuí como objetivo gerar *clusters* de forma a alocar todas as transações(ocorrências) em somente um dos padrões gerados. Foram realizados diversos testes através da ferramenta Weka com este algoritmo.

O algoritmo recebeu como parâmetro 11 *clusters* para todas as ocorrências. O motivo de utilizar essa quantidade de grupos é dada pelo fato de possuírem 11 tipos de emergência diferentes e, também, pelo fato de que 10 bairros somados possuírem mais do que 50% de todas as ocorrências.

Característica	Valor	Grupos com o mesmo valor
id bairro	Centro	1,2,3,5,6,8,9,10
id logradouro	3281(FLORIANOPOLIS-CENTRO)	Nenhum
id tp emergencia	5(Atendimento pré-hospitalar)	1,2,4,5,6,8
Precipitacao	Sem chuva	Todos
Temperatura máxima	Alta	Nenhum
Temperatura mínima	Normal	1,2,3,4,6,7,8,9,10
Insolacao	Alta	1,4,5,6,8,10
Temperatura compensada media	Alta	1,4,6
Umidade relativa média	Normal	1,2,3,4,8,9
Velocidade vento média	Baixa	1,3,4,5,6,7,8,9,10
Numero do dia	23	9
Dia da semana	Friday(Sexta-feira)	5,6
Hora	15	8,9
Estacao do ano	Verao	3,6
mes	2(Fevereiro)	Nenhum

tabela 9: Comparação das características do grupo 0 aos demais.

A maior parte dos padrões gerados possuem a característica ”Atendimento pré-hospitalar”(valor 5) em seu ”id tp emergencia”. Ainda assim, em outross grupos a presença do tipo emergência ”Acidente de trânsito”(valor 8) e apenas um grupo com o tipo de emergência ”Auxílios e apoios”(valor 2). A quantidade de transações atrelada para os dois principais grupos são respectivamente, 15.7% e 19.1% de todas as transações(ocorrências). A tabela 9 mostra as características identificadas no grupo 0 e em quais *clusters* a mesma característica foi identificada, enquanto a tabela 10 mostra as respectivas informações para o grupo 1.

Característica	Valor	Grupos com o mesmo valor
id bairro	Centro	0,2,3,5,6,8,9,10
id logradouro	6631(PAULO FONTES)	3
id tp emergencia	5(Atendimento pré-hospitalar)	0,2,4,5,6,8
Precipitacao	Sem chuva	Todos
Temperatura máxima	Normal	2,3,4,5,6,7,8,9,10
Temperatura mínima	Normal	0,2,3,4,6,7,8,9,10
Insolacao	Alta	0,4,5,6,8,10
Temperatura compensada media	Normal	0,4,6
Umidade relativa média	Normal	0,2,3,4,8,9
Velocidade vento média	Baixa	0,3,4,5,6,7,8,9,10
Numero do dia	12	Nenhum
Dia da semana	Saturday(Sábado)	0,10
Hora	18	3
Estacao do ano	Outono	5
mes	4(Maio)	Nenhum

tabela 10: Comparação das características do grupo 1 aos demais.

O resultado obtido destaca as características mais comuns da cidade. O K-means utiliza a distância em relação ao centroide como critério para agrupar os atributos e, como é possível observar nos padrões gerados, as características mais comuns identificadas são referentes às características comuns dos dias na cidade de Florianópolis. Ou seja, o uso dos dados climáticos como característica adicional das ocorrências só destacou as características climáticas mais comuns entre todos os dias da cidade, ao invés de destacar características incomuns que potencializassem as ocorrências, indicando assim, que essas características climáticas diárias não possuem grande influência em relação às ocorrências. É importante lembrar que essas características climáticas referem-se ao dia da ocorrência e não ao momento (hora) em que ela foi atendida. É possível que as informações climáticas ainda sejam fatores importantes para a ocorrência, mas essa informação só poderá ser confirmada realizando outras análises ou talvez utilizando as características da condição climática do instante em que a ocorrência. Os atributos que são vinculados a época da ocorrência não trazem muitas informações também, com exceção da hora. Os padrões gerados possuem horários que variam entre as 09 horas e as 18 horas, destacando o período em que a maioria das ocorrências são atendidas que, ademais, é a faixa de horário onde a maioria das pessoas está fora de casa executando as atividades do dia-a-dia.

3.5. Modelagem e Avaliação: PaNDa+

O PaNDa+ possui o objetivo gerar padrões de forma a alocar as transações mais relevantes considerando uma margem de erro.

Foram realizados diversos testes com o algoritmo. Os testes variaram em relação à margem de erro configurada como parâmetro e os dados de entrada, os quais variaram entre todas as ocorrências e ocorrências relativas a tipos específicos de emergências. Quanto maior a margem de erro configurada, mais atributos eram encontrados nos padrões considerados relevantes, enquanto quando a margem de erro era configurada como ”0”, os padrões possuíam poucas transações vinculadas a eles, mas com atributos sempre presentes nas transações vinculadas. Em outros testes, foram utilizados os dados climáticos no PaNDa+, mas estes apontaram as mesmas características comuns agrupadas pelos *clusters* do K-means. Por este motivo, removeu-se a maioria os dados climáticos para o testes disponibilizado utilizando o PaNDa+, deixando apenas a presença da chuva, e foram adicionado mais informações em relação as datas das ocorrências. Por não possuir métrica para definir a margem de erro para os dados, utilizou-se para disponibilização deste trabalho apenas os testes que usaram as configurações padrão do algoritmo. A configuração padrão do algoritmo utiliza uma margem de erro dinâmica baseada na função de custo, que é calculada a cada iteração do algoritmo.

Os resultados do PaNDa+ não foram melhores do que o do K-means. Pelas características que foram utilizadas para as ocorrências, esperava-se encontrar padrões sólidos que indicassem momentos onde as ocorrências acontecessem, como horários do dia ou meses do ano. Foram selecionados 3 modelos e os parâmetros e os dados de entrada, para cada modelo, são os seguintes:

3.5.1. Modelo 1 : Todas as ocorrências

O primeiro modelo recebeu como parâmetro o padrão do algoritmo e os dados de entrada desse modelo são todas as ocorrências atendidas. O modelo possuí o objetivo de identificar as características mais comuns das ocorrências com a sua respectiva proporção em relação a todas as ocorrências. O algoritmo também aloca a mesma transação em agrupamentos diferentes. Dessa forma, a soma das transações alocadas em cada grupo pode ultrapassar 100%, já que a mesma transação pode aparecer em um ou mais grupos gerados. Essa característica é uma das mais importantes ao comparar o Panda+ com o K-means, no qual obriga que cada transação esteja em apenas um dos grupos gerados.

- O padrão "1" destacou que 70.28% das transações podem ser vinculados às características "Temporada:Comum", "TipoDoDIa:DiaUtil"e "Precipitacao: Sem chuva". Que é uma informação muito similar ao que foi encontrado em relação ao K-means, o fato de que as ocorrências acontecem independente das características selecionadas, e sim, pelo fato de que a maioria dos dias em que as ocorrências foram atendidas são dias comuns.
- O padrão "2" destacou que 15.61% das ocorrências possuem a característica de ser um atendimentos pré-hospitalar e possuir um dia com chuva.
- O padrão "3" destacou que 19.66% das ocorrências possuem a característica de estarem na "Estacao:Verao", que é praticamente o mesmo período da "Temporada:AltaTemporada"de Florianópolis, na qual o verão faz referência ao meses de "Dezembro, Janeiro e Fevereiro"enquanto a "AltaTemporada"só abrange os meses de "Janeiro"e "Fevereiro". A informação do "TipoDoDia:DiaUtil"só indica que o final de semana não é um fator que aumenta o número de ocorrências.

3.5.2. Modelo 2 : Ocorrências do tipo "Atendimento pré-hospitalar"

O segundo modelo recebeu como parâmetro o padrão do algoritmo e os dados de entrada desse modelo são todas as ocorrências do tipo "Atendimento pré-hospitalar". O modelo possuí o objetivo de identificar as características mais comuns em relação ao tipo de ocorrência selecionada, indicando a sua respectiva proporção em relação a todas as ocorrências.

- O padrão "1" destacou que 46.61% das transações podem ser vinculados às características "ATENDIMENTO PRÉ-HOSPITALAR"e "Precipitacao: Com chuva". Como todos os dados usados para esse teste são do tipo "Atendimento pré-hospitalar", não é uma característica relevante para esse agrupamento. Por outro lado, a informação "Precipitacao: Com chuva"reforça a descoberta feita no teste com todas as ocorrências, indicando que a chuva realmente aumenta o número de atendimentos pré-hospitalares.
- O padrão "2" destacou que 8.87% das ocorrências possuem a característica de ser "TipoDoDia:FinalDeSemana"e "Estacao:Verao", permitindo a interpretação de que durante o verão as ocorrências de atendimento pré-hospitalar acontecem mais durante os finais de semana comparado-se aos dias úteis. Fato que também é reforçado pelo padrão "3", dessa vez para o inverno.

3.5.3. Modelo 3 : Ocorrências do tipo ”Acidente de trânsito”

O terceiro modelo recebeu como parâmetro o padrão do algoritmo e os dados de entrada desse modelo são todas as ocorrências do tipo ”Acidente de trânsito”.O modelo possuí o objetivo de identificar as características mais comuns em relação ao tipo de ocorrência selecionada, indicando a sua respectiva proporção em relação a todas as ocorrências.

- O padrão ”1” destacou que 25.41% das transações podem ser vinculados às características ”ACIDENTE DE TRÂNSITO” e ”Precipitacao: Com chuva”. Como todos os dados usados para esse teste são do tipo ”ACIDENTE DE TRÂNSITO”, não é uma característica relevante para esse agrupamento. Contudo, a informação ”Precipitacao: Com chuva”, indica que essa ocorrência é potencializada pela presença de chuva.

4. Conclusão

Este artigo teve como objetivo realizar a coleta de dados e estudar as ocorrências atendidas pelo corpo de bombeiros através do processo de mineração de dados para a descoberta de padrões. A partir de dados como: registros dos atendimentos, características climáticas das datas das ocorrências, informações referente a estação do ano e horário do dia; foi feito o entendimento dos dados, levantando informações relevantes que pudessem auxiliar no processo de *data mining*, assim como informações importantes sobre os lugares onde as ocorrências mais ocorrem, como a identificação dos principais bairros e logradouros. Em seguida foi realizada a preparação dos dados para que pudessem ser utilizados pelos algoritmos K-means, Associação Apriori e PaNDa+.

O projeto enfrentou muitas dificuldades em etapas diferentes. O primeiro desafio foi em relação aos próprios registros dos bairros e logradouros. Muitos estavam duplicados, com erros de digitação e até inexistentes. Visto que muitos dos logradouros fazem referência a áreas abrangentes como bairros inteiros e praças, para não desperdiçar as informações das ocorrências atendidas nesses locais, dedicou-se muito tempo para ajustar esses dados, principalmente em relação aos bairros, onde foram desenhados polígonos para os que não foram encontrados por api. Depois de finalizar a aplicação e notar a incoerência de muitos dados, desejou-se verificar com os próprios fornecedores informações em relação ao cadastro de ocorrências. Observou-se que os dados fornecidos são generalizados. Por exemplo, ”Salvamento e Resgates”na base oficial é detalhada como ”Afogamento”, ”Alagamento”, ”Colisão/Choque”, entre outros.

O principal objetivo de verificar a origem dos dados foi obter a geolocalização das ocorrências atendidas fornecidas em um segundo momento através de 14 planilhas diferentes as quais tiveram de ser unidas. Cada planilha fazia referência a um batalhão da cidade de Florianópolis. Esse exercício teve que ser feito com cuidado pois cada planilha tinha um conjunto diferente de informações. A dificuldade seguinte foi em relação a modelagem das técnicas utilizadas para a mineração de dados. Foram utilizadas diversas técnicas diferentes de classificação, *clustering*, entre outras para encontrar padrões dentro dos dados disponibilizados. A grande maioria dos resultados obtidos foram infrutíferos, provavelmente pela granularidade muito generalizada. Portanto as informações mais relevantes extraídas durante o projeto foram durante na etapa de exploração de dados, etapa revisitada muitas vezes. Em conclusão, as principais descobertas do trabalho são as seguintes:

- O bairro ”Centro” é a região que mais atende chamados de emergência durante o ano todo.
- O bairro ”Canasvieiras” é o único bairro com aumento significativo de ocorrências durante o Verão. black
- Os dez bairros com mais ocorrências sâo: Centro, Lagoa da Conceição, Estreito, Capoeiras, Trindade, Barra da Lagoa, Canasvieiras, Agronômica, Coqueiros e Rio Vermelho;
- A rua ”Paulo Fontes”possui o maior número de ocorrências do tipo ”Acidente de trânsito”.
- O logradouro ”Renato Ântônio de Souza”possui o maior número de ocorrências do tipo ”Auxílios e Apoios”; black
- Os logradouros com maior número de ocorrências são: Renato Antônio de Souza, Paulo Fontes, Florianópolis-Centro, Jornalista Rubens de Arruda Ramos, Lauro Linhares, Santos Saraiva, SC-405, Governador GUstavo Richard, BR-282, Ivo Silveira, Mauro Ramos, João GUalberto Soares, Canasvieiras, Barra da Lagoa, Baldicero Filomeno.
- A maior parte dos atendimentos são do tipo ”Atendimento pré-hospitalar” e ”Acidente de trânsito”. A corporação deve estar preparada para atender uma demanda maior de ocorrências desse tipo do que das demais. black
- A utilização dos dados climáticos do dia para atendimentos das emergências do Corpo de Bombeiros de Santa Catarina não é recomendada para trabalhos posteriores.
- A maior parte dos acidentes de trânsito ocorrem as 10 e 22 horas, sendo mais comuns entre as 17 e 18 horas.

Referências

Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules. In *Proc. of 20th Intl. Conf. on VLDB*, pages 487–499.

Charu C., A. and Chandan K., R. (2013). *Data Clustering: Algorithms and Applications*, volume 652. Chapman e Hall/CRC.

Claudio, L., Salvatore, O., and Raffaele, P. (2013). A unifying framework for greedy mining approximate top-k binary patterns and their evaluation. 14:27–34.

Junjie, W. (2012). *Advances in K-means Clustering: A Data Mining Thinking*, volume 194. Springer Publishing Company.

Matt, J. W., A, C. M., H, F. P., A, H. Z., J, B. T., J, W. G., and S., B. D. M. J. (2015). Climate-induced variations in global wildfire danger from 1979 to 2013. 6(7537):387–404.

OpenStreetMap (2019). Openstreetmap. `https://www.openstreetmap.org`. Accessed: 2019-05-11.

Pang-ning, T., Michael, S., and Vipin, K. (2009). *Introdução ao Data Mining. Rio*, volume 900. Ciência Moderna.

APÊNDICE B – Códigos da ferramenta de visualização

Bombeirosmap.py:

```
# coding: utf-8

from flask import Flask, request, url_for, render_template
from flask_googlemaps import GoogleMaps
from flask_googlemaps import Map
from db import ocorrencias, bairro_logradouro, logradouros,
    bairros, ocorrencias, tp_emergencia
import cadastros
import mapUtil
import poligonos
import estatisticas
import json
from collections import OrderedDict

app = Flask("wtf")
GoogleMaps(app)

#Utilizado para cadastrar ocorrencias
@app.route("/ocorrencias/cadastrao", methods=["GET", "POST"])
def cadastrao():
        cadastros.cadastrar_bairro()
        #cadastros.visualizar_bairros()
        cadastros.cadastrar_logradouros()
        #cadastros.visualizar_logradouros()
        cadastros.cadastrar_relacionamento()
        #cadastros.visualizar_relacionamento()
        cadastros.cadastrar_tp_emergencia()
        #cadastros.visualizar_tp_emergencia()
        #cadastros.cadastrar_ocorrencia()
        cadastros.cadastrar_ocorrencias()
        #cadastros.visualizar_ocorrencia()
        cadastros.cadastrar_ocorrenciasComGeolocalizacao()
```

```
#Tela inicial
@app.route("/ocorrencias")
def index():
        bairros = mapUtil.geraListaNomes(list(cadastros.
            get_todos_bairros()),'nome_bairro')

        resumoOcorrencias = {}
        tpEmergencias = ['INCENDIO','AUXILIOS / APOIOS', '
            PRODUTOS_PERIGOSOS', 'SALVAMENTO / BUSCA / RESGATE',
            'ATENDIMENTO_PRE-HOSPITALAR', 'DIVERSOS', 'ACIDENTE
            DE_TRANSITO', 'ACOES_PREVENTIVAS', 'AVERIGUACAO /
            CORTE_DE_ARVORE', 'AVERIGUACAO / MANEJO_DE_INSETO']
        for x in range(0,10):
                dictOcorrencias = { tpEmergencias[x] : cadastros
                    .ocorrenciasPorAnoPorEmergencia(tpEmergencias
                    [x])}
                resumoOcorrencias.update(dictOcorrencias)

        mymap = Map(
                style= "height:550px;width:1300px;margin:0;",
                zoom = 12,
        identifier="view-side",
        lat=-27.6074359,
        lng=-48.533273,
        polygons=poligonos.getTodosPolygons()
    )
        bairros = mapUtil.geraListaNomes(list(cadastros.
            get_todos_bairros()),'nome_bairro')

        return render_template("telaPrincipal2.html", bairros=
            bairros, mymap=mymap, resumoOcorrenciasIncendios =
            resumoOcorrencias[tpEmergencias[0]],
            resumoOcorrenciasAuxilios = resumoOcorrencias[
            tpEmergencias[1]], resumoOcorrenciasPerigosos =
            resumoOcorrencias[tpEmergencias[2]],
            resumoOcorrenciasSalvamento = resumoOcorrencias[
            tpEmergencias[3]], resumoOcorrenciasHospitalar =
            resumoOcorrencias[tpEmergencias[4]],
```

```
        resumoOcorrenciasDiversos = resumoOcorrencias[
        tpEmergencias[5]], resumoOcorrenciasAcidente =
        resumoOcorrencias[tpEmergencias[6]],
        resumoOcorrenciasPreventivas = resumoOcorrencias[
        tpEmergencias[7]], resumoOcorrenciasArvore =
        resumoOcorrencias[tpEmergencias[8]],
        resumoOcorrenciasInseto = resumoOcorrencias[
        tpEmergencias[9]])

#Tela da cidade toda com filtros
@app.route("/ocorrencias/cidade")
def indexus():
        tipo_emergencia = request.args.get("emergencia")
        ano = request.args.get("ano")
        estacao = request.args.get("estacao")
        diaDaSemana = request.args.get("diaDaSemana")

        option = 0
        if(tipo_emergencia == None or tipo_emergencia == "Todos"
           ):
                option = option + 1
        if(ano == None or ano == "Todos"):
                option = option + 2
        if(estacao == None or estacao == "Todos"):
                option = option + 4
        if(diaDaSemana == None or diaDaSemana == "Todos"):
                option = option + 8

        statement = cadastros.statementBuilder(tipo_emergencia,
           ano, estacao,diaDaSemana)
        ocorrenciasPorFiltroPorBairro = cadastros.
           get_densidades_bairro(statement)

        aux = 0;
        dictDensidadePorBairro = {}
        maxOcorrencia = 0
        arrayBairros = []

        for row in ocorrenciasPorFiltroPorBairro:
```

```
            aux = aux + 1
            if(aux == 1):
                    maxOcorrencia = row["totalDeOcorrencias"
                        ]
            dictBairro = {int(row["id_original_bairro"]) :
                round(row["totalDeOcorrencias"]/maxOcorrencia
                ,2)}
            arrayBairros.append(int(row["id_original_bairro"
                ]))
            dictDensidadePorBairro.update(dictBairro)
    bairros = mapUtil.geraListaNomes(list(cadastros.
        get_todos_bairros()),'nome_bairro')

    resumoOcorrencias = {}
    tpEmergencias = ['INCENDIO','AUXILIOS / APOIOS', '
        PRODUTOS_PERIGOSOS', 'SALVAMENTO / BUSCA / RESGATE',
        'ATENDIMENTO_PRE-HOSPITALAR', 'DIVERSOS', 'ACIDENTE_
        DE_TRANSITO', 'ACOES_PREVENTIVAS', 'AVERIGUACAO / _
        CORTE_DE_ARVORE', 'AVERIGUACAO / MANEJO_DE_INSETO']
    for x in range(0,10):
            dictOcorrencias = { tpEmergencias[x] : cadastros
                .ocorrenciasPorAnoPorEmergencia(tpEmergencias
                [x])}
            resumoOcorrencias.update(dictOcorrencias)

    poligonosParaOMapa = poligonos.getPolygons(arrayBairros,
        dictDensidadePorBairro)

    mymap = Map(
            style= "height:550px;width:1300px;margin:0;",
            zoom = 12,

    identifier="view-side",
    lat=-27.6074359,
    lng=-48.533273,
    polygons=poligonosParaOMapa
)
    bairros = mapUtil.geraListaNomes(list(cadastros.
        get_todos_bairros()),'nome_bairro')
```

```
        return render_template("telaPrincipal2.html", bairros=
            bairros, mymap=mymap, resumoOcorrenciasIncendios =
            resumoOcorrencias[tpEmergencias[0]],
            resumoOcorrenciasAuxilios = resumoOcorrencias[
            tpEmergencias[1]], resumoOcorrenciasPerigosos =
            resumoOcorrencias[tpEmergencias[2]],
            resumoOcorrenciasSalvamento = resumoOcorrencias[
            tpEmergencias[3]], resumoOcorrenciasHospitalar =
            resumoOcorrencias[tpEmergencias[4]],
            resumoOcorrenciasDiversos = resumoOcorrencias[
            tpEmergencias[5]], resumoOcorrenciasAcidente =
            resumoOcorrencias[tpEmergencias[6]],
            resumoOcorrenciasPreventivas = resumoOcorrencias[
            tpEmergencias[7]], resumoOcorrenciasArvore =
            resumoOcorrencias[tpEmergencias[8]],
            resumoOcorrenciasInseto = resumoOcorrencias[
            tpEmergencias[9]])

#Tela da cidade toda com histogramas
@app.route("/ocorrencias/cidade2")
def indexu():
        tipo_emergencia = request.args.get("emergencia")
        ano = request.args.get("ano")
        estacao = request.args.get("estacao")
        diaDaSemana = request.args.get("diaDaSemana")

        option = 0
        if(tipo_emergencia == None or tipo_emergencia == "Todos"
            ):
                option = option + 1
        if(ano == None or ano == "Todos"):
                option = option + 2
        if(estacao == None or estacao == "Todos"):
                option = option + 4
        if(diaDaSemana == None or diaDaSemana == "Todos"):
                option = option + 8
```

```
statement = cadastros.statementBuilder(tipo_emergencia,
    ano, estacao,diaDaSemana)
ocorrenciasPorFiltroPorBairro = cadastros.
    get_densidades_bairro(statement)

lista_ocorrencia = cadastros.getAll(tipo_emergencia, ano
    , estacao,diaDaSemana)

dictLinksParaHistogramas = estatisticas.
    criaDicionarioDeEstatisticas(lista_ocorrencia)

aux = 0;
dictDensidadePorBairro = {}
maxOcorrencia = 0
arrayBairros = []

for row in ocorrenciasPorFiltroPorBairro:
        aux = aux + 1
        if(aux == 1):
                maxOcorrencia = row["totalDeOcorrencias"
                    ]
        dictBairro = {int(row["id_original_bairro"]) :
            round(row["totalDeOcorrencias"]/maxOcorrencia
            ,1)}
        arrayBairros.append(int(row["id_original_bairro"
            ]))
        dictDensidadePorBairro.update(dictBairro)

bairros = mapUtil.geraListaNomes(list(cadastros.
    get_todos_bairros()),'nome_bairro')

resumoOcorrencias = {}
tpEmergencias = ['INCENDIO','AUXILIOS_/_APOIOS', '
    PRODUTOS_PERIGOSOS', 'SALVAMENTO_/_BUSCA_/_RESGATE',
    'ATENDIMENTO_PRE-HOSPITALAR', 'DIVERSOS', 'ACIDENTE_
    DE_TRANSITO', 'ACOES_PREVENTIVAS', 'AVERIGUACAO_/_
    CORTE_DE_ARVORE', 'AVERIGUACAO_/_MANEJO_DE_INSETO']
for x in range(0,10):
```

```
                dictOcorrencias = { tpEmergencias[x] : cadastros
                    .ocorrenciasPorAnoPorEmergencia(tpEmergencias
                    [x])}
                resumoOcorrencias.update(dictOcorrencias)

        poligonosParaOMapa = poligonos.getPolygons(arrayBairros,
            dictDensidadePorBairro)

        mymap = Map(
                style= "height:550px;width:1300px;margin:0;",
                zoom = 12,

        identifier="view-side",
        lat=-27.6074359,
        lng=-48.533273,
        polygons=poligonosParaOMapa
)
        bairros = mapUtil.geraListaNomes(list(cadastros.
            get_todos_bairros()),'nome_bairro')

        return render_template("telaPrincipal.html", bairros=
            bairros, mymap=mymap, resumoOcorrenciasIncendios =
            resumoOcorrencias[tpEmergencias[0]],
            resumoOcorrenciasAuxilios = resumoOcorrencias[
            tpEmergencias[1]], resumoOcorrenciasPerigosos =
            resumoOcorrencias[tpEmergencias[2]],
            resumoOcorrenciasSalvamento = resumoOcorrencias[
            tpEmergencias[3]], resumoOcorrenciasHospitalar =
            resumoOcorrencias[tpEmergencias[4]],
            resumoOcorrenciasDiversos = resumoOcorrencias[
            tpEmergencias[5]], resumoOcorrenciasAcidente =
            resumoOcorrencias[tpEmergencias[6]],
            resumoOcorrenciasPreventivas = resumoOcorrencias[
            tpEmergencias[7]], resumoOcorrenciasArvore =
            resumoOcorrencias[tpEmergencias[8]],
            resumoOcorrenciasInseto = resumoOcorrencias[
            tpEmergencias[9]], dictLinksParaHistogramas=
            dictLinksParaHistogramas)
```

```
@app.route("/ocorrencias/bairros/<bairro_escolhido>", methods=["
    GET", "POST"])
def ocorrenciasBairros(bairro_escolhido):

        tipo_emergencia = request.args.get("emergencia")
        ano = request.args.get("ano")
        estacao = request.args.get("estacao")
        diaDaSemana = request.args.get("diaDaSemana")

        option = 0
        if(tipo_emergencia == None or tipo_emergencia == "Todos"
            ):
                option = option + 1
        if(ano == None or ano == "Todos"):
                option = option + 2
        if(estacao == None or estacao == "Todos"):
                option = option + 4
        if(diaDaSemana == None or diaDaSemana == "Todos"):
                option = option + 8

        bairro_escolhido = bairro_escolhido.upper()

        #Pega a lista de relacoes bairro_logradouro a partir de
            um bairro escolhido
        lista_relacao_bairro_logradouro_do_bairro_escolhido =
            cadastros.get_relacao_bairro(bairro_escolhido)
        #Pega a lista de logradouros daquele bairro
        lista_nome_logradouros_do_bairro = mapUtil.
            geraListaDeLogradouros(
            lista_relacao_bairro_logradouro_do_bairro_escolhido)

        arrayDeIdDeLogradouros = []
        for x in range(len(
            lista_relacao_bairro_logradouro_do_bairro_escolhido))
            :
                arrayDeIdDeLogradouros.append(
                    lista_relacao_bairro_logradouro_do_bairro_escolhido
                    [x]['id'])
```

```
resumoOcorrencias = {}
tpEmergencias = ['INCENDIO','AUXILIOS / APOIOS', '
    PRODUTOS_PERIGOSOS', 'SALVAMENTO / BUSCA / RESGATE',
    'ATENDIMENTO_PRE-HOSPITALAR', 'DIVERSOS', 'ACIDENTE_
    DE_TRANSITO', 'ACOES_PREVENTIVAS', 'AVERIGUACAO / _
    CORTE_DE_ARVORE', 'AVERIGUACAO / MANEJO_DE_INSETO']
for x in range(0,10):
        dictOcorrencias = { tpEmergencias[x] : cadastros
            .ocorrenciasPorAnoPorEmergenciaPorLogradouros
            (tpEmergencias[x], arrayDeIdDeLogradouros)}
        resumoOcorrencias.update(dictOcorrencias)

#Pega ocorrencias que aconteceram naquele bairro,
    utilizando a lista de relacao bairro_logradouro com o
     bairro escolhido
#lista_ocorrencia = cadastros.get_ocorrencias(
    lista_relacao_bairro_logradouro_do_bairro_escolhido)

#lista_ocorrencia = cadastros.get_ocorrenciasComFiltro(
    lista_relacao_bairro_logradouro_do_bairro_escolhido,
    tipo_emergencia, ano, estacao, diaDaSemana, option)
ocorrencias_do_bairro = cadastros.getTodasOcorrencias(
    lista_relacao_bairro_logradouro_do_bairro_escolhido,
    tipo_emergencia, ano, estacao, diaDaSemana)

counter = 0
lista_ocorrencia = {}
for row in ocorrencias_do_bairro:
        novoDict = { counter : {"id_tp_emergencia":row["
            id_tp_emergencia"],"id_bairro_logradouro":row
            ["id_bairro_logradouro"],"ano":row["ano"],"
            hora_da_ocorrencia":row["hora_da_ocorrencia"
            ],"mes":row["mes"],"dia_da_semana":row["
            dia_da_semana"],"dia_do_mes":row["dia_do_mes"
            ],"estacao_do_ano":row["estacao_do_ano"],"
            precipitacao":row["precipitacao"],"
            temperatura_minima":row["temperatura_minima"
            ],"temperatura_maxima":row["
            temperatura_maxima"],"evaporacao_piche":row["
```

```
                evaporacao_piche"],"velocidade_vento_media":
                row["velocidade_vento_media"],"
                umidade_relativa_media":row["
                umidade_relativa_media"],"insolacao":row["
                insolacao"]}}
            counter = counter + 1
            lista_ocorrencia.update(novoDict)

    dictLinksParaHistogramas = estatisticas.
        criaDicionarioDeEstatisticas(lista_ocorrencia)

    #Pega o bairro escolhido, sera usado como centro do mapa
         na hora de carregar o mapa
    bairro = cadastros.get_bairro(bairro_escolhido)

    #Carrega os marcadores no mapa a partir da relacao
        bairro_logradouro, visto que essa sera a
        geolocalizacao da ocorrencia
    markers_on_map = mapUtil.criar_marcadores(
        lista_relacao_bairro_logradouro_do_bairro_escolhido,
        lista_ocorrencia)
    #ocorrencia = ocorrencias.find_one(id=ocorrencia_id)
    mymap = Map(
            style= "height:550px;width:1000px;margin:0;",
            zoom = 10,
    identifier="view-side",
    lat=bairro['latitude'],
    lng=bairro['longitude'],
    polygons= [poligonos.poligonoBairros(int(bairro["
        id_original_bairro"]),0)],
    markers = markers_on_map
)
    #O uso das {{}} no template.html identifica os campos
        que serao carregados atrav s do render_template. No
        caso, serao as colunas adjacentes do banco.
    return render_template("ocorrencias_por_bairro.html",
        bairro=bairro_escolhido, nome_logradouro_do_bairro =
        lista_nome_logradouros_do_bairro, mymap=mymap,
        resumoOcorrenciasIncendios = resumoOcorrencias[
```

```
        tpEmergencias[0]], resumoOcorrenciasAuxilios =
        resumoOcorrencias[tpEmergencias[1]],
        resumoOcorrenciasPerigosos = resumoOcorrencias[
        tpEmergencias[2]], resumoOcorrenciasSalvamento =
        resumoOcorrencias[tpEmergencias[3]],
        resumoOcorrenciasHospitalar = resumoOcorrencias[
        tpEmergencias[4]], resumoOcorrenciasDiversos =
        resumoOcorrencias[tpEmergencias[5]],
        resumoOcorrenciasAcidente = resumoOcorrencias[
        tpEmergencias[6]], resumoOcorrenciasPreventivas =
        resumoOcorrencias[tpEmergencias[7]],
        resumoOcorrenciasArvore = resumoOcorrencias[
        tpEmergencias[8]], resumoOcorrenciasInseto =
        resumoOcorrencias[tpEmergencias[9]],
        dictLinksParaHistogramas=dictLinksParaHistogramas)

@app.route("/ocorrencias/logradouros/<logradouro_escolhido>")
def ocorrenciasLogradouros(logradouro_escolhido):

        tipo_emergencia = request.args.get("emergencia")
        ano = request.args.get("ano")
        estacao = request.args.get("estacao")
        diaDaSemana = request.args.get("diaDaSemana")

        option = 0
        if(tipo_emergencia == None or tipo_emergencia == "Todos"
           ):
                option = option + 1
        if(ano == None or ano == "Todos"):
                option = option + 2
        if(estacao == None or estacao == "Todos"):
                option = option + 4
        if(diaDaSemana == None or diaDaSemana == "Todos"):
                option = option + 8

        #Pega a lista de relacoes bairro_logradouro a partir de
           um logradouro escolhido
        relacao_bairro_logradouro_do_logradouro_escolhido =
           cadastros.get_relacao_logradouro(logradouro_escolhido
```

```
    )

id_do_bairro =
    relacao_bairro_logradouro_do_logradouro_escolhido[0][
    "id_bairro"]


bairro = cadastros.get_bairro_id(id_do_bairro)

#Pega o logradouro escolhido, usado para colocar os
    marcadores
logradouro_selecionado = cadastros.get_logradouro(
    logradouro_escolhido)

#Pega ocorrencias que aconteceram naquele bairro,
    utilizando a lista de relacao bairro_logradouro com o
     bairro escolhido
#lista_ocorrencia = cadastros.get_ocorrencias(
    relacao_bairro_logradouro_do_logradouro_escolhido)
lista_ocorrencia = cadastros.get_ocorrenciasComFiltro(
    relacao_bairro_logradouro_do_logradouro_escolhido,
    tipo_emergencia, ano, estacao, diaDaSemana, option)
arrayDeIdDeLogradouros = [
    relacao_bairro_logradouro_do_logradouro_escolhido[0][
    "id"]]
resumoOcorrencias = {}

tpEmergencias = ['INCENDIO','AUXILIOS / APOIOS', '
    PRODUTOS_PERIGOSOS', 'SALVAMENTO / BUSCA / RESGATE',
     'ATENDIMENTO_PRE-HOSPITALAR', 'DIVERSOS', 'ACIDENTE_
    DE_TRANSITO', 'ACOES_PREVENTIVAS', 'AVERIGUACAO / _
    CORTE_DE_ARVORE', 'AVERIGUACAO / MANEJO_DE_INSETO']
for x in range(0,10):
        dictOcorrencias = { tpEmergencias[x] : cadastros
            .ocorrenciasPorAnoPorEmergenciaPorLogradouros
            (tpEmergencias[x], arrayDeIdDeLogradouros)}
        resumoOcorrencias.update(dictOcorrencias)

resumoOcorrenciasMes = {}
```

```
    for x in range(0,10):
            dictOcorrenciasMes = { tpEmergencias[x] :
               cadastros.
               ocorrenciasPorMesPorEmergenciaPorLogradouro(
               tpEmergencias[x], arrayDeIdDeLogradouros)}
            resumoOcorrenciasMes.update(dictOcorrenciasMes)

    dictLinksParaHistogramas = estatisticas.
       criaDicionarioDeEstatisticas(lista_ocorrencia)

    #Pega o bairro escolhido, sera usado como centro do mapa
        na hora de carregar o mapa
    bairro = cadastros.get_bairro_id(
       relacao_bairro_logradouro_do_logradouro_escolhido[0][
       'id_bairro'])

    #Carrega os marcadores no mapa a partir da relacao
       bairro_logradouro, visto que essa sera a
       geolocalizacao da ocorrencia
    markers_on_map = mapUtil.criar_marcadores(
       relacao_bairro_logradouro_do_logradouro_escolhido,
       lista_ocorrencia)

    mymap = Map(
            style= "height:550px;width:1000px;margin:0;",
            zoom = 13,
    identifier="view-side",
    lat=bairro['latitude'],
    lng=bairro['longitude'],
    polygons= [poligonos.poligonoBairros(int(bairro["
       id_original_bairro"]),0)],
    markers = markers_on_map
)

    #O uso das {{}} no template.html identifica os campos
       que serao carregados atrav s do render_template. No
       caso, serao as colunas adjacentes do banco.
    return render_template("ocorrencias_por_logradouro.html"
       , nome_logradouro = logradouro_selecionado['
```

```
            nome_logradouro'], lista_ocorrencia =
            lista_ocorrencia, mymap=mymap,
            resumoOcorrenciasIncendios = resumoOcorrencias[
            tpEmergencias[0]], resumoOcorrenciasAuxilios =
            resumoOcorrencias[tpEmergencias[1]],
            resumoOcorrenciasPerigosos = resumoOcorrencias[
            tpEmergencias[2]], resumoOcorrenciasSalvamento =
            resumoOcorrencias[tpEmergencias[3]],
            resumoOcorrenciasHospitalar = resumoOcorrencias[
            tpEmergencias[4]], resumoOcorrenciasDiversos =
            resumoOcorrencias[tpEmergencias[5]],
            resumoOcorrenciasAcidente = resumoOcorrencias[
            tpEmergencias[6]], resumoOcorrenciasPreventivas =
            resumoOcorrencias[tpEmergencias[7]],
            resumoOcorrenciasArvore = resumoOcorrencias[
            tpEmergencias[8]], resumoOcorrenciasInseto =
            resumoOcorrencias[tpEmergencias[9]],
            resumoOcorrenciasMesIncendios = resumoOcorrenciasMes[
            tpEmergencias[0]], resumoOcorrenciasMesAuxilios =
            resumoOcorrenciasMes[tpEmergencias[1]],
            resumoOcorrenciasMesPerigosos = resumoOcorrenciasMes[
            tpEmergencias[2]], resumoOcorrenciasMesSalvamentos =
            resumoOcorrenciasMes[tpEmergencias[3]],
            resumoOcorrenciasMesHospitalar = resumoOcorrenciasMes
            [tpEmergencias[4]], resumoOcorrenciasMesDiversos =
            resumoOcorrenciasMes[tpEmergencias[5]],
            resumoOcorrenciasMesAcidentes = resumoOcorrenciasMes[
            tpEmergencias[6]], resumoOcorrenciasMesPreventivas =
            resumoOcorrenciasMes[tpEmergencias[7]],
            resumoOcorrenciasMesArvore = resumoOcorrenciasMes[
            tpEmergencias[8]], resumoOcorrenciasMesInseto =
            resumoOcorrenciasMes[tpEmergencias[9]],
            dictLinksParaHistogramas=dictLinksParaHistogramas)

@app.route("/ocorrencias/logradouros/Geo/<logradouro_escolhido>"
    )
def ocorrenciasLogradourosGeo(logradouro_escolhido):

        tipo_emergencia = request.args.get("emergencia")
```

```
ano = request.args.get("ano")
estacao = request.args.get("estacao")
diaDaSemana = request.args.get("diaDaSemana")

option = 0

if(tipo_emergencia == None or tipo_emergencia == "Todos"
    ):
        option = option + 1
if(ano == None or ano == "Todos"):
        option = option + 2
if(estacao == None or estacao == "Todos"):
        option = option + 4
if(diaDaSemana == None or diaDaSemana == "Todos"):
        option = option + 8

#Pega a lista de relacoes bairro_logradouro a partir de
    um logradouro escolhido
relacao_bairro_logradouro_do_logradouro_escolhido =
    cadastros.get_relacao_logradouro(logradouro_escolhido
    )

id_do_bairro =
    relacao_bairro_logradouro_do_logradouro_escolhido[0][
    "id_bairro"]


bairro = cadastros.get_bairro_id(id_do_bairro)

#Pega o logradouro escolhido, usado para colocar os
    marcadores
logradouro_selecionado = cadastros.get_logradouro(
    logradouro_escolhido)

#Pega ocorrencias que aconteceram naquele bairro,
    utilizando a lista de relacao bairro_logradouro com o
     bairro escolhido
ocorrencias_do_logradouro= cadastros.
    getTodasOcorrenciasGeo(
```

```
    relacao_bairro_logradouro_do_logradouro_escolhido,
    tipo_emergencia, ano, estacao, diaDaSemana)
#lista_ocorrencia = cadastros.get_ocorrenciasGeo(
    relacao_bairro_logradouro_do_logradouro_escolhido)
counter = 0
lista_ocorrencia = {}
for row in ocorrencias_do_logradouro:
        novoDict = { counter : {"tp_emergencia":row["
            tp_emergencia"],"temperatura_comp_media":row[
            "temperatura_comp_media"],"dia":row["dia"],"
            id_tp_emergencia":row["id_tp_emergencia"],"
            lng":row["lng"],"lat":row["lat"],"
            id_bairro_logradouro":row["
            id_bairro_logradouro"],"ano":row["ano"],"
            hora_da_ocorrencia":row["hora_da_ocorrencia"
            ],"mes":row["mes"],"dia_da_semana":row["
            dia_da_semana"],"dia_do_mes":row["diaDoMes"],
            "estacao_do_ano":row["estacao_do_ano"],"
            precipitacao":row["precipitacao"],"
            temperatura_minima":row["temperatura_minima"
            ],"temperatura_maxima":row["
            temperatura_maxima"],"evaporacao_piche":row["
            evaporacao_piche"],"velocidade_vento_media":
            row["velocidade_vento_media"],"
            umidade_relativa_media":row["
            umidade_relativa_media"],"insolacao":row["
            insolacao"]}}
        counter = counter + 1
        lista_ocorrencia.update(novoDict)

arrayDeIdDeLogradouros = [
    relacao_bairro_logradouro_do_logradouro_escolhido[0][
    "id"]]
resumoOcorrencias = {}

tpEmergencias = ['INCENDIO', 'AUXILIOS / APOIOS', '
    PRODUTOS_PERIGOSOS', 'SALVAMENTO / BUSCA / RESGATE',
    'ATENDIMENTO_PRE-HOSPITALAR', 'DIVERSOS', 'ACIDENTE_
    DE_TRANSITO', 'ACOES_PREVENTIVAS', 'AVERIGUACAO / '
```

```
    CORTE_DE_ARVORE', 'AVERIGUACAO_/_MANEJO_DE_INSETO']
for x in range(0,10):

        dictOcorrencias = { tpEmergencias[x] : cadastros
            .ocorrenciasPorAnoPorEmergenciaPorLogradouros
            (tpEmergencias[x], arrayDeIdDeLogradouros)}
        resumoOcorrencias.update(dictOcorrencias)

resumoOcorrenciasMes = {}
for x in range(0,10):
        dictOcorrenciasMes = { tpEmergencias[x] :
            cadastros.
            ocorrenciasPorMesPorEmergenciaPorLogradouro(
            tpEmergencias[x], arrayDeIdDeLogradouros)}
        resumoOcorrenciasMes.update(dictOcorrenciasMes)

dictLinksParaHistogramas = estatisticas.
    criaDicionarioDeEstatisticas(lista_ocorrencia)
#dictLinksParaHistogramas = {}
#Pega o bairro escolhido, sera usado como centro do mapa
     na hora de carregar o mapa
bairro = cadastros.get_bairro_id(
    relacao_bairro_logradouro_do_logradouro_escolhido[0][
    'id_bairro'])

#Carrega os marcadores no mapa a partir da relacao
    bairro_logradouro, visto que essa sera a
    geolocalizacao da ocorrencia
markers_on_map = mapUtil.criar_marcadoresComGeo(
    lista_ocorrencia)

mymap = Map(
        style= "height:550px;width:1000px;margin:0;",
        zoom = 13,
identifier="view-side",
lat=bairro['latitude'],
lng=bairro['longitude'],
polygons= [poligonos.poligonoBairros(int(bairro["
    id_original_bairro"]),0)],
```

```
        markers = markers_on_map
    )

        return render_template("ocorrencias_por_logradouro(geo).
            html", nome_logradouro = logradouro_selecionado['
            nome_logradouro'], lista_ocorrencia =
            lista_ocorrencia, mymap=mymap,
            resumoOcorrenciasIncendios = resumoOcorrencias[
            tpEmergencias[0]], resumoOcorrenciasAuxilios =
            resumoOcorrencias[tpEmergencias[1]],
            resumoOcorrenciasPerigosos = resumoOcorrencias[
            tpEmergencias[2]], resumoOcorrenciasSalvamento =
            resumoOcorrencias[tpEmergencias[3]],
            resumoOcorrenciasHospitalar = resumoOcorrencias[
            tpEmergencias[4]], resumoOcorrenciasDiversos =
            resumoOcorrencias[tpEmergencias[5]],
            resumoOcorrenciasAcidente = resumoOcorrencias[
            tpEmergencias[6]], resumoOcorrenciasPreventivas =
            resumoOcorrencias[tpEmergencias[7]],
            resumoOcorrenciasArvore = resumoOcorrencias[
            tpEmergencias[8]], resumoOcorrenciasInseto =
            resumoOcorrencias[tpEmergencias[9]],
            resumoOcorrenciasMesIncendios = resumoOcorrenciasMes[
            tpEmergencias[0]], resumoOcorrenciasMesAuxilios =
            resumoOcorrenciasMes[tpEmergencias[1]],
            resumoOcorrenciasMesPerigosos = resumoOcorrenciasMes[
            tpEmergencias[2]], resumoOcorrenciasMesSalvamentos =
            resumoOcorrenciasMes[tpEmergencias[3]],
            resumoOcorrenciasMesHospitalar = resumoOcorrenciasMes
            [tpEmergencias[4]], resumoOcorrenciasMesDiversos =
            resumoOcorrenciasMes[tpEmergencias[5]],
            resumoOcorrenciasMesAcidentes = resumoOcorrenciasMes[
            tpEmergencias[6]], resumoOcorrenciasMesPreventivas =
            resumoOcorrenciasMes[tpEmergencias[7]],
            resumoOcorrenciasMesArvore = resumoOcorrenciasMes[
            tpEmergencias[8]], resumoOcorrenciasMesInseto =
            resumoOcorrenciasMes[tpEmergencias[9]],
            dictLinksParaHistogramas=dictLinksParaHistogramas)
```

```
@app.route("/ocorrencias/bairros/Charts/<chart>")
def carregaChartBairros(chart):
        stringDoArquivo = "Charts/" + str(chart)
        return render_template(stringDoArquivo)

@app.route("/ocorrencias/logradouros/Charts/<chart>")
def carregaChartLogradouros(chart):
        stringDoArquivo = "Charts/" + str(chart)
        return render_template(stringDoArquivo)

@app.route("/ocorrencias/logradouros/Geo/Charts/<chart>")
def carregaChartLogradourosGeo(chart):
        stringDoArquivo = "Charts/" + str(chart)
        return render_template(stringDoArquivo)

if __name__ == "__main__":
    #app.run(debug=False, use_reloader=True)
    # caso tenha problemas com multithreading na hora de inserir
        o registro no db use   #return base_html.format(title=
       ocorrencia['bairro'], body=ocorrencia_html)
    app.run(debug=False, use_reloader=False)
```

cadastros.py:

```
# coding: utf-8

from flask import Flask, request, url_for, render_template
from flask_googlemaps import GoogleMaps
from flask_googlemaps import Map
from db import ocorrencias, bairro_logradouro, logradouros,
   bairros, ocorrencias, tp_emergencia, db,
   ocorrenciasComGeolocalizacao
import random
import decimal
import collections
import time

#Revisado
def cadastrar_bairro():
        file = open("CSV-extraidos/bairros-tratados(true).csv","
           r")
```

```
        textoDoArquivo = file.read()
        file.close
        bairrosExtraidos = textoDoArquivo.split("\n")
        total = 0
        for x in range(len(bairrosExtraidos)-2):
                x=x+1
                dado = bairrosExtraidos[x].split(",")
                novo_bairro = {'id_original_bairro':int(dado[0])
                    ,'nome_bairro': dado[1].replace("␣","-"), '
                    latitude':dado[2], 'longitude':dado[3]}
                bairros.insert(novo_bairro)
                total = total +1

#Revisado
def cadastrar_logradouros():
        file = open("CSV-extraidos/logradouros-tratados(true).
            csv","r")
        textoDoArquivo = file.read()
        file.close
        logradourosExtraidos = textoDoArquivo.split("\n")
        total = 0
        for x in range(len(logradourosExtraidos)-2):
                x=x+1
                dado = logradourosExtraidos[x].split(",")
                novo_logradouro = {'id_original_logradouro':int(
                    dado[0]),'id_original_logradouro':int(dado
                    [1]),'nome_logradouro': dado[2].replace("␣","
                    -")}
                logradouros.insert(novo_logradouro)
                total = total +1

#Revisado
def cadastrar_relacionamento():
        file = open("CSV-extraidos/logradouros-tratados(true).
            csv","r")
        textoDoArquivo = file.read()
        file.close
        logradourosExtraidos = textoDoArquivo.split("\n")
        total = 0
```

```
        totalErro = 0
        for x in range(len(logradourosExtraidos)-2):
                x=x+1
                dadosLogradouro = logradourosExtraidos[x].split(
                    ",")

                # Variavel auxiliar para gerar localiza   o
                    para testar
                latitude = dadosLogradouro[3]
                longitude = dadosLogradouro[4]

                logradouro = logradouros.find_one(
                    id_original_logradouro=int(dadosLogradouro
                    [1]))
                bairro = bairros.find_one(id_original_bairro=int
                    (dadosLogradouro[0]))

                try:
                        novo_relacionamento = {'id_bairro':
                            bairro['id'], 'id_logradouro':
                            logradouro['id'], 'latitude':
                            latitude, 'longitude' : longitude}
                        bairro_logradouro.insert(
                            novo_relacionamento)
                        total = total + 1
                except:
                        totalErro = totalErro +1

#Revisado
def get_logradouro(logradouro):
        logradouro_selecionado = logradouros.find_one(
            nome_logradouro = logradouro)
        return logradouro_selecionado
#Revisado
def get_logradouro_id(id_logradouro):
        logradouro_selecionado = logradouros.find_one(id =
            id_logradouro)
        return logradouro_selecionado
```

```
#Revisado
def get_todos_logradouros():
        logradouro_selecionado = logradouros.find()
        return logradouro_selecionado

#Revisado
def get_todos_bairros():
        logradouro_selecionado = bairros.find()
        return logradouro_selecionado

#Revisado
def cadastrar_ocorrencias():
        file = open("CSV-extraidos/Ocorrencias/ocorrencias-
           filtradas.csv","r")
        textoDoArquivo = file.read()
        file.close
        ocorrenciasExtraidas = textoDoArquivo.split("\n")
        total = 0
        total_com_erro = 0
        stringParaErro = ""
        for x in range(int(len(ocorrenciasExtraidas))-1):
                dado = ocorrenciasExtraidas[x].split(",")
                id_do_logradouro = logradouros.find_one(
                   id_original_logradouro = dado[1])
                try:
                        idRelacao = bairro_logradouro.find_one(
                           id_logradouro = list(id_do_logradouro
                           .values())[0])
                        nova_ocorrencia = {"id_original_bairro":
                            dado[0],'id_bairro_logradouro':
                           idRelacao['id'],'dia': dado[3], '
                           hora_da_ocorrencia':dado[15], '
                           tp_emergencia':tp_emergencia.find_one
                           (id = dado[2])['tp_emergencia'],'
                           id_tp_emergencia':dado[2], '
                           precipitacao':dado[5], '
                           temperatura_maxima':dado[6], '
                           temperatura_minima':dado[7], '
                           insolacao':dado[8], 'evaporacao_piche
```

```
                    ':dado[9], 'temperatura_comp_media':
                    dado[10], 'umidade_relativa_media':
                    dado[11], 'velocidade_vento_media':
                    dado[12], 'dia_da_semana':dado[14], '
                    estacao_do_ano':dado[16], 'ano':dado
                    [17], 'mes':dado[18], 'dia_do_mes':
                    dado[13]}
            except:
                    total_com_erro = total_com_erro + 1
                    stringParaErro = stringParaErro + str(
                        dado[0]) + "\n"
            try:
                    ocorrencias.insert(nova_ocorrencia)
                    total = total + 1
            except:
                    pass
    file = open("CSV-extraidos/logLogradourosErro.csv","w")
    textoDoArquivo = file.write(stringParaErro)
    file.close

#Revisado
def cadastrar_ocorrenciasComGeolocalizacao():
    file = open("CSV-extraidos/Ocorrencias/ocorrencias-
        filtradas-geolocalizacao.csv","r")
    textoDoArquivo = file.read()
    file.close
    ocorrenciasExtraidas = textoDoArquivo.split("\n")
    total = 0
    total_com_erro = 0
    stringParaErro = ""
    for x in range(int(len(ocorrenciasExtraidas))-2):
            x = x + 1
            dado = ocorrenciasExtraidas[x].split(",")
            id_do_logradouro = logradouros.find_one(
                id_original_logradouro = dado[1][2:-1])
            try:
                    idRelacao = bairro_logradouro.find_one(
                        id_logradouro = list(id_do_logradouro
                        .values())[0])
```

```
                    nova_ocorrencia = {"id_original_bairro":
                        dado[0], 'id_bairro_logradouro':
                        idRelacao['id'], 'dia': dado[3][2:-1],
                        'hora_da_ocorrencia':dado[15], '
                        tp_emergencia':tp_emergencia.find_one
                        (id = dado[2][2:-1])['tp_emergencia'
                        ], 'id_tp_emergencia':dado[2][2:-1], '
                        precipitacao':dado[5], '
                        temperatura_maxima':dado[6][2:-1], '
                        temperatura_minima':dado[7][2:-1], '
                        insolacao':dado[8], 'evaporacao_piche
                        ':dado[9][2:-1], '
                        temperatura_comp_media':dado
                        [10][2:-1], 'umidade_relativa_media':
                        dado[11], 'velocidade_vento_media':
                        dado[12][2:-1], 'diaDoMes' : dado
                        [13], 'dia_da_semana':dado[14],'hora'
                        : dado[15], 'estacao_do_ano':dado
                        [16], 'ano':dado[17], 'mes':dado[18],
                        'lat' : dado[19], 'lng' : dado[20]}
                except:
                        total_com_erro = total_com_erro + 1
                        stringParaErro = stringParaErro + str(
                            dado[1]) + "\n"
                ocorrenciasComGeolocalizacao.insert(
                    nova_ocorrencia)
                try:
                        ocorrenciasComGeolocalizacao.insert(
                            nova_ocorrencia)
                        total = total + 1
                except:
                        pass
        file = open("CSV-extraidos/logLogradourosErro.csv","w")
        textoDoArquivo = file.write(stringParaErro)
        file.close

#Melhor guardar
def get_ocorrencias(bairro_logradouros):
```

```
        lista_bairro_logradouros = bairro_logradouros
        lista_ocorrencias = []
        for x in range(len(lista_bairro_logradouros)):
                lista_de_ocorrencias_do_bairro_logradouro = list
                   (ocorrencias.find(id_bairro_logradouro =
                   lista_bairro_logradouros[x]['id']))
                for y in range(len(
                   lista_de_ocorrencias_do_bairro_logradouro)):
                        lista_ocorrencias.append(
                           lista_de_ocorrencias_do_bairro_logradouro
                           [y])
        return lista_ocorrencias

def get_ocorrenciasGeo(bairro_logradouros):
        lista_bairro_logradouros = bairro_logradouros
        lista_ocorrencias = []
        for x in range(len(lista_bairro_logradouros)):
                lista_de_ocorrencias_do_bairro_logradouro = list
                   (ocorrenciasComGeolocalizacao.find(
                   id_bairro_logradouro =
                   lista_bairro_logradouros[x]['id']))
                for y in range(len(
                   lista_de_ocorrencias_do_bairro_logradouro)):
                        lista_ocorrencias.append(
                           lista_de_ocorrencias_do_bairro_logradouro
                           [y])
        return lista_ocorrencias

#Revisado
def getOcorrenciasOptions(id_bairro_logradouro, tipo_emergencia,
    anoEscolhido, estacao, diaDaSemana, option):
        if(option == 0):
                return list(ocorrencias.find(
                   id_bairro_logradouro = id_bairro_logradouro,
                   tp_emergencia = tipo_emergencia, ano =
                   anoEscolhido, estacao_do_ano = estacao,
                   dia_da_semana = diaDaSemana))
        if(option == 1):
                return list(ocorrencias.find(
```

```
                id_bairro_logradouro = id_bairro_logradouro,
                ano = anoEscolhido, estacao_do_ano = estacao,
                 dia_da_semana = diaDaSemana))
    if(option == 2):
            return list(ocorrencias.find(
                id_bairro_logradouro = id_bairro_logradouro,
                tp_emergencia = tipo_emergencia,
                estacao_do_ano = estacao, dia_da_semana =
                diaDaSemana))
    if(option == 3):
            return list(ocorrencias.find(
                id_bairro_logradouro = id_bairro_logradouro,
                estacao_do_ano = estacao, dia_da_semana =
                diaDaSemana))
    if(option == 4):
            return list(ocorrencias.find(
                id_bairro_logradouro = id_bairro_logradouro,
                tp_emergencia = tipo_emergencia, ano =
                anoEscolhido, dia_da_semana = diaDaSemana))
    if(option == 5):
            return list(ocorrencias.find(
                id_bairro_logradouro = id_bairro_logradouro,
                ano = anoEscolhido, dia_da_semana =
                diaDaSemana))
    if(option == 6):
            return list(ocorrencias.find(
                id_bairro_logradouro = id_bairro_logradouro,
                tp_emergencia = tipo_emergencia,
                dia_da_semana = diaDaSemana))
    if(option == 7):
            return list(ocorrencias.find(
                id_bairro_logradouro = id_bairro_logradouro,
                dia_da_semana = diaDaSemana))
    if(option == 8):
            return list(ocorrencias.find(
                id_bairro_logradouro = id_bairro_logradouro,
                tp_emergencia = tipo_emergencia, ano =
                anoEscolhido, estacao_do_ano = estacao))
    if(option == 9):
```

```
                return list(ocorrencias.find(
                    id_bairro_logradouro = id_bairro_logradouro,
                    ano = anoEscolhido, estacao_do_ano = estacao)
                    )
        if(option == 10):
                return list(ocorrencias.find(
                    id_bairro_logradouro = id_bairro_logradouro,
                    tp_emergencia = tipo_emergencia,
                    estacao_do_ano = estacao))
        if(option == 11):
                return list(ocorrencias.find(
                    id_bairro_logradouro = id_bairro_logradouro,
                    estacao_do_ano = estacao))
        if(option == 12):
                return list(ocorrencias.find(
                    id_bairro_logradouro = id_bairro_logradouro,
                    tp_emergencia = tipo_emergencia, ano =
                    anoEscolhido))
        if(option == 13):
                return list(ocorrencias.find(
                    id_bairro_logradouro = id_bairro_logradouro,
                    ano = anoEscolhido))
        if(option == 14):
                return list(ocorrencias.find(
                    id_bairro_logradouro = id_bairro_logradouro,
                    tp_emergencia = tipo_emergencia))
        if(option == 15):
                return list(ocorrencias.find(
                    id_bairro_logradouro = id_bairro_logradouro))
#Revisado
def get_ocorrenciasComFiltro(bairro_logradouros, tipo_emergencia
    , anoEscolhido, estacao, diaDaSemana, option):
        lista_bairro_logradouros = bairro_logradouros

        lista_ocorrencias = []
        for x in range(len(lista_bairro_logradouros)):
                lista_de_ocorrencias_do_bairro_logradouro =
                    getOcorrenciasOptions(
                    lista_bairro_logradouros[x]['id'],
```

```
                    tipo_emergencia, anoEscolhido, estacao,
                    diaDaSemana, option);
                for y in range(len(
                    lista_de_ocorrencias_do_bairro_logradouro)):
                        lista_ocorrencias.append(
                            lista_de_ocorrencias_do_bairro_logradouro
                            [y])

        return lista_ocorrencias

#Revisado
def ocorrenciasPorAnoPorEmergencia(emergencia):
        ocorrenciasDoAno = []
        auxiliarOcorrenciasDoAno = []
        i = 0
        for x in range(2007,2018):
                if(x == 2007):
                        ocorrenciaDoAnoAtual = ocorrencias.count
                            (ano = x, tp_emergencia = emergencia)
                        ocorrenciasDoAno.append(
                            ocorrenciaDoAnoAtual)
                        auxiliarOcorrenciasDoAno.append(
                            ocorrenciaDoAnoAtual)
                else:
                        i = i + 1
                        ocorrenciaDoAnoAtual = ocorrencias.count
                            (ano = x, tp_emergencia = emergencia)
                        ocorrenciasDoAno.append(
                            ocorrenciaDoAnoAtual)
                        auxiliarOcorrenciasDoAno.append(
                            ocorrenciaDoAnoAtual)
                        try:
                                variacao = (ocorrenciaDoAnoAtual
                                    -auxiliarOcorrenciasDoAno[i
                                    -1])/auxiliarOcorrenciasDoAno
                                    [i-1]*100
                                if(variacao > 0):
                                        variacaoAnoAnterior = "+
                                            " + format(variacao,'
```

```
                                        .2f')
                                else:
                                        variacaoAnoAnterior =
                                            format(variacao,'.2f'
                                            )
                            except:
                                        variacaoAnoAnterior = "N/A"
                            ocorrenciasDoAno.append(
                                variacaoAnoAnterior)
        return ocorrenciasDoAno

def ocorrenciasPorMesPorEmergenciaPorLogradouro(emergencia,
    logradouros):
        meses = ['01','02','03','04','05','06','07','08','09','
            10','11','12']
        ocorrenciasDoAno = {}
        ocorrenciasDoAnoNovo = {}
        statement = "SELECT count(insolacao) as
            totalDeOcorrencias, ano, mes from ocorrencias where
            tp_emergencia = '" + str(emergencia) + "' and
            id_bairro_logradouro in " + str(logradouros).replace(
            "[","(").replace("]",")") + " group by ano, mes"

        result = get_densidades_bairro(statement)
        for x in range(2007,2019):
                dictDoAno = {x : {}}
                ocorrenciasDoAnoNovo.update( dictDoAno)
                for y in meses:
                        ocorrenciasDoAnoNovo[int(x)].update({ y
                            : 0})
        for row in result:
                ocorrenciasDoAnoNovo[int(row["ano"])][row["mes"
                    ]] = row["totalDeOcorrencias"]

        for x in range(2007,2018):
                arrayDeContagemDeOcorrencias = []
                auxiliarOcorrenciasDoMesAno = []
                i = 0
```

```
for y in meses:
        if(y == '01'):
                numeroDeOcorrenciasDaqueleMes =
                    ocorrenciasDoAnoNovo[x][y]
                arrayDeContagemDeOcorrencias.
                    append(
                    numeroDeOcorrenciasDaqueleMes
                    )
                auxiliarOcorrenciasDoMesAno.
                    append(
                    numeroDeOcorrenciasDaqueleMes
                    )
        else:
                i = i + 1
                numeroDeOcorrenciasDaqueleMes =
                    ocorrenciasDoAnoNovo[x][y]
                arrayDeContagemDeOcorrencias.
                    append(
                    numeroDeOcorrenciasDaqueleMes
                    )
                auxiliarOcorrenciasDoMesAno.
                    append(
                    numeroDeOcorrenciasDaqueleMes
                    )
                try:
                        variacao = (
                            numeroDeOcorrenciasDaqueleMes
                            -
                            auxiliarOcorrenciasDoMesAno
                            [i-1])/
                            auxiliarOcorrenciasDoMesAno
                            [i-1]*100
                        if(variacao > 0):
                                variacaoMesAnterior
                                    = "+" +
                                format(
                                variacao, '.2f
                                ')
                        else:
```

```
                                                variacaoMesAnterior
                                                    = format(
                                                    variacao,'.2f
                                                    ')
                                        except:
                                                variacaoMesAnterior = "N
                                                    /A"
                                        arrayDeContagemDeOcorrencias.
                                            append(variacaoMesAnterior)
                    dictDasOcorrenciasDoAno = {x:
                        arrayDeContagemDeOcorrencias}
                    ocorrenciasDoAno.update(dictDasOcorrenciasDoAno)
            return ocorrenciasDoAno

#Revisado
def ocorrenciasPorAnoPorEmergenciaPorLogradouros(emergencia,
    logradouros):
            ocorrenciasDoAno = []
            ocorrenciasDoAnoNovo = {}
            statement = "SELECT count(insolacao) as
                totalDeOcorrencias, ano from ocorrencias where
                tp_emergencia = '" + str(emergencia) + "' and
                id_bairro_logradouro in " + str(logradouros).replace(
                "[","(").replace("]",")") + " group by ano"
            result = get_densidades_bairro(statement)
            for x in range(2007,2019):
                    dictDoAno = {x : 0}
                    ocorrenciasDoAnoNovo.update(dictDoAno)
            for row in result:
                    ocorrenciasDoAnoNovo[int(row["ano"])] = row["
                        totalDeOcorrencias"]

            auxiliarOcorrenciasDoAno = []
            i = 0
            for x in range(2007,2018):
                    if(x == 2007):
                            ocorrenciaDoAnoAtual =
                                ocorrenciasDoAnoNovo[x]
                            ocorrenciasDoAno.append(
```

```
                        ocorrenciaDoAnoAtual)
                    auxiliarOcorrenciasDoAno.append(
                        ocorrenciaDoAnoAtual)
                else:
                    i = i + 1
                    ocorrenciaDoAnoAtual =
                        ocorrenciasDoAnoNovo[x]
                    ocorrenciasDoAno.append(
                        ocorrenciaDoAnoAtual)
                    auxiliarOcorrenciasDoAno.append(
                        ocorrenciaDoAnoAtual)
                    try:
                        variacao = (ocorrenciaDoAnoAtual
                            -auxiliarOcorrenciasDoAno[i
                            -1])/auxiliarOcorrenciasDoAno
                            [i-1]*100
                        if(variacao > 0):
                            variacaoAnoAnterior = "+
                                " + format(variacao,'
                                .2f')
                        else:
                            variacaoAnoAnterior =
                                format(variacao,'.2f'
                                )
                    except:
                        variacaoAnoAnterior = "N/A"
                    ocorrenciasDoAno.append(
                        variacaoAnoAnterior)
        return ocorrenciasDoAno

#Revisado
def cadastrar_tp_emergencia():
        novo_tp_emergencia = {'tp_emergencia': 'INCENDIO'}
        tp_emergencia.insert(novo_tp_emergencia)
        novo_tp_emergencia = {'tp_emergencia': 'AUXILIOS / 
            APOIOS'}
        tp_emergencia.insert(novo_tp_emergencia)
        novo_tp_emergencia = {'tp_emergencia': 'PRODUTOS 
            PERIGOSOS'}
```

```
        tp_emergencia.insert(novo_tp_emergencia)
        novo_tp_emergencia = {'tp_emergencia': 'SALVAMENTO / BUSCA / RESGATE'}
        tp_emergencia.insert(novo_tp_emergencia)
        novo_tp_emergencia = {'tp_emergencia': 'ATENDIMENTO PRE-HOSPITALAR'}
        tp_emergencia.insert(novo_tp_emergencia)
        novo_tp_emergencia = {'tp_emergencia': 'OCORRENCIA NAO ATENDIDA'}
        tp_emergencia.insert(novo_tp_emergencia)
        novo_tp_emergencia = {'tp_emergencia': 'DIVERSOS'}
        tp_emergencia.insert(novo_tp_emergencia)
        novo_tp_emergencia = {'tp_emergencia': 'ACIDENTE DE TRANSITO'}
        tp_emergencia.insert(novo_tp_emergencia)
        novo_tp_emergencia = {'tp_emergencia': 'ACOES PREVENTIVAS'}
        tp_emergencia.insert(novo_tp_emergencia)
        novo_tp_emergencia = {'tp_emergencia': 'AVERIGUACAO / CORTE DE ARVORE'}
        tp_emergencia.insert(novo_tp_emergencia)
        novo_tp_emergencia = {'tp_emergencia': 'AVERIGUACAO / MANEJO DE INSETO'}
        tp_emergencia.insert(novo_tp_emergencia)

#Revisado
def get_relacao_bairro_logradouro(id):
        return bairro_logradouro.find_one(id = id)

#Revisado
def get_relacao_bairro(nome_do_bairro):
        bairro = bairros.find_one(nome_bairro = nome_do_bairro)
        return list(bairro_logradouro.find(id_bairro = bairro['id']))
#Revisado
def get_relacao_logradouro(nome_do_logradouro):
        logradouro = logradouros.find_one(nome_logradouro = nome_do_logradouro)
        return list(bairro_logradouro.find(id_logradouro =
```

```
            logradouro['id']))
#Revisado
def get_bairro(nome_do_bairro):
        bairro = bairros.find_one(nome_bairro = nome_do_bairro)
        return bairro

#Revisado
def get_bairro_id(id):
        bairro = bairros.find_one(id = id)
        return bairro

def get_densidades_bairro(statement):
        result = db.query(statement)
        return result

def getAll(tipoEmergencia, ano, estacao, diaDaSemana):
        state = 0;
        statement = "SELECT * from ocorrencias "
        if(tipoEmergencia != "Todos"):
                state = state + 1
                statement = statement + "where tp_emergencia = '
                   " + str(tipoEmergencia) + "'"
        if(ano != "Todos"):
                if(state > 0):
                        statement = statement + " AND ano = " +
                           str(ano)
                else:
                        statement = statement + "where ano = " +
                            str(ano)
                state = state + 2
        if(estacao != "Todos"):
                if(state > 0):
                        statement = statement + " AND
                           estacao_do_ano = '" + str(estacao) +
                           "'"
                else:
                        statement = statement + "where
                           estacao_do_ano = '" + str(estacao) +
                           "'"
```

```
                state = state + 4
        if(diaDaSemana != "Todos"):
                if(state > 0):
                        statement = statement + "␣AND␣
                           dia_da_semana␣=␣'" + str(diaDaSemana)
                            + "'"
                else:
                        statement = statement + "where␣
                           dia_da_semana␣=␣'" + str(diaDaSemana)
                            + "'"
                state = state + 8
        result = list(ocorrencias.find())
        return result

def statementBuilder(tipoEmergencia, ano, estacao, diaDaSemana):
        state = 0;
        statement = "SELECT␣count(insolacao)␣as␣
           totalDeOcorrencias,␣id_original_bairro␣from␣
           ocorrencias␣"
        if(tipoEmergencia != "Todos"):
                state = state + 1
                statement = statement + "where␣tp_emergencia␣=␣'
                   " + str(tipoEmergencia) + "'"
        if(ano != "Todos"):
                if(state > 0):
                        statement = statement + "␣AND␣ano␣=␣" +
                           str(ano)
                else:
                        statement = statement + "where␣ano␣=␣" +
                            str(ano)
                state = state + 2
        if(estacao != "Todos"):
                if(state > 0):
                        statement = statement + "␣AND␣
                           estacao_do_ano␣=␣'" + str(estacao) +
                           "'"
                else:
                        statement = statement + "where␣
                           estacao_do_ano␣=␣'" + str(estacao) +
```

```
                        "'"
                state = state + 4
        if(diaDaSemana != "Todos"):
                if(state > 0):
                        statement = statement + " AND
                           dia_da_semana = '" + str(diaDaSemana)
                            + "'"
                else:
                        statement = statement + "where
                           dia_da_semana = '" + str(diaDaSemana)
                            + "'"
                state = state + 8
        statement = statement + " group by id_original_bairro
           order by totalDeOcorrencias desc"
        return statement

def getTodasOcorrencias(lista_relacao_bairro_logradouro,
   tipoEmergencia, ano, estacao, diaDaSemana):
        lista_logradouros_do_bairro = "("
        for x in range(len(lista_relacao_bairro_logradouro)):
                lista_logradouros_do_bairro =
                   lista_logradouros_do_bairro + str(
                   lista_relacao_bairro_logradouro[x]['id']) + "
                   ,"
        lista_logradouros_do_bairro =
           lista_logradouros_do_bairro[:-1]+") "
        statement = "SELECT * from ocorrencias where
           id_bairro_logradouro in " +
           lista_logradouros_do_bairro
        if(tipoEmergencia != "Todos" and tipoEmergencia!=None):
                statement = statement + "AND tp_emergencia = '"
                   + str(tipoEmergencia) + "'"
        if(ano != "Todos" and ano!=None):
                statement = statement + " AND ano = " + str(ano)
        if(estacao != "Todos" and estacao!=None):
                statement = statement + " AND estacao_do_ano = '
                   " + str(estacao) + "'"
        if(diaDaSemana != "Todos" and diaDaSemana!=None):
                statement = statement + " AND dia_da_semana = '"
```

```
                + str(diaDaSemana) + "'"
        result = db.query(statement)
        return result

def getTodasOcorrenciasGeo(lista_relacao_bairro_logradouro,
    tipoEmergencia, ano, estacao, diaDaSemana):
        lista_logradouros_do_bairro = "("
        for x in range(len(lista_relacao_bairro_logradouro)):
                lista_logradouros_do_bairro =
                    lista_logradouros_do_bairro + str(
                    lista_relacao_bairro_logradouro[x]['id']) + "
                    ,"
        lista_logradouros_do_bairro =
            lista_logradouros_do_bairro[:-1]+")"
        statement = "SELECT * from ocorrenciasComGeolocalizacao
            where id_bairro_logradouro in " +
            lista_logradouros_do_bairro
        if(tipoEmergencia != "Todos" and tipoEmergencia!=None):
                statement = statement + "AND tp_emergencia = '"
                    + str(tipoEmergencia) + "'"
        if(ano != "Todos" and ano!=None):
                statement = statement + " AND ano = " + str(ano)
        if(estacao != "Todos" and estacao!=None):
                statement = statement + " AND estacao_do_ano = '
                    " + str(estacao) + "'"
        if(diaDaSemana != "Todos" and diaDaSemana!=None):
                statement = statement + " AND dia_da_semana = '"
                    + str(diaDaSemana) + "'"
        result = db.query(statement)
        return result
```

mapUtil.py:

```
# coding: utf-8

import cadastros
#Prefixo de todos os icones
prefix = 'http://maps.google.com/mapfiles/ms/icons/'

#dicionario id_tp_emergencia -> icone
```

```
dicionario_de_icones = { 1 : prefix + 'red-dot.png',2 : prefix +
    'green-dot.png',3 : prefix + 'blue-dot.png',4 : prefix + '
   yellow-dot.png',5 : prefix + 'msmarker.png',6 : prefix + '
   ltblue-dot.png',7 : prefix + 'orange-dot.png',8 : prefix + '
   pink-dot.png',9 : prefix + 'purple-dot.png',10 : prefix + '
   grn-pushpin.png',11 : prefix + 'ltblu-pushpin.png'}

#array com o nome de todas as emergencias
arrayTpEmergencias = [0,1,2,3,4,5,6,7,8,9,10]

dicionario_de_icones_string = { '1' : prefix + 'red-dot.png', '2
   ' : prefix + 'green-dot.png', '3' : prefix + 'blue-dot.png',
   '4' : prefix + 'yellow-dot.png', '5' : prefix + 'msmarker.png
   ', '6' : prefix + 'ltblue-dot.png', '7' : prefix + 'orange-
   dot.png', '8' : prefix + 'pink-dot.png', '9' : prefix + '
   purple-dot.png', '10' : prefix + 'grn-pushpin.png', '11' :
   prefix + 'ltblu-pushpin.png'}

def criar_marcadores(listaDeRelacoes,lista_ocorrencia):
        dictBairroLogradouroMarker = {}

        for x in range(len(listaDeRelacoes)):
                nomeLogradouro = cadastros.get_logradouro_id(
                   listaDeRelacoes[x]['id_logradouro'])['
                   nome_logradouro']
                markerBairroLogradouro = {listaDeRelacoes[x]['id
                   '] : {'quantidade':0,'nome_logradouro' :
                   nomeLogradouro, 'posicao_array':None, '
                   tp_emergencia':0}}
                for y in range(0,11):
                        dictKey = arrayTpEmergencias[y] + 1
                        dictEmergencia = {dictKey: {'quantidade'
                            : 0}}
                        markerBairroLogradouro[listaDeRelacoes[x
                           ]['id']].update(dictEmergencia)
                        dictBairroLogradouroMarker.update(
                           markerBairroLogradouro)
        markers = []
        for x in range(len(lista_ocorrencia)):
```

```
ocorrencia = lista_ocorrencia[x]
bairro_logradouro_da_ocorrencia = cadastros.
    get_relacao_bairro_logradouro(ocorrencia['
    id_bairro_logradouro'])

logradouro = dictBairroLogradouroMarker[
    bairro_logradouro_da_ocorrencia['id']]

tpEmergenciaDaOcorrencia = int(lista_ocorrencia[
    x]['id_tp_emergencia'])

quantidadeDeUmaOcorrencia = logradouro[
    tpEmergenciaDaOcorrencia]['quantidade']
quantidadeDeUmaOcorrencia =
    quantidadeDeUmaOcorrencia + 1

if(quantidadeDeUmaOcorrencia > logradouro[
    tpEmergenciaDaOcorrencia]['quantidade']):
        logradouro['quantidade'] =
            quantidadeDeUmaOcorrencia
        logradouro['tp_emergencia'] =
            tpEmergenciaDaOcorrencia

new_mark = {
                'icon': {'url':
                    dicionario_de_icones_string[
                    lista_ocorrencia[x]['
                    id_tp_emergencia']]},
                'lat':
                    bairro_logradouro_da_ocorrencia
                    ['latitude'],
                'lng':
                    bairro_logradouro_da_ocorrencia
                    ['longitude'],
                'infobox': "<b>"+ logradouro['
                    nome_logradouro'] + "</b>"
                }
if(logradouro['posicao_array'] == None):
        markers.append(new_mark)
```

```
                        logradouro['posicao_array'] = len(
                            markers)-1
                else:
                        markers[logradouro['posicao_array']] =
                            new_mark
        return markers

def criar_marcadoresComGeo(lista_ocorrencia):
        markers = []
        for x in range(len(lista_ocorrencia)):
                ocorrencia = lista_ocorrencia[x]
                new_mark = {
                                'icon': {'url':
                                    dicionario_de_icones_string[
                                    lista_ocorrencia[x]['
                                    id_tp_emergencia']]},
                                'lat': lista_ocorrencia[x]['lat'
                                    ],
                                'lng': lista_ocorrencia[x]['lng'
                                    ]
                                }
                markers.append(new_mark)
        return markers

def geraListaDeLogradouros(lista_relacao_bairro_logradouro):
        lista_logradouros_do_bairro = []
        for x in range(len(lista_relacao_bairro_logradouro)):
                logradouro = cadastros.get_logradouro_id(
                    lista_relacao_bairro_logradouro[x]['
                    id_logradouro'])
                lista_logradouros_do_bairro.append(logradouro['
                    nome_logradouro'])
        return lista_logradouros_do_bairro

def geraListaDeBairros(bairros):
        lista_bairros = []
        for x in range(len(bairros)):
                lista_bairros.append(bairros['nome_bairros'])
        return lista_logradouros_do_bairro
```

```
#revisar
def geraListaDeLogradouros2(logradouros):
        lista_logradouros = []
        for x in range(len(logradouros)):
                lista_logradouros.append(logradouro['
                    nome_logradouro'])
        return lista_logradouros_do_bairro

#revisar
def geraListaNomes(listaDeEntrada, stringNomeDoCampo):
        lista = []
        for x in range(len(listaDeEntrada)):
                lista.append(listaDeEntrada[x][stringNomeDoCampo
                    ])
        return lista
```

poligonos.py:

```
# coding: utf-8
import polygons
from faker import Factory

arrayPoligonos = polygons.getArrayPoligonos()
dictPoligonos = polygons.getDictPoligonos()

def poligonoTodosBairros(IdBairro):
        global arrayPoligonos
        arrayPoligons = []
        fake = Factory.create()
        corDoPoligono = fake.hex_color()
        polygons = {
        'stroke_color': "red",
        'stroke_opacity': 0.6,
        'stroke_weight': 4,
        'fill_color':"red",
        'fill_opacity':dictPoligonos[IdBairro]["Todos"],
        'path': dictPoligonos[IdBairro]["Poligonos"]
            }
        arrayPoligons.append(polygons)
```

```
        return arrayPoligons

def poligonoBairros(IdBairro, densidade):
        global dictPoligonos
        polygono = {
        'stroke_color': "red",
        'stroke_opacity': 0.6,
        'stroke_weight': 4,
        'fill_color':"red",
        'fill_opacity':densidade,
        'path': dictPoligonos[IdBairro]["Poligonos"]
            }
        return polygono

def getTodosPolygons():
        arrayPoligons = []
        fake = Factory.create()
        arrayDoDesespero = [38, 13936, 13961, 13925, 487, 13952,
            13957, 13960, 13947, 106, 13931, 13923, 13942,
           13956, 13933, 36497, 13966, 74, 13958, 13932, 13945,
           13921, 23, 116, 13927, 13950, 37025, 36464, 73, 128,
           13938, 13965, 48, 13922, 100, 36379, 36378, 13944,
           39486, 13934, 13967, 13940, 252, 13937, 3, 13924,
           13953, 148, 37129, 13949, 143, 18, 117, 121, 183,
           13948, 26, 36486, 13968, 13928, 13939, 36463, 13941,
           13964, 54, 13963, 126, 55, 52, 13962, 191, 98, 140,
           40, 264, 13959, 13926, 13946, 373, 181, 364, 362,
           225, 114, 75, 131, 363, 72, 237, 180, 115, 10000010,
           88, 10084, 360, 11, 223, 348, 118, 93]
        for x in range(len(arrayDoDesespero)):
                polygons = poligonoTodosBairros(arrayDoDesespero
                    [x])[0]
                arrayPoligons.append(polygons)
        return arrayPoligons

def getPolygons(arrayBairros, dictDensidade):
        arrayPoligons = []
        fake = Factory.create()
        counter = 0
```

```
        for x in range(len(arrayBairros)):
                counter = counter + 1
                try:
                        polygono = poligonoBairros(arrayBairros[
                           x],dictDensidade[arrayBairros[x]])
                        arrayPoligons.append(polygono)
                except:
            pass
        return arrayPoligons
```

estatisticas.py:

```
import plotly.plotly as py
import plotly
import plotly.graph_objs as go
import numpy as np
from time import gmtime, strftime

plotly.tools.set_credentials_file(username='Luanes', api_key='
   pUhBommqrfrXN1x6rYLz')
plotly.tools.set_config_file(world_readable=True,
                             sharing='public')

def criaDicionarioDeEstatisticas(listaDeOcorrencias):
        dictParaHistogramas = {'ano': {2007: 0, 2008: 0, 2009:
           0, 2010: 0, 2011: 0, 2012: 0, 2013: 0, 2014: 0, 2015:
            0, 2016: 0, 2017: 0, 2018: 0}, 'mes': {'1': 0, '2':
           0, '3': 0, '4': 0, '5': 0, '6': 0, '7': 0, '8': 0, '9
           ': 0, '10': 0, '11': 0, '12': 0}, 'diaDoMes': {'1':
           0, '2': 0, '3': 0, '4': 0, '5': 0, '6': 0, '7': 0, '8
           ': 0, '9': 0, '10': 0, '11': 0, '12': 0,'13': 0, '14'
           : 0, '15': 0, '16': 0, '17': 0, '18': 0, '19': 0, '20
           ': 0, '21': 0, '22': 0, '23': 0, '24': 0,'25': 0, '26
           ': 0, '27': 0, '28': 0, '29': 0, '30': 0, '31': 0} ,'
           diaDaSemana': {'Monday': 0, 'Tuesday': 0, 'Wednesday'
           : 0, 'Thursday': 0, 'Friday': 0, 'Saturday': 0, '
           Sunday': 0}, 'estacaoDoAno': {'Verao': 0, 'Primavera'
           : 0, 'Inverno': 0, 'Outono': 0}, 'hora': {'0': 0, '1'
           : 0, '2': 0, '3': 0, '4': 0, '5': 0, '6': 0, '7': 0,
           '8': 0, '9': 0, '10': 0, '11': 0, '12': 0, '13': 0, '
```

```
    14': 0, '15': 0, '16': 0, '17': 0, '18': 0, '19': 0,
    '20': 0, '21': 0, '22': 0, '23': 0}, 'precipitacao':
    [], 'temperaturaMinima': [], 'temperaturaMaxima': [],
     'evaporacaoPiche': [], 'velocidadeVento': [], '
    umidadeRelativa': [], 'insolacao': []}

for x in range(len(listaDeOcorrencias)):
        dictParaHistogramas["ano"][int(
           listaDeOcorrencias[x]["ano"])] =
           dictParaHistogramas["ano"][int(
           listaDeOcorrencias[x]["ano"])] + 1
        dictParaHistogramas["hora"][listaDeOcorrencias[x
           ]["hora_da_ocorrencia"]] =
           dictParaHistogramas["hora"][str(int(
           listaDeOcorrencias[x]["hora_da_ocorrencia"]))
           ] + 1
        dictParaHistogramas["mes"][listaDeOcorrencias[x
           ]["mes"]] = dictParaHistogramas["mes"][
           listaDeOcorrencias[x]["mes"]] + 1
        dictParaHistogramas["diaDaSemana"][
           listaDeOcorrencias[x]["dia_da_semana"]] =
           dictParaHistogramas["diaDaSemana"][
           listaDeOcorrencias[x]["dia_da_semana"]] + 1
        dictParaHistogramas["diaDoMes"][
           listaDeOcorrencias[x]["dia_do_mes"]] =
           dictParaHistogramas["diaDoMes"][
           listaDeOcorrencias[x]["dia_do_mes"]] + 1
        try:
                dictParaHistogramas["estacaoDoAno"][
                   listaDeOcorrencias[x]["estacao_do_ano
                   "]] = dictParaHistogramas["
                   estacaoDoAno"][listaDeOcorrencias[x][
                   "estacao_do_ano"]] + 1
        except:
                pass
        try:
                dictParaHistogramas["precipitacao"].
                   append(listaDeOcorrencias[x]["
                   precipitacao"])
```

```
            except:
                    pass
            try:
                    dictParaHistogramas["temperaturaMinima"
                        ].append(listaDeOcorrencias[x]["
                        temperatura_minima"])
            except:
                    pass
            try:
                    dictParaHistogramas["temperaturaMaxima"
                        ].append(listaDeOcorrencias[x]["
                        temperatura_maxima"])
            except:
                    pass
            try:
                    dictParaHistogramas["evaporacaoPiche"].
                        append(listaDeOcorrencias[x]["
                        evaporacao_piche"])
            except:
                    pass
            try:
                    dictParaHistogramas["velocidadeVento"].
                        append(listaDeOcorrencias[x]["
                        velocidade_vento_media"])
            except:
                    pass
            try:
                    dictParaHistogramas["umidadeRelativa"].
                        append(round(float(listaDeOcorrencias
                        [x]["umidade_relativa_media"])))
            except:
                    pass
            try:
                    dictParaHistogramas["insolacao"].append(
                        listaDeOcorrencias[x]["insolacao"])
            except:
                    pass
    return criaHistogramas(dictParaHistogramas)
```

```
def criaDicionarioDeEstatisticasGeo(listaDeOcorrencias):
        dictParaHistogramas = {'ano': {2007: 0, 2008: 0, 2009:
           0, 2010: 0, 2011: 0, 2012: 0, 2013: 0, 2014: 0, 2015:
            0, 2016: 0, 2017: 0, 2018: 0}, 'mes': {'1': 0, '2':
           0, '3': 0, '4': 0, '5': 0, '6': 0, '7': 0, '8': 0, '9
           ': 0, '10': 0, '11': 0, '12': 0}, 'diaDaSemana': {'
           Monday': 0, 'Tuesday': 0, 'Wednesday': 0, 'Thursday':
            0, 'Friday': 0, 'Saturday': 0, 'Sunday': 0}, '
           estacaoDoAno': {'Verao': 0, 'Primavera': 0, 'Inverno'
           : 0, 'Outono': 0}, 'hora': {'0': 0, '1': 0, '2': 0, '
           3': 0, '4': 0, '5': 0, '6': 0, '7': 0, '8': 0, '9':
           0, '10': 0, '11': 0, '12': 0, '13': 0, '14': 0, '15':
            0, '16': 0, '17': 0, '18': 0, '19': 0, '20': 0, '21'
           : 0, '22': 0, '23': 0}, 'precipitacao': [], '
           temperaturaMinima': [], 'temperaturaMaxima': [], '
           evaporacaoPiche': [], 'velocidadeVento': [], '
           umidadeRelativa': [], 'insolacao': []}
        return criaHistogramas(dictParaHistogramas)

def criaHistogramas(dictParaHistogramas):
        dictLinksParaHistogramas = {"Ano" : "", "Mes" : "", "
           DiaDaSemana" : "", "EstacaoDoAno" : "", "HoraDoDia" :
            "", "Precipitacao" :"", "TemperaturaMinima" : "", "
           TemperaturaMaxima":"", "Piche" : "", "VelocidadeVento
           " : "", "UmidadeRelativa" : "", "Insolacao" : ""}
        arrayPorAno = []
        for x in range(2007,2018):
                for y in range(0,dictParaHistogramas["ano"][x]):
                        arrayPorAno.append(x)

        arrayPorMes = []
        for x in range(1,13):
                for y in range(0,dictParaHistogramas["mes"][str(
                   x)]):
                        arrayMes = ["MesInexistente","Janeiro",
                           "Fevereiro", "Mar o", "Abril", "Maio
                           ", "Junho", "Julho", "Agosto", "
                           Setembro", "Outubro", "Novembro","
                           Dezembro"]
```

```
                arrayPorMes.append(arrayMes[x])

arrayPorDiaDoMes = []
for x in range(1,32):
        for y in range(0,dictParaHistogramas["diaDoMes"
            ][str(x)]):
                arrayPorDiaDoMes.append(x)

arrayPorDiaDaSemana = []
arrayDeDiasDaSemana = ["Monday", "Tuesday","Wednesday",
    "Thursday", "Friday", "Saturday", "Sunday"]
for x in range(len(arrayDeDiasDaSemana)):
        for y in range(0,dictParaHistogramas["
            diaDaSemana"][arrayDeDiasDaSemana[x]]):
                arrayDiaSemana = ["Segunda","Ter a", "
                    Quarta", "Quinta", "Sexta", "S bado"
                    , "Domingo"]
                arrayPorDiaDaSemana.append(
                    arrayDiaSemana[int(x)])
arrayPorEstacao = []
arrayDeEstacoes = ['Verao', 'Primavera', 'Inverno', '
    Outono']
for x in arrayDeEstacoes:
        for y in range(0,dictParaHistogramas["
            estacaoDoAno"][x]):
                arrayPorEstacao.append(x)

arrayPorHoraDoDia = []
for x in range(0,24):
        for y in range(0,dictParaHistogramas["hora"][str
            (x)]):
                arrayPorHoraDoDia.append(x)

arrayPorPrecipitacao = dictParaHistogramas["precipitacao
    "]
arrayPorTemperaturaMinima = dictParaHistogramas["
    temperaturaMinima"]
arrayPorTemperaturaMaxima = dictParaHistogramas["
    temperaturaMaxima"]
```

```
arrayPorEvaporacaoPiche = dictParaHistogramas["
    evaporacaoPiche"]
arrayPorVelocidadeVento = dictParaHistogramas["
    velocidadeVento"]
arrayPorUmidadeRelativa = dictParaHistogramas["
    umidadeRelativa"]
arrayPorInsolacao = dictParaHistogramas["insolacao"]

dictLinksParaHistogramas["Ano"] = criaHistograma(
    arrayPorAno, "Ano.html", "Ano")
dictLinksParaHistogramas["Mes"] = criaHistograma(
    arrayPorMes, "Meses.html", "Mes")
dictLinksParaHistogramas["DiaDaSemana"] = criaHistograma
    (arrayPorDiaDaSemana, "Dia␣da␣semana.html", "Dia␣da␣
    semana")
dictLinksParaHistogramas["EstacaoDoAno"] =
    criaHistograma(arrayPorEstacao, "Estacao␣do␣ano.html"
    , "Esta  o␣do␣ano")
dictLinksParaHistogramas["HoraDoDia"] = criaHistograma(
    arrayPorHoraDoDia, "Horario_do_dia.html", "Hora␣do␣
    dia")
dictLinksParaHistogramas["NumeroDoDia"] = criaHistograma
    (arrayPorDiaDoMes, "NumeroDoDia.html", "Dia␣do␣m s")
dictLinksParaHistogramas["Precipitacao"] =
    criaHistograma(arrayPorPrecipitacao, "Precipitacao.
    html", "Precipita  o␣acumulada␣(mm)")
dictLinksParaHistogramas["TemperaturaMinima"] =
    criaHistograma(arrayPorTemperaturaMinima, "
    Temperatura␣minima.html", "Temperadura␣m nima␣( C )"
    )
dictLinksParaHistogramas["TemperaturaMaxima"] =
    criaHistograma(arrayPorTemperaturaMaxima, "
    Temperatura␣maxima.html","Temperadura␣m xima␣( C )")
dictLinksParaHistogramas["Piche"] = criaHistograma(
    arrayPorEvaporacaoPiche, "Evaporacao␣piche.html", "
    Evapora  o␣(mm)")
dictLinksParaHistogramas["VelocidadeVento"] =
    criaHistograma(arrayPorVelocidadeVento, "Velocidade␣
    media␣do␣vento.html", "Velocidade␣m dia␣do␣vento␣(Km
```

```
        /h)")
    dictLinksParaHistogramas["UmidadeRelativa"] =
        criaHistograma(arrayPorUmidadeRelativa, "Umidade␣
        relativa␣do␣ar␣(␣%␣).html", "Umidade␣relativa␣do␣ar␣(
        mm)")
    dictLinksParaHistogramas["Insolacao"] = criaHistograma(
        arrayPorInsolacao, "Insolacao.html", "Insola   o␣(W/
        m )")
    return dictLinksParaHistogramas

def criaHistograma(arrayDeDados, nomeDoArquivo, eixoX):
    x = arrayDeDados
    data = [go.Histogram(x=x)]
    layout = go.Layout(
        title=go.layout.Title(
            text='Quantidade␣de␣ocorrencia␣x␣' +
                nomeDoArquivo.replace(".html",""),
            xref='paper',
            x=0,
            y=0
        ),
        xaxis=go.layout.XAxis(
            title=go.layout.xaxis.Title(
                text=eixoX,
                font=dict(
                    family='Courier␣New,␣monospace',
                    size=15,
                    color='#7f7f7f'
                )
            )
        ),
        yaxis=go.layout.YAxis(
            title=go.layout.yaxis.Title(
                text='Quantidade␣de␣ocorrencias',
                font=dict(
                    family='Courier␣New,␣monospace',
                    size=15,
                    color='#7f7f7f'
                )
```

```
                )
            )
        )
        timeNow = strftime("%m:%d:%H:%M:%S", gmtime())
        fig = go.Figure(data=data, layout=layout)
        linkParaOHistograma = plotly.offline.plot(fig, filename=
            "templates/Charts/"+ str(timeNow) + nomeDoArquivo,
            auto_open=False)
        linkParaOHistograma = str(linkParaOHistograma)

        return linkParaOHistograma
```

db.py:

```
# coding: utf-8

import dataset

db = dataset.connect('sqlite:///webPageBombeiros.db')
ocorrencias = db['ocorrencias']
ocorrenciasComGeolocalizacao = db['ocorrenciasComGeolocalizacao'
    ]
bairro_logradouro = db['bairro_logradouro']
logradouros = db['logradouros']
bairros = db['bairros']
tp_emergencia = db['tp_emergencia']
```