Trabalho de Conclusão de Residência

Projeto de Melhoria: Desenvolvimento de protótipos inovadores baseados em Ciência de Dados e Automatização na Prefeitura de São Paulo

Autor: Silvio Miyadaira Amâncio - 943257

Orientador: Bruno Martinelli

Avaliador:

Resumo: Ferramentas de tratamento de dados estão disponíveis no pacote MSOffice365, adquirido pela Prefeitura do Município de São Paulo (PSMP) sob a forma de contratos renováveis. Dentre as ferramentas disponíveis neste pacote, encontra-se o PowerAutomate, que permite customização de soluções de automatização de dados. Diversas tarefas podem ser facilitadas ou ter seu desempenho incrementado pelo uso de tais ferramentas, entretanto, nem todas as licenças disponibilizadas aos usuários permitem a realização de tarefas relacionadas, desta forma, foram testadas e desenvolvidas soluções mistas, baseadas tanto no PowerAutomate como em ferramentas de programação de código aberto e gratuitas como o Python e Postgresql.

TEMA: Transformação Digital e Redesenho de Serviços Públicos, com foco no usuário, Desenho de Indicadores para Monitoramento de Políticas Públicas, Promoção da Integridade, Transparência e Acesso à informação, Racionalização de Processos.

Palavras-chave: Automatização, Ciência de Dados, Transformação Digital, Racionalização de Processos

Minicurrículo: Residente em Gestão Pública SEGES/Lab11 (PMSP); Engenheiro Eletricista e Mestre em Sistemas Digitais (POLI-USP); MBA em Engenharia da Gestão e Qualidade

1. Introdução

Grandes cidades em todo o mundo apresentam muitos fatores em comum, dentre eles situa-se o estudo e tratamento de condições de trânsito visando aumento de sua fluidez. Painéis de dados ou baseados na coleta e tratamento dos mesmos podem prover *insights* acerca dos mais diversos aspectos que corriqueiramente não são notados [1]. Uma estimativa dos custos médios de desperdícios no trânsito dos EUA é apresentada em [2] como da ordem de US$ 13.480 nos últimos 5 anos. Cidades como a cidade do México e Londres, por exemplo, buscam aumentar a fluidez de seus sistemas através de projetos de incentivos de substituição de veículos de transporte público com mais de dez anos de idade e implantação de anéis viários, respectivamente. Em [3] é apresentada a modelagem, a partir de dados de GPS obtidos por aplicativos de táxis, para o trânsito na cidade do Porto – Portugal. Os congestionamentos, acidentes, alagamentos, quedas de árvores dentre outras ocorrências, geram, além dos prejuízos à saúde mental e física do cidadão e transeunte em geral, perdas diretamente relacionadas ao tempo, ocasionando danos financeiramente mensuráveis tanto às pessoas quanto à economia.

O conceito de *Big Data and Open Linked Data – BOLD* [4] é um importante campo de pesquisa relacionado às cidades inteligentes (inovações e governança da cidade e nos espaços urbanos suportadas por tecnologia) que é recente e tem evoluído rapidamente, e refere-se aos aspectos não só de tamanho, mas também de Volume, Velocidade, Variedade, Valor, Variabilidade e Veracidade que são aplicáveis na otimização da utilização de recursos e sugestão de melhorias. Já o termo Open Data está relacionado à sua disponibilidade gratuita.

O crescimento da adoção da Inteligência Artificial na Administração Pública tem se expandido rapidamente nos últimos anos, tendo grande parte da pesquisa sido focada no desenvolvimento de sistemas baseados em algoritmos para decisões automatizadas. A crítica a estas aplicações fundamenta-se na possibilidade de opacidade e falta de explicações entre as relações causais de entrada e saída, a dificuldade em se determinar se o tratamento às demandas tem sido feito de maneira consistente e que não sejam introduzidos vieses. A adoção do *Machine Learning* ao invés da aplicação das Regras de Negócio é questionada pois ambos estão sujeitos aos vieses e tomadas de decisão incorretas [5], mas apresenta como vantagens a

otimização de processos e economia de recursos, principalmente na realização de tarefas trabalhosas e repetitivas.

Embora existam imensos volumes de dados disponíveis na administração pública em geral, eles ainda podem não estar suficientemente acessíveis para pesquisas específicas, apesar de diversas iniciativas e esforços, sendo provável a ocorrência de perda de informações que podem vir a ser posteriormente úteis em pesquisas diversas. As práticas ETL (*Extract Transform Load*) e ELT (*Extract Load Transform*) referem-se à maneira como é realizado o processo de obtenção e armazenagem de dados, ao escolher-se uma prática ao invés da outra, obtém-se diferentes resultados na estrutura de seu armazenamento (respectivamente *Datawarehouse* – dados já processados e sem ruídos ou *Datalake* – dados sem processamentos e análises). Estando interessados nos dados não agregados, de forma a permitir futuras pesquisas, optou-se pela estrutura *Datalake*, em uma arquitetura NoSQL (base de dados não relacional). É ainda necessário que sejam observados diversos aspectos, tais como qualidade, confidencialidade, integridade, detecção e tratamento de erros, dentre outros [6].

Protótipos funcionais [7] (também conhecidos como modelos ou *mock-ups*) são versões que são simulações de produtos reais, ou versões iniciais de aplicativos que visam demonstrar a viabilidade da implementação de novas soluções sem necessariamente investir grandes quantidades de esforço e recursos. Inovação é o processo de criação no qual são desenvolvidos novos produtos, métodos ou ideias que resultam em melhorias nos mesmos.

Foram desenvolvidos quatro protótipos inovadores (“Protótipos são ferramentas fundamentais para explorar ideias inovadoras e transformar conceitos em realidades tangíveis que podem ser avaliadas e evoluídas.” [8]) de aplicações/serviços para melhoria dos processos realizados pela PMSP relacionados à utilização e tratamento de dados, apresentados nos itens 1.1 a 1.4. Os protótipos desenvolvidos, embora iniciais, visam demonstrar a viabilidade de sua utilização nos referidos serviços, e apresentam restrições de funcionamento em função de características de volume e priorização de processos de acordo com a licença do MS Office 365 atualmente disponível.

1.1. Protótipo de Automatizador de respostas a emails e chatbot

Ferramentas de automatização de processos como o PowerAutomate podem reduzir o trabalho de classificação de dados, realizando tarefas extensas e trabalhosas como classificação e respostas de mensagens. Tal processamento pode envolver algoritmos de identificação de palavras ou avaliação da similaridade de textos, como o Processamento de Linguagem Natural (NLP – *Natural Language Processing*). Para o segundo caso, não há suporte na versão do PowerAutomate disponibilizada, sendo necessário seu tratamento por meio de outra ferramenta (Python). A ferramenta de programação Python, é uma ferramenta gratuita, a qual possui muitas bibliotecas que podem ser aplicadas à realização de uma grande variedade de tarefas, e foi bastante utilizada nas tarefas descritas neste trabalho.

Foi desenvolvido, utilizando-se o PowerAutomate, um protótipo de aplicação para resposta automatizada a perguntas de usuários (alunos da EMASP – Escola Municipal de Administração do Município de São Paulo) realizada via e-mail e chatbot no MS-TEAMS.

1.2. Levantamento exploratório de dados

A Prefeitura do Município de São Paulo (PMSP) dispõe de diversas e imensas bases de dados, algumas das quais disponíveis de forma bruta para acesso via APIs e *websites* (como o do Centro de Gerenciamento de Emergências - CGE [11])*;* visando a aplicação deles para realizar tal tarefa, é necessário que sejam disponibilizados em bases adequadas (principalmente acessíveis de acordo com a necessidade do interessado), filtrados e tratados de forma a obter conhecimento útil. A expectativa é que estas informações sejam disponibilizadas aos setores que necessitam implementar melhorias nos ramos específicos relacionados. Os dados obtidos por meio de ‘*webscraping’* ou raspagem são de domínio público (dependendo tal disponibilidade de seu acesso, de forma que dados históricos não podem ser acessados pelo simples fato de não terem sido acessados e registrados), entretanto, a forma como são disponibilizados originalmente não é a ideal para a análise e ciência de dados. Este trabalho implementou um protótipo de sistema para coleta de dados com objetivos de demonstrar a plausibilidade da aplicação destas coletas e posterior integração à bases de dados, possibilitando o levantamento de informações e elaboração de estatísticas acerca de informações relevantes sobre os custos e demais impactos decorrentes de eventos diversos nas vias da cidade.

Foram solicitados dados desagregados através da LAI (Lei de Acesso à Informação - N° do processo: 140.00106839/2025-21) ao DETRAN-SP (Departamento Estadual de Trânsito SP) sobre os acidentes ocorridos nos últimos 10 anos na cidade de São Paulo (acessíveis de forma agregada através do sistema INFOSIGA [9]). Os dados obtidos contêm informações de:

Sinistros fatais 2014 - 2024

Sinistros não fatais 2019 - 2024

Óbitos 2014 – 2024

Os dados contêm ainda, informações de latitude/longitude e diversas outras informações (dados desagregados).

1.3. Integração dos sistemas de consulta a dados cadastrais de imóveis

No Arquivo Público Municipal (ARQUIP), um dos processos que é bastante recorrente é a pesquisa de informações cadastrais de imóveis do município. Essa atividade exige a consulta e o cruzamento de até sete bases de dados distintas — como SEI, SIMPROC, Geosampa, entre outras — o que demanda não apenas esforço cognitivo considerável, mas também um tempo significativo para seu adequado tratamento. Como maneira de simplificar tal procedimento, foram integradas duas bases de dados locais (implementadas em MSAccess), antes acessíveis independentemente, de forma a permitir consultas por assunto, alvará, despacho, interessado e logradouro em um mesmo aplicativo.

1.4. Desenvolvimento de protótipo de automatização para classificação de e-mails de solicitações enviados ao e-SIC (Sistema Eletrônico de Informação ao Cidadão)

A PMSP recebe solicitações de informações de serviços públicos via e-SIC, os quais referem-se a um dos 740 serviços prestados pelos 70 órgãos municipais cadastrados A tarefa de classificar tais mensagens para posterior tratamento demanda grande quantidade de trabalho e tempo, e crê-se que possa ser feita com o auxílio de Inteligência Artificial. Visou-se, neste trabalho, obter um protótipo funcional de um software que pré-classifique tais documentos e indique a qual tema, assunto e serviço refere-se cada mensagem. Os dados de tais solicitações podem ser acessados na Plataforma de dados abertos da Prefeitura do Município de São Paulo [10].

2. Metodologia

Apesar de não ter sido aplicada formalmente uma filosofia de desenvolvimento Ágil (ou framework como SCRUM – utilizado no gerenciamento do desenvolvimento de projetos complexos que divide o trabalho em ciclos curtos), vários conceitos destes foram utilizados nos desenvolvimentos apresentados neste capítulo, tais como eventos: reuniões curtas e periódicas, desenvolvimento mais focado no software que documentações etc., visando a obtenção de resultados mais rapidamente.

Os desenvolvimentos iniciais utilizaram o Google Colab com a ferramenta Python e SQLite3 visando estabelecer a exequibilidade dos projetos de análise exploratória de dados.

Após os testes iniciais, visando-se estabilidade, acessibilidade e maior segurança dos dados, realizou-se a implementação de sistema protótipo localmente, utilizando-se para tal do ambiente MS-VSCode com a ferramenta Python e SGBD Postgresql.

Os dados obtidos foram carregados em uma Base de Dados a fim de permitir a realização de consultas para levantamento de estatísticas, hipóteses e estimativas.

2.1. Protótipo de Automatizador de respostas a emails e chatbot

Para realizar a classificação dos e-mails de solicitação de informações sobre os cursos oferecidos pela EMASP, foi desenvolvido um protótipo de sistema de automatização de respostas sobre questões comumente feitas sobre os cursos da Escola Municipal de Administração Pública de São Paulo – EMASP. A aplicação foi modelada utilizando-se o PowerAutomate através da aplicação de lógica simples aos textos em questão. Para seu desenvolvimento, inicialmente foram classificadas as perguntas mais comumente realizadas pelos interessados. Uma vez estabelecida a distinção entre elas, escolheu-se as palavras-chave relacionadas, segmentando os testes para as categorias relacionadas (um exemplo destas categorias é apresentado na Fig. 1).

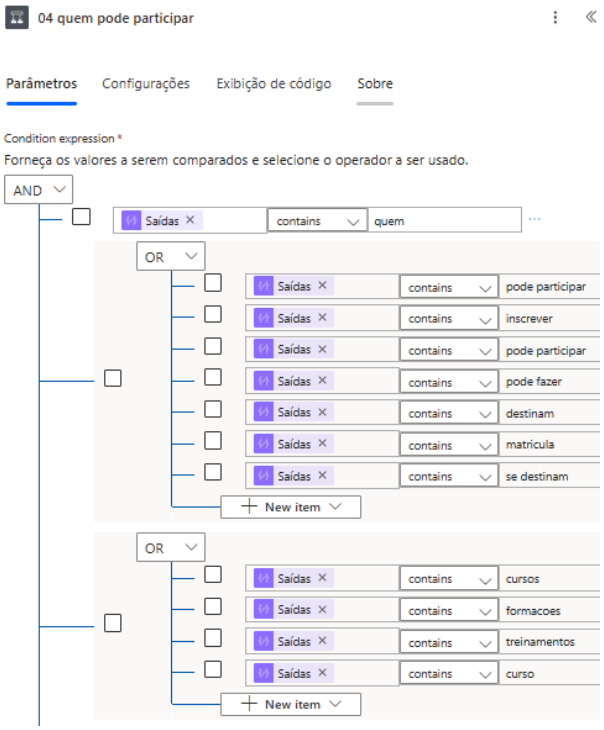

Fig. 1. Exemplo de um teste para a classificação de uma categoria

A seguir foi definido o algoritmo a ser implementado e funções requeridas (identificação dos e-mails que necessitam classificação, exclusão de eventuais domínios e tarefas de controle e registro). De maneira similar, o sistema para respostas no chat do TEAMS utiliza o mesmo algoritmo para classificar e responder aos usuários.

2.2. Levantamento exploratório de dados

O levantamento exploratório de dados visa identificar os principais dados a serem obtidos, as fontes de tais dados, formas de sua obtenção, seu armazenamento e disponibilização para implementações de inteligência de dados e futuras melhorias no suporte a serviços. Dentre o universo de dados disponibilizados pela PMSP e pelo Governo do Estado de São Paulo de interesse nesta pesquisa, foram escolhidos os assuntos relacionados à: chuvas, alagamentos, quedas de galhos e árvores e sinistros ocorridos em vias públicas.

Inicialmente, foram realizados testes exploratórios de coletas de dados através do *Google Colab*, com a ferramenta Python - (Apêndice I) e o SGBD (Sistema Gerenciador de Bancos de Dados) sqlite3 acessível por meio da respectiva biblioteca para python e interface web *sqliteviewer*, de forma totalmente online.

A seguir, estes protótipos foram desenvolvidos localmente na linguagem Python, de maneira a manter sua disponibilidade independentemente do estado das conexões de internet, sendo os dados armazenados em um SGBD Postgresql.

2.3. Integração dos sistemas de consulta a dados cadastrais de imóveis

Duas das bases de dados disponíveis anteriormente (conhecidas como “LINDÃO” e “BDI”) que são utilizadas no ARQUIP permitem a busca por informações cadastrais de imóveis e alvarás separadamente, sendo necessário ao usuário alternar entre as mesmas para em um momento buscar informações sobre alvarás e em um outro momento dados de registro no outro sistema. Visou-se o desenvolvimento de uma aplicação que em uma mesma interface permitisse a pesquisa em ambas as bases de dados simultaneamente e apresentasse os resultados.

2.4. Desenvolvimento de protótipo de automatização para classificação de e-mails de solicitações enviados ao e-SIC

O protótipo de sistema de automatização para classificação de e-mails subdivide-se em duas partes: o recebimento, seleção, armazenagem e disponibilização das mensagens (e-mails – realizado pelo PowerAutomate) e o tratamento e classificação das mesmas (realizado em Python).

De maneira similar à classificação de e-mail enviados à EMASP, os e-mails enviados à e-SIC devem ser identificados no PowerAutomate em função do assunto, no campo ASSUNTO do mesmo (foram criadas bases de te). A seguir, são realizadas tarefas iniciais de obtenção de seu conteúdo e armazenagem em uma planilha do Excel (fig.2), a qual é lida por um programa desenvolvido em Python.

Visando-se aplicar o Processamento de Linguagem Natural (NLP) ao processamento do conteúdo daqueles e-mails, foram testadas diversas bibliotecas disponíveis no Python (NTLK, spaCy, Scikit-learn) e também pesquisados e testados protótipos de algoritmos de classificação (baseados em lógica Fuzzy, Bag Of Words, algoritmo de similaridade de Jaccard, funções de vetorização, lematização dentre outros) que realizam grande número de tarefas, tais como extração de características, classificação de palavras, vetorização, treinamento e predição. Embora sejam bastante eficientes em alguns casos, muitos desses não apresentaram resultados satisfatórios, sendo descartados.

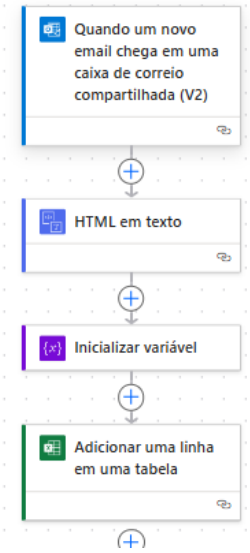

Fig. 2: Exemplo de automatização de dados com o PowerAutomate.

Após este tratamento inicial, o software desenvolvido em Python, aplicando técnicas e algoritmos de avaliação de similaridade textual avalia cada linha da planilha, escolhe o resultado mais similar e registra a sugestão da classificação mais plausível para cada registro. O PLN atualmente em testes apresenta uma acurácia bastante baixa (da ordem de 10 a 20%) necessitando de melhorias.

3. Literatura Relacionada

A literatura utilizada para estudos iniciais e diagnósticos fundamentou-se na escolha dos assuntos mais relevantes à aplicação de modelagens para avaliação do problema e relevância e dimensões das cidades citadas e estudos disponíveis. Os métodos de diagnóstico concentram-se nos eventos mais significativos que impactam no referido tema. Também se incluem na bibliografia literaturas acerca de ciência de dados, práticas de processamento de dados e dos conceitos de disponibilidade e confiabilidade de sistemas, bem como estatística.

4. Proposta de melhoria

O projeto tem como objetivos:

1- Reduzir tarefas repetitivas e ocasionadoras de grande dispêndio de tempo por meio da implementação de sistemas de classificação de mensagens e textos, utilizando-se de ferramentas de automatização e processamento baseado em NLP.

2- Desenvolver protótipos de aplicações para realizar obtenção automatizada de dados públicos (raspagem), implementar Bases de Dados que possibilite a geração de consultas para finalidades estatísticas diversas, e a partir destas:
    a. Identificar onde os alagamentos ocorrem, sua intensidade, e identificar qual a quantidade de chuva e os impactos gerados por esses alagamentos;
    b. Identificar localidades para as quais há probabilidade de ocorrências de quedas de árvores, bem como seus impactos
    c. Gerar mapas de calor de ocorrências

4.1 Protótipo de Automatizador de respostas a e-mails e chatbot sobre questões comumente feitas sobre os cursos da EMASP

As perguntas normalmente feitas via e-mail são divididas em 22 assuntos, que incluem solicitações sobre temas dos cursos, oferecimentos de alimentação e estacionamento, carga horária, certificados dentre outros. Ao realizar uma análise dos conteúdos daquelas mensagens, pode-se classificá-las com nível razoável de acurácia nestas categorias, desde que a solicitação não seja muito extensa por meio de testes de identificação de palavras-chave. Tendo sido gerado por ferramenta automatizada, seu código-fonte é bastante extenso (49.000 caracteres), portanto o mesmo não foi incluído neste trabalho.

Conforme pode ser visto no diagrama de transição de estados (fig. 4) deste protótipo, uma vez que um e-mail é recebido com determinado assunto (“Dúvidas”) a uma específica caixa de e-mail (“linguagemsimples@prefeitura.sp.gov.br”), é verificado se o mesmo teve como origem determinado domínio (caso positivo, a mensagem é excluída), caso negativo, são buscadas palavras chave que classificam a mensagem em um dos 22 assuntos, uma resposta automática é enviada ao remetente e, por fim, a mensagem e o retorno são registrados em uma planilha do Excel.

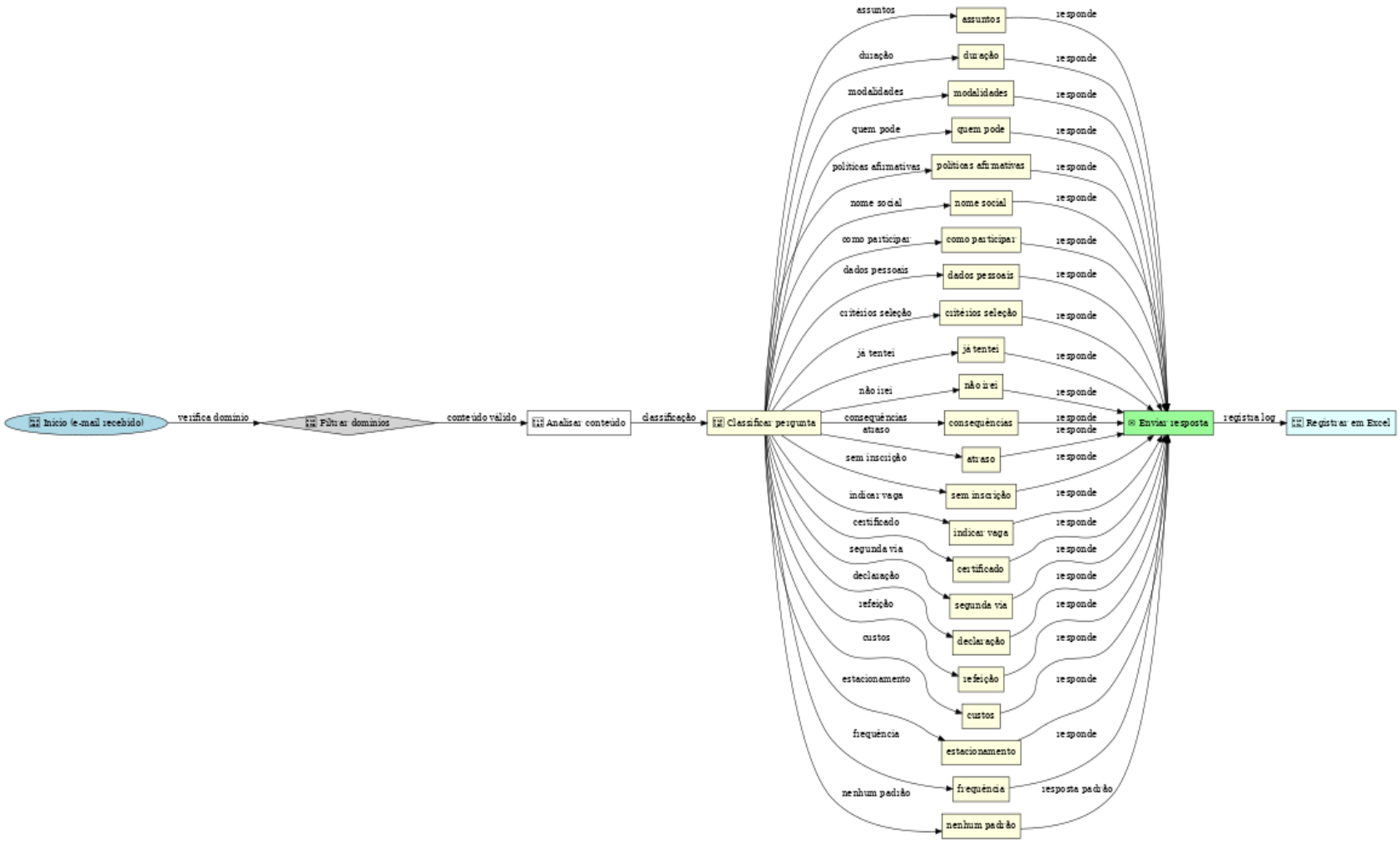
Início (e-mail recebido)
verifica domínio
Filtrar domínios
conteúdo válido
Analisar conteúdo
classificação
Classificar pergunta
assuntos
duração
modalidades
quem pode
políticas afirmativas
nome social
como participar
dados pessoais
critérios seleção
já tentei
não irei
consequências
atraso
sem inscrição
indicar vaga
certificado
segunda via
declaração
refeição
custos
estacionamento
frequência
nenhum padrão
responde
resposta padrão
Enviar resposta
registra log
Registrar em Excel

Fig. 4 – Classificador de e-mails enviados à EMASP pelo PowerAutomate

Utilizando-se basicamente a mesma lógica, implementou-se protótipo de sistema de respostas automáticas via TEAMS (fig. 5).

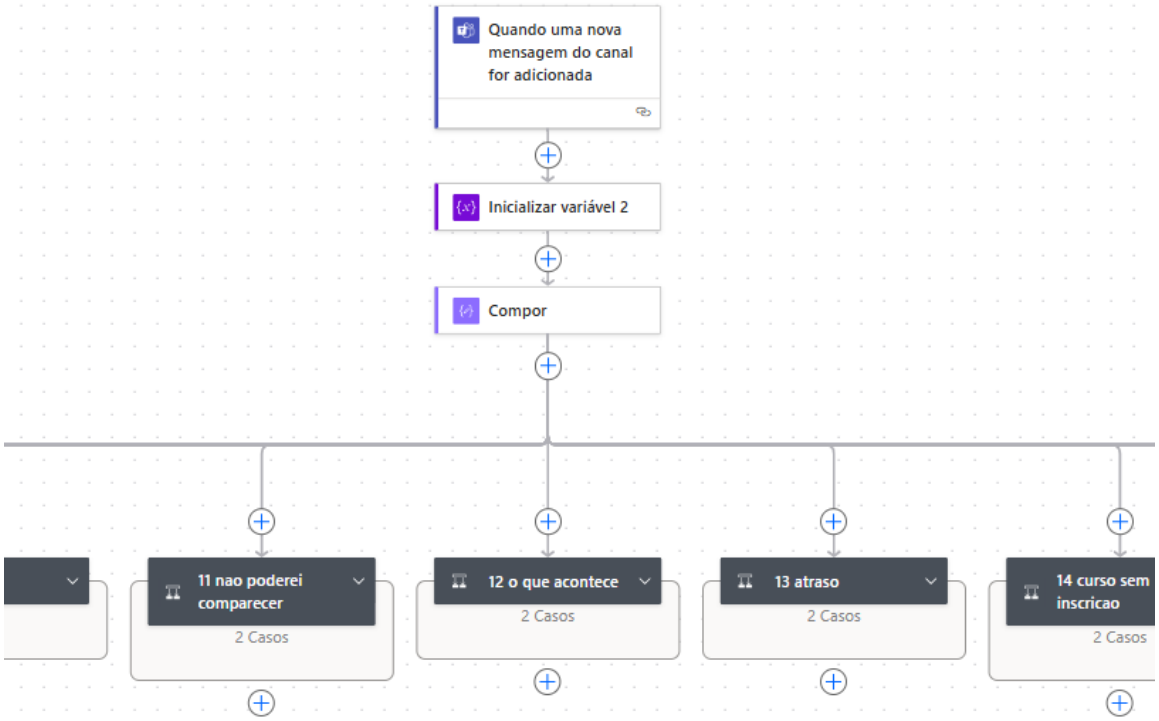

Fig.5 - Sistema de respostas automáticas pelo PowerAutomate via TEAMS

4.2. Levantamento exploratório de dados para Identificação e mapeamento de pontos críticos/eventos relacionados ao trânsito no Município de São Paulo

Definidos os assuntos para os quais se pretende gerar repositórios de dados, etapas que podem ser mais ou menos extensas referem-se a sua aquisição, armazenagem e possível tratamento. Foram identificadas como fontes de dados: o CGE, SP156, DETRAN-SP [9], IBGE e INMET. Após uma avaliação inicial, constatou-se que alguns destes não apresentam as características desejadas sendo desconsiderados nas pesquisas estatísticas, mas mantidos na base de dados (conforme dicionário de dados apresentado no Apêndice 3).

4.2.1. Chuvas e alagamentos

Por meio de operações de raspagem (*webscraping*) do sítio do CGE, podem ser obtidos, em tempo real, dados sobre pluviosidade nas 30 estações de medição

distribuídas na capital paulista. Dados históricos sobre alagamentos também estão disponíveis no sítio do CGE, este sítio contém grande quantidade de informações históricas, razão pela qual foi desenvolvida outra ferramenta de *webscraping* para coleta e armazenagem destes dados.

A coleta destes dados tem como objetivo a elaboração de estatísticas, avaliação de correlações e possíveis testes de hipóteses relacionados às ocorrências de alagamentos comparando-se os dados de velocidade de ventos, pluviosidade, temperatura etc.

Foram desenvolvidos protótipos em linguagem Python para as tarefas de coleta (coleta de dados em tempo real e coleta de registros históricos de alagamentos), sendo os dados armazenados em uma Base de Dados Postgresql. Também foram testadas, em caráter experimental, interfaces (fig. 6) que tem a finalidade de facilitar tal tarefa e permitir sua utilização por usuários leigos. Os mesmos foram ainda convertidos em linguagem C e compilados (utilizando a biblioteca *nuitka*) para permitir sua utilização em qualquer dispositivo, também de forma experimental.

Em relação à correção de tais dados, deve-se considerar: muitas das atividades de coleta, sobretudo aquelas em tempo real, geram dados com falhas ou parcialmente ilegíveis (em função de falhas de conexão, indisponibilidade de páginas, ou mesmo alterações em seus formatos), os quais podem necessitar de tratamento e/ou descarte *a posteriori*; sendo assim quase certa a existência de dados incorretos/incompletos/inválidos na Base de Dados em função da metodologia de coleta, tratamento, formatação, etc. ou provenientes de sistemas legados. Desta forma, torna-se importante o registro da origem de tais informações e condições em que foram adquiridos.

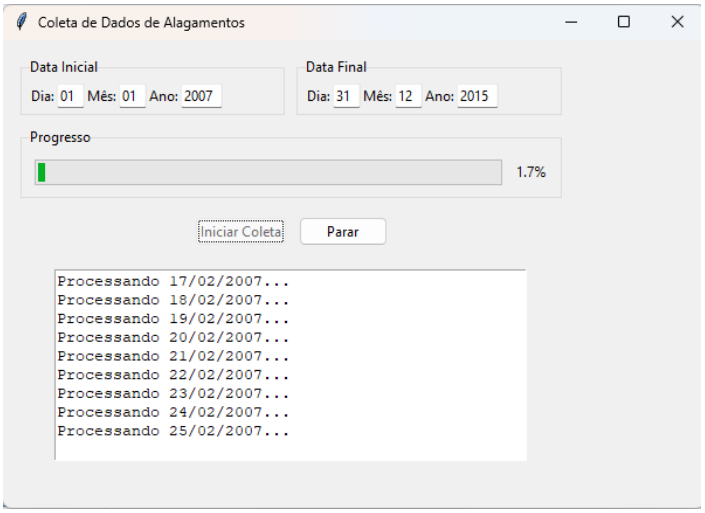

Fig. 6 - Protótipo funcional de interface para coleta dos dados históricos de alagamentos.

Em posse de tais dados, pode-se gerar estatísticas diversas, tais como número de alagamentos registrados por ano (fig. 7), em que é possível notar o efeito da variação da pluviosidade no ano de 2018.

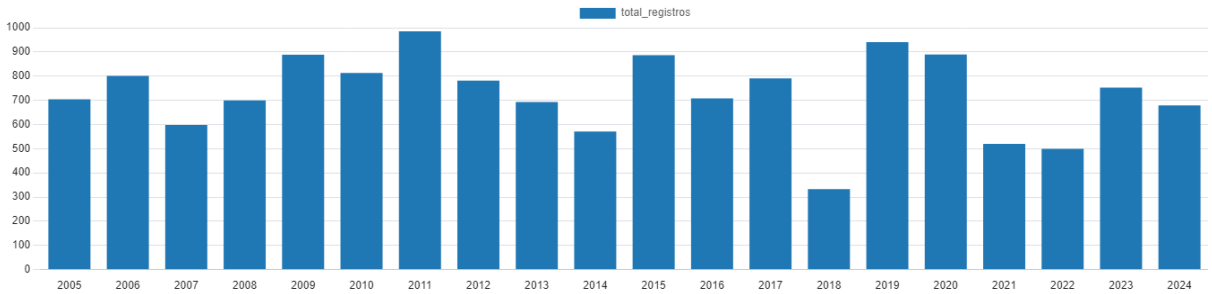

Fig. 7- Números absolutos de ocorrências de alagamentos registrados no período 2005 a 2024.

Ou ainda, média de horas de interdição de vias por período, conforme fig. 8. A média móvel indica uma significativa redução no tempo de interdição média das vias no período (em torno de 2,5h para 1,75h ou seja, 30%).

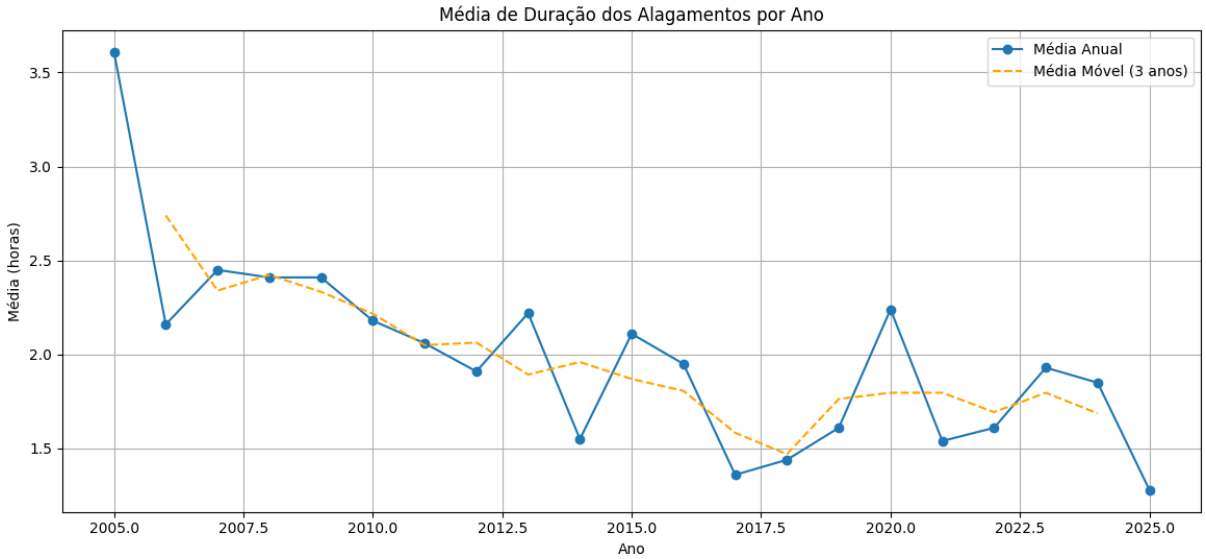

Fig. 8- Média de horas por interdição de via no período 2005 a 2025

4.2.2. Quedas de árvores e galhos

Dentre os dados disponibilizados pelo SP156 e de interesse, encontram-se informações sobre quedas de árvores e galhos, solicitações de limpeza de bueiros, tapa-buracos, dentre outros, os quais podem ser baixados por meio de API ou portal dados abertos.

Inicialmente, visou-se o levantamento de informações relacionadas aos alagamentos, como ocorrência de quedas de galhos e árvores. Aqueles ocasionam impacto direto no trânsito, gerando congestionamentos cujos impactos financeiros podem ser mensurados por meio de estimativas. No caso das quedas de galhos e árvores, apesar de seus efeitos poderem ser mais indiretos, ou mesmo não causarem efeitos no trânsito, existe a probabilidade de geração de danos e prejuízos diversos, tais como interdições, danos físicos etc. As planilhas do MS-Excel relativas àquelas informações foram então importadas à Base de Dados por meio de script desenvolvido em Python.

Uma vez que os dados obtidos foram inseridos no Banco de Dados, é importante manter registro sobre informações específicas sobre seu período de obtenção, restrições relativas à anonimização etc. Por exemplo os dados relativos a quedas de árvores contêm informações anuais sobre quedas de árvores e galhos, de um período de um ano, iniciado em 01 de outubro e encerrado em 30 de setembro. Dados sobre alagamentos apresentam o nome da via e alguma referência, mas não contêm

informação sobre latitude e longitude etc. A partir destes dados, e considerando-se as mencionadas restrições, pode-se realizar estimativas anuais sobre ocorrências diversas, neste caso, uma consulta feita em SQL (*Structured Query Language*) pode prover informações como: sob quais subprefeituras ocorreram o maior número de quedas de árvores, como se pode verificar na tabela 1., a seguir:

ANO	Subprefeitura	Quedas de galhos ou árvores
2023	SANTO AMARO	851
2024	VILA MARIANA	838
2023	VILA MARIANA	738
2023	BUTANTA	717
2023	CAMPO LIMPO	654
2024	PINHEIROS	588
2024	SE	554
2023	M BOI MIRIM	403
2023	PINHEIROS	399
2023	IPIRANGA	395
2024	BUTANTA	378
2024	SANTO AMARO	368
2024	MOOCA	329
2023	LAPA	322
2024	LAPA	313
2023	MOOCA	305
2024	IPIRANGA	300

Tab. 1 – Quantidades de ocorrências de quedas de galhos ou árvores relacionados às subprefeituras, por ano.

4.2.3. Sinistros

Os registros das ocorrências de sinistros permitem a elaboração de estimativas de seus custos em função de despesas hospitalares (apresentado no item 4.5).

Conforme apresentado no item 1.2, a existência de grandes e diversos volumes de dados públicos ou acessíveis por meio de pedidos via LAI, possibilitou a implementação de uma Base de Dados, contendo informações sobre sinistros de trânsito em geral (óbitos, acidentes, atropelamentos dentre outros).

4.3. Junção das bases de dados “Lindão” e “BDI”

A integração foi desenvolvida em html – *HyperText Markup Language* (formato hta – html *application*) com VBScript. Inicialmente foi necessário integrar os conteúdos das bases de dados em um arquivo único (todos em MSAccess), as linguagens foram

escolhidas em função de sua possibilidade de utilização nos PCs da prefeitura sem a necessidade de instalação de outros aplicativos ou alterar permissões de usuários. O sistema encontra-se em uso por funcionários do ARQUIP.

4.4. Desenvolvimento de sistema de classificação de e-mails enviados ao e-SIC

Com o auxílio das ferramentas PowerAutomate e Python, foi desenvolvido um protótótipo de sistema de classificação de e-mails baseado na análise de seus conteúdos. Os e-mails referem-se aos serviços providos pela PMSP (cerca de 740 serviços) e são classificados em função destes.

Foi gerada uma listagem dos serviços (dividida em tema, assunto e serviço), atribuído um identificador único a cada item, adicionados os acrônimos referentes aos serviços (quando aplicável). Posteriormente a mesma listagem foi carregada à base de dados.

O PowerAutomate registra em uma planilha no Sharepoint os e-mails enviados a específica caixa de correio de testes (aquelas que tem como assunto determinada palavra).

O sistema desenvolvido em Python (cujo Diagrama de Fluxo de Dados é apresentado na figura 9), através de bibliotecas de NLP (como spaCy) realiza varredura nos textos e os classifica (atribui notas) comparando com uma base de dados que contém a listagem dos serviços, salvando a seguir na planilha utilizada pelo Sharepoint.

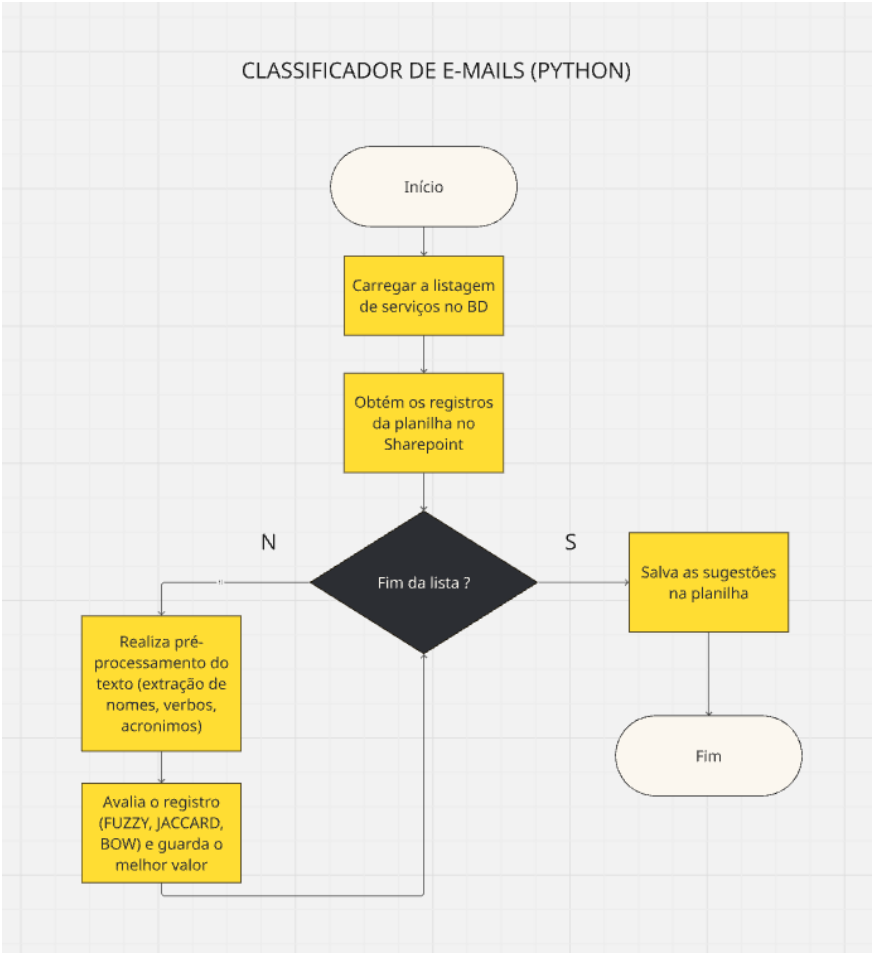

Fig. 9 – Diagrama de Fluxo de Dados do Sistema de Classificação (Python).

4.5. Modelagens, Estatísticas e Estimativas

Um exemplo de estatísticas que podem ser geradas a partir dos registros é apresentada na tabela 2, nesta podem ser observados: as 20 vias com maior número de ocorrências não fatais nos últimos 5 anos e as respectiva quantidades.

Este exemplo demonstra detalhes sobre a variabilidade dos dados (por exemplo, as duas localidades com os maiores números absolutos no ano de 2024 não constam nas 20 vias com maiores quantidades de ocorrências nos 4 anos anteriores.

SINISTROS NÃO FATAIS									
2024		2023		2022		2021		2020	
LOGRADOURO	OCORRENCIAS	LOGRADOURO	OCORRENCIAS	LOGRADOURO	OCORRENCIAS	LOGRADOURO	OCORRENCIAS	LOGRADOURO	OCORRENCIAS
MARGINAL PINHEIRO	508	ESTRADA DE ITAPECEF	444	AVENIDA SAPOPEMB	405	AVENIDA SAPOPEMB	445	ESTRADA DE ITAPECI	369
MARGINAL TIETÊ	503	AVENIDA SAPOPEMBA	340	AVENIDA ARICANDUV	373	ESTRADA DE ITAPECE	397	AVENIDA SAPOPEMI	343
AVENIDA SENADOR T	415	AVENIDA ARICANDUV	323	ESTRADA DE ITAPECE	372	AVENIDA ARICANDU	380	AVENIDA ARICANDU	325
ESTRADA DE ITAPECE	404	ESTRADA DO M'BOI M	307	AVENIDA SENADOR T	352	AVENIDA SENADOR T	378	AVENIDA SENADOR	321
AVENIDA ARICANDUV	388	AVENIDA INTERLAGOS	262	AVENIDA DONA BELN	282	VIADUTO RODOANEL	343	AVENIDA DONA BEL	267
ESTRADA DO M'BOI M	374	AVENIDA DO ESTADO	259	VIADUTO RODOANEL	278	AVENIDA DONA BELN	319	ESTRADA DO M'BOI	261
AVENIDA DAS NAÇÕE	361	AVENIDA SENADOR TE	252	AVENIDA DAS NACOE	273	ESTRADA DO M'BOI I	302	AVENIDA MARECHA	250
AVENIDA SAPOPEMB	355	AVENIDA DONA BELM	250	AVENIDA RAGUEB CH	268	AVENIDA MARECHAL	268	AVENIDA DO ESTAD	238
RODOANEL MÁRIO C	347	ESTRADA DO CAMPO	249	AVENIDA DOS BANDE	254	AVENIDA DAS NACOE	257	AVENIDA RAGUEB C	229
AVENIDA JACUPÊSSE	288	AVENIDA MARECHAL	248	AVENIDA DO ESTADO	243	AVENIDA DO ESTADO	256	AVENIDA WASHINGT	216
AVENIDA WASHINGT	280	AVENIDA RAGUEB CH	244	AVENIDA WASHINGT	242	AVENIDA RAGUEB CH	250	AVENIDA INTERLAG	211
AVENIDA DO ESTADO	262	AVENIDA DAS NACOES	212	AVENIDA JACÚ PESSE	230	AVENIDA WASHINGT	243	AVENIDA GUARAPIR	205
AVENIDA 23 DE MAIC	262	AVENIDA DOS BANDE	204	AVENIDA MARECHAL	218	ACESSO RODOVIA RA	225	AVENIDA DAS NACO	203
AVENIDA DONA BELN	256	AVENIDA PROFESSOR	194	ESTRADA DO CAMPO	213	AVENIDA DOS BANDE	216	AVENIDA JACÚ PESS	197
AVENIDA ATLÂNTICA	253	BR 116	194	AVENIDA INTERLAGO	208	AVENIDA JACÚ PESSE	211	AVENIDA PROFESSO	190
AVENIDA RADIAL LES	230	VIADUTO RODOANEL	186	AVENIDA VINTE E TRE	207	ESTRADA DO CAMPO	211	AVENIDA PROFESSO	177
AVENIDA MARECHAL	218	AVENIDA INAJAR DE S	183	ACESSO ESTRADA DO	204	AVENIDA INTERLAGO	206	ESTRADA DO CAMP	171
AVENIDA SÃO MIGUE	215	AVENIDA GUARAPIRA	180	AVENIDA ATLANTICA	189	ACESSO AVENIDA MA	199	AVENIDA RAIMUND	159
SP270	213	ESTRADA DO ALVARE	174	AVENIDA PROFESSOR	187	AVENIDA VINTE E TRI	187	AVENIDA VINTE E TR	158
AVENIDA PROFESSOR	211	AVENIDA GIOVANNI G	168	AVENIDA PROFESSOR	179	AVENIDA PROFESSOR	185	AVENIDA DOS BAND	156

Tabela 2- Ocorrências de sinistros por logradouros (período 2020 a 2025).

Com a utilização desta base de dados também é possível levantar informações acerca da ocorrência de sinistros fatais/dia da semana, horário, gênero etc. (possível interesse para campanhas educativas, dentre outros - figs.10 a 13) e sua evolução ano a ano.

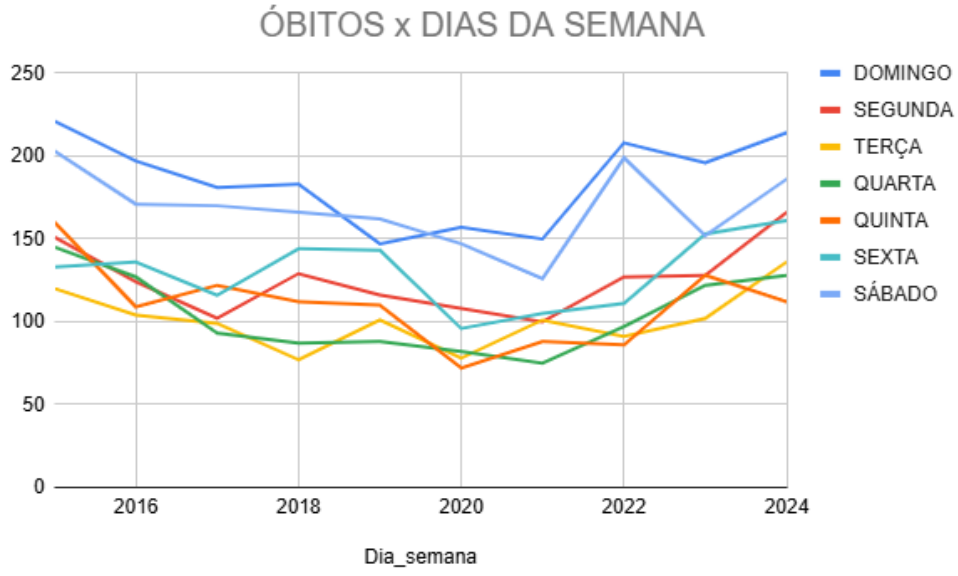

Fig. 10 – Exemplo de estatística: óbitos por dia da semana

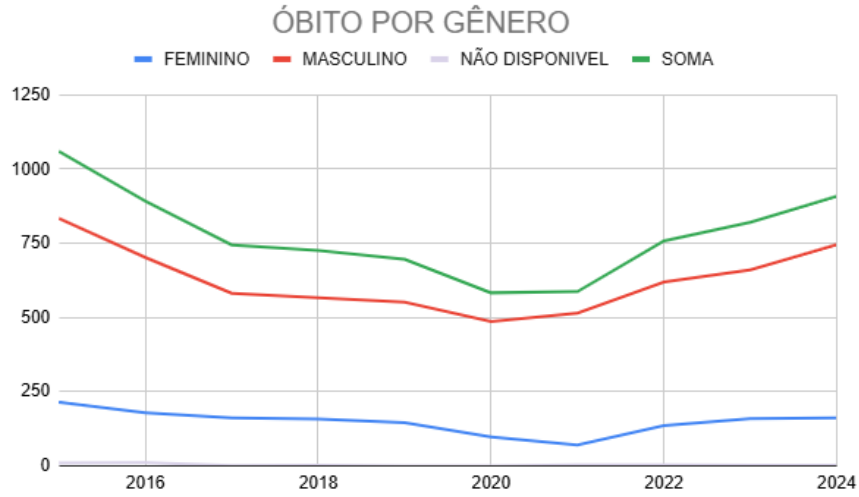

Fig. 11 – Óbitos por gênero (2015 a 2024)

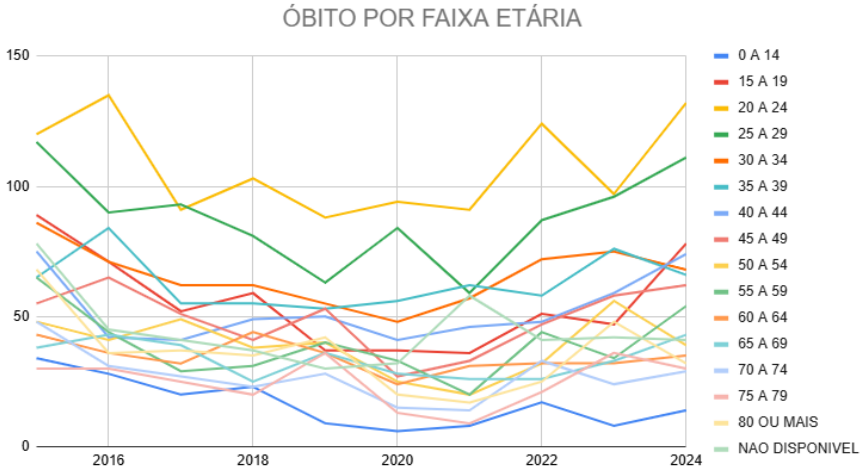

Fig. 12 – Óbitos por faixa etária (2015 a 2024)

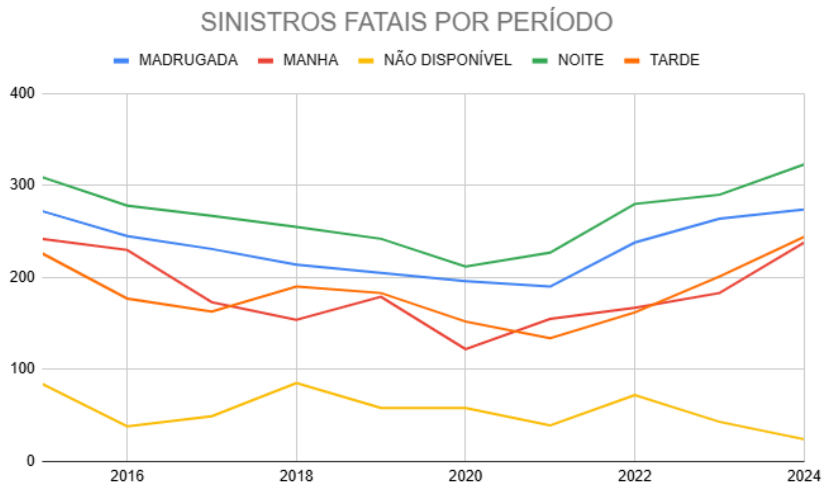

Fig. 13 – Sinistros fatais por período do dia (2015 a 2024)

Exemplo de localidade com alagamento observada no *google maps*: (figura 1. Apêndice 2). Estimativa do congestionamento nos arredores para esta ocorrência: (figuras 2,3 e 4 – Apêndice 2). Neste exemplo, observando-se as vias congestionadas em um raio de cerca de 200m da via citada como intransitável em função do alagamento, mediu-se, via *Google Maps* [12], 646m + 108m + 257m (1,011km).

Os dados consolidados (tab. 3) apresentam a somatória de pontos de alagamento e horas de interdição (total ou parcial) que os mesmos ocasionam.

Ano	∑ pontos	∑ horas / ano	média diária pontos
2005	704	2538:53'	3,61
2006	801	1732:11'	2,16
2007	598	1465:58'	2,45
2008	699	1683:53'	2,41
2009	888	2138:55'	2,41
2010	813	1769:25'	2,18
2011	985	2031:51'	2,06
2012	781	1494:17'	1,91
2013	693	1535:26'	2,22
2014	571	886:27'	1,55
2015	886	1,868:32'	2,11
2016	708	1,378:35':	1,95
2017	791	1,074:42'	1,36
2018	333	478:35'	1,44
2019	940	1,509:52'	1,61
2020	889	1,994:25'	2,24
2021	520	800:35'	1,54
2022	499	801:58’	1,61
2023	752	1,450:54'	1,93
2024	679	1,259:14'	1,85
Média	**726.5**	**1,494:43'**	**2,03**

Tab. 3 – Dados consolidados de pontos de alagamentos e respectivas horas de interdição, ano a ano.

Objetivando-se estimar os custos relacionados a estes eventos, considerou-se os seguintes valores:

- Média salarial de SP = R$ 4.263 por mês, 213 por dia, 25 reais por hora.

- Tamanho médio dos veículos = 5m

- Quantidades médias de faixas das vias = 2,5

- Número médio de pessoas por veículo = 2

- Consumo médio de um veículo à velocidade padrão da via = 0,0026 litros/min

- Emissão de CO_2 por litro de gasolina = 2,28 kg

Desta forma temos:

Média pontos	Média horas / ano	Média diária pontos	Custo diário (R$)	Custo Anual médio (R$)
726,5	1494:43’	2,03	205.233,00	74.910.045,00

Tab. 4 - Médias de pontos de alagamentos e estimativa de custos em função do desperdício de homens-hora.

Para a emissão de dióxido de carbono, tem-se (tab. 5):

Média minutos/dia	Média diária pontos	litros/dia	kg CO_2/dia	kg CO_2/ano
245	2,03	653,6	1.490,36	543982

Tab. 5 – Estimativa da emissão média de dióxido de carbono em função dos congestionamentos ocasionados por alagamentos (veículos supostamente sem sistema *auto start-stop*).

Ou seja, conforme apresentado nas tabelas 4 e 5, ocorre diariamente um prejuízo médio da ordem de R$ 200 mil ou R$ 75 milhões por ano em função da quantidade de homens-hora não produtivos e cerca de 1,5T CO_2/dia ou 544T CO_2/ano em função daquelas ocorrências. A estimativa sobre o consumo de combustível foi feita considerando-se como base a velocidade média de 90km/h de um veículo que atinge a média de 14km/l (0,16 l/h) e pode ser considerada conservadora (outras estimativas avaliam este valor entre 0,6 e 5 litros por hora (o que é mais provável em função da idade média da frota: 17,4 anos no município), o que ocasionaria um valor final linearmente proporcional a este incremento).

Levando-se em conta a somatória de intervalos de tempo entre os óbitos ocorridos em decorrência de sinistros de trânsito, é possível realizar uma estimativa dos custos totais de internação em UTIs (considerando-se como custo de uma diária de UTI de R$ 2.500,00), o custo anual (estimado em R$ 40,8 Milhões) é similar ao do orçamento de algumas secretarias (ex. Secretaria Municipal de Relações Internacionais da PMSP, em 2024: R$ 40,5 Milhões; Secretaria Municipal da Pessoa com Deficiência,

também em 2024: R$ 32,5 Milhões). Segundo estimativa [13] feita pelo Conselho Regional de Medicina do Rio Grande do Sul (CREMERS), a despesa nacional com acidentes de trânsito chega a R$ 3 Bilhões em 10 anos, por esta projeção, 1/6 desta despesa ocorre no município de São Paulo.

Uma forma de apresentação de dados visualmente atraente e autoexplicativo são os *heat-maps*, ou mapas de calor, geráveis em Python (biblioteca *folium*). Um exemplo destes para o risco de quedas de árvores é apresentado na figura 12 (os dados dos relatos de risco de queda de galhos e árvores mantêm as informações de latitude e longitude, diferentemente da dos registros das ocorrências de quedas disponibilizados publicamente, o que possibilitou tal mapeamento);

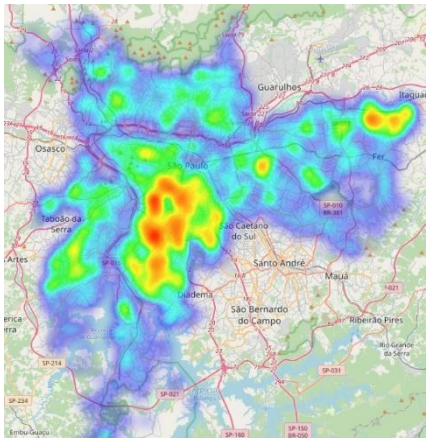

Fig. 12 – *Heat map* do risco de ocorrência de galhos e árvores no município de São Paulo, conforme relatos dos munícipes registrados pelo SP156.

5. Conclusões

Utilizando-se os dados obtidos mencionados, foram gerados mapas de calor, estatísticas gerais e estimativas, conforme apresentado no item 4.5. Este projeto visou apresentar, através do tratamento e da aplicação de dados reais, modelagens que possibilitem estimar custos e o estudo de hipóteses e melhorias ao setor.

Dentre os impactos esperados, citam-se: a possibilidade de utilização destas informações por outras Secretarias Municipais e público em geral, redução de gargalos e aumento da eficácia no manejo de árvores em localidades críticas, aumento da previsibilidade e melhoria das condições de trânsito na cidade, dentre outros.

Embora sejam sistemas prototípicos, a flexibilidade e consistência no uso das ferramentas de desenvolvimento e das bases de dados demonstram que esses sistemas podem ser continuamente aprimorados e incrementados, sugerindo que essa abordagem facilitaria a geração de insights e pesquisas.

Naturalmente, caso seja realizada implementação destes sistemas, será necessário avaliar e implementar controles de acesso (uma vez que poderá conter dados sensíveis no caso de não estarem previamente anonimizados), e demais considerações de segurança (sugere-se como validação o padrão OWASP – *Open Worldwide Application Security Project*).

6. Bibliografia

1- Matheus R., Janssen M., Maheshwari D., Data **science empowering the public: Data-driven dashboards for transparent and accountable decision-making in smart cities.** 2020. Disponível em: https://www.sciencedirect.com/science/article/pii/S0740624X18300303

2- Qin, Y.; Luo, Q., Wang, H., - **Markov Chain-based capacity modeling for mixed traffic flow with bi-class connected vehicle platoons on minor road at priority intersections**. 2025. Disponível em https://www.sciencedirect.com/science/article/abs/pii/S0378437124008112

3- Besenczi R. , et al - **Large-scale simulation of traffic flow using Markov model.** 2021. Disponível em: https://pmc.ncbi.nlm.nih.gov/articles/PMC7872230/#pone.0246062.ref019

4- Janssen, M., Matheus, R., Zuiderwijk, A., **Big and Open Linked Data (BOLD) to Create Smart Cities and Citizens: Insights from Smart Energy and Mobility Cases**, 2016. Disponível em: https://hal.science/hal-01412238/file/346786_1_En_6_Chapter.pdf

5- Janssen, M., Hartog, M., et al, **Will Algorithms Blind People? The Effect of Explainable AI and Decision-Makers' Experience on AI-supported Decision-Making in Government**, 2020. Disponível em: https://statics.teams.cdn.office.net/evergreen-assets/safelinks/1/atp-safelinks.html

6- Seenivasan D., **ETL (Extract, Transform, Load) Best Practices,** Mphasis, Texas, USA, 2023. Disponível em: https://www.researchgate.net/publication/368300449_ETL_Extract_Transform_Load_Best_Practices

7- Pressman, Roger S. **Engenharia de Software: Uma Abordagem Profissional**. 8. ed. São Paulo: McGraw-Hill, 2016

8- Brown, T. **Change by Design: How Design Thinking Creates New Alternatives for Business and Society.** 2009

9- Infosiga: Disponível em: https://www.infosiga.sp.gov.br/?name=identificacao4&contextId=8a80809939587c0901395881fc2b0004#obitos Acesso em 02/2025.

10- Plataforma de dados abertos da Prefeitura do Município de São Paulo: http://dados.prefeitura.sp.gov.br/dataset/pedidos-de-informacao-protocolados-a-prefeitura-via-e-sic.;. Acesso em 22/04/2025.

11-CGE: https://www.cgesp.org/v3/alagamentos.jsp

12-Google Maps: https://www.google.com/maps

13-Em dez anos, acidentes de trânsito consomem quase R$ 3 bilhões do SUS. Disponível em: https://cremers.org.br/2453-2/ Acesso em 11/02/2025.

6. Apêndices

Apêndice 1.1: Códigos-fonte em Python para coleta de dados e geração de Bases de Dados (Google Colab)

```
### PASSO 1
from bs4 import BeautifulSoup
import csv, requests, datetime
from requests_html import HTMLSession
import asyncio
import time
import os
import nest_asyncio
from requests_html import AsyncHTMLSession

nest_asyncio.apply()
def numeroestacao(strURL,identif):
    text=strURL
    pos = text.find(identif)
    if pos != -1:
        return text[pos+1:]
    else:
        return text

# OBTEM A LISTA DE ESTACOES E SALVA EM ARQUIVO
def obter_estacoes():
    url = 'https://www.cgesp.org/v3/estacoes-meteorologicas.jsp'
    # Send a GET request to the URL
    response = requests.get(url)
    response.raise_for_status()  # Check for HTTP errors
    # Parse the HTML content
    soup = BeautifulSoup(response.text, 'html.parser')
    # Find the <ul> element with class 'arial-data' and id 'lista-estacoes'
    ul_element = soup.find('ul', {'class': 'arial-data', 'id': 'lista-
estacoes'})
    # Extract the data from the <ul> element
    data = []
    for li in ul_element.find_all('li'):
        a_tag = li.find('a')
        if a_tag:
            station_name = a_tag.text.strip()
            station_url =numeroestacao(a_tag['href'],"=")
            data.append((station_name, station_url))

    # Write the extracted data to a CSV file
    with open('estacoes.csv', 'w', newline='', encoding='utf-8') as file:
```

```
        writer = csv.writer(file)
        writer.writerow(['ESTACAO', 'POSTO'])
        writer.writerows(data)
    print("estacoes.csv foi atualizado")

if not os.path.exists('dados_meteorologicos.csv'):
    print("falha no arquivo")
    with open('dados_meteorologicos.txt', 'w', newline='', encoding='utf-8')
as file:
        pass
async def page_scraper(url):
    print(f"pagescraper called:{url}")
    estacao = numeroestacao(url, "=")
    session = AsyncHTMLSession()

    response = await session.get(url)  # Await the response
    try:
        response.html.arender(timeout=60, sleep=1)  # Await rendering, added
sleep
    except TimeoutError:
        print(f"Timeout for {url}. Skipping...")
        return  # Skip this URL and continue with the next

    if response.status_code == 200:
        soup = BeautifulSoup(response.html.html, 'html.parser')
        main_filename = estacao + '.html'
        with open(main_filename, 'w', encoding='utf-8') as file:
            file.write(response.html.html)
        with open(main_filename, 'r', encoding='utf-8') as file:
            html_content = file.read()
        print("arquivo saida:",main_filename)
        soup = BeautifulSoup(html_content, 'html.parser')
        # Find the table within the specified div
        try:
            table = soup.find('div', {'id': 'instrumentos'}).find('table')
            if table:
                # Extract the headers
                headers = [header.text for header in table.find_all('th')]
                    # Extract the rows
                rows = []
                for row in table.find_all('tr')[1:]:
                    cells = row.find_all('td')
                    row_data = []
                    for cell in cells:
                        inner_table = cell.find('table')
                        if inner_table:
                            inner_rows = inner_table.find_all('tr')
```

```
                            for inner_row in inner_rows:
                                inner_cells = inner_row.find_all('td')
                                for inner_cell in inner_cells:
                                    row_data.append(inner_cell.text.strip())
                        else:
                            row_data.append(cell.text.strip())
                    rows.append(row_data)
            else:
                print("tabela instrumentos não encontrada")
                return
        except Exception as e:
            print("Erro: ", e)
            return
    else:
        print(f"Pagina nao acessivel. Status code: {response.status_code}")
        return

    #extrai os dados do arquivo baixado e salva em dados_meteorologicos.csv
    datahora = str(datetime.datetime.now())
    with open('dados_meteorologicos.txt', 'a', newline='', encoding='utf-8')
as file:
        file.write("\nDados do scraping:" + "\n")
        file.write(estacao + "\n")
        file.write(datahora + "\n")
        file.write(str(rows))
    print("Dados escritos em dados_meteorologicos.txt")

if not os.path.exists('dados_meteorologicos.csv'):
    print("falha no arquivo")
    with open('dados_meteorologicos.csv', 'w', newline='', encoding='utf-8')
as file:
        pass
# funcao que lista todas as estacoes e salva em .csv
async def obtem_dados_estacoes():
    with open('estacoes.csv', 'r', newline='', encoding='utf-8') as file:
        reader = csv.reader(file)
        # Skip the header row
        next(reader, None)
        for row in reader:
            endereco = "https://www.cgesp.org/v3/estacao.jsp?POSTO=" + row[1]
            print(f"endereco:  {endereco} \n nome: {row[0]}")
            # chama a funcao que coleta os dados de cada estacao
            print(row)
            await page_scraper(endereco)  # Await page_scraper
async def main():
  #while(1): # rodar ad eternum
  for i in range(10):
```

```
    print("I")
    obter_estacoes()
    await obtem_dados_estacoes()
    time.sleep(60)
    print("O")

if __name__ == "__main__":  # default run
    asyncio.run(main())
```

```
# PASSO 2 - REMOVIDOS DADOS POR FALTA DE ID
import re
import csv
import ast
import pytz

def pick_element(data, title):
    for sublist in data:
        if title in sublist:
            index = sublist.index(title)
            return sublist[index + 1]
    return None

def convert_txt_to_csv(txt_file, csv_file):
    with open(txt_file, 'r', encoding='utf-8') as infile, \
            open(csv_file, 'w', newline='', encoding='utf-8') as outfile:
        writer = csv.writer(outfile)
        writer.writerow(['estacao', 'datahora', 'chuva_periodo_atual',
                         'chuva_periodo_anterior', 'chuva_zeramento',
                         'temperatura_atual', 'temperatura_maxima',
'temperatura_minima',
                        #'umidade_atual', 'umidade_maxima', 'umidade_minima',
                         'vento_direcao', 'vento_velocidade',
'vento_rajada',])  # Header row
        cont = 0
        estacao = None
        datahora = None
        rows = None

        for line in infile:
            line = line.strip()
            # print(line)
            if line.startswith("Dados do scraping:"):
                cont = 0
            elif line:  # If the line is not empty
                if cont == 1:
                    estacao = line
```

```
                elif cont == 2:
                    try:
                        # Assuming '%Y-%m-%d %H:%M:%S.%f' format initially
                        dt_object = datetime.datetime.strptime(line, '%Y-%m-%d
%H:%M:%S.%f')
                    except ValueError:
                        # If the above format fails, try '%Y-%m-%d %H:%M:%S'
                        dt_object = datetime.datetime.strptime(line, '%Y-%m-%d
%H:%M:%S')
                    #datahora = line
                    #correcao timezone = GMT +6
                    local_dt =
pytz.timezone('America/Sao_Paulo').localize(dt_object)
                    #print("antes",dt_object)
                    adjusted_dt = local_dt - datetime.timedelta(hours=6)
                    datahora = adjusted_dt.astimezone(pytz.utc)
                    #print("depois",datahora)

                elif cont == 3:
                    # Safely evaluate the string representation of the list
                    rows = ast.literal_eval(line)
                    # Extract the desired values using pick_element
                    chuva_periodo_atual = pick_element(rows, "Per. Atual:")
                    chuva_periodo_anterior = pick_element(rows, "Per.
Anterior:")
                    chuva_zeramento = pick_element(rows, "Zeramento:")
                    temperatura_atual = pick_element(rows, "Atual:")
                    temperatura_maxima = pick_element(rows, "Máxima:")
                    temperatura_minima = pick_element(rows, "Mínima:")
                    #umidade_atual = pick_element(rows, "Atual:")
                    #umidade_maxima = pick_element(rows, "Máxima:")
                    #umidade_minima = pick_element(rows, "Mínima:")
                    vento_direcao = pick_element(rows, "Direção:")
                    vento_velocidade = pick_element(rows, "Velocidade:")
                    vento_rajada = pick_element(rows, "Rajada:")
                    #pressao_atual = pick_element(rows, "Atual:")
                    #pressao_maxima = pick_element(rows, "Máxima:")
                    #pressao_minima = pick_element(rows, "Mínima:")

                    # Write the extracted data to the CSV file
                    writer.writerow([estacao, datahora, chuva_periodo_atual,
                                     chuva_periodo_anterior, chuva_zeramento,
                                     temperatura_atual, temperatura_maxima,
temperatura_minima,
                                     #umidade_atual, umidade_maxima,
umidade_minima,
```

```
                                        vento_direcao, vento_velocidade,
vento_rajada,])
                                        #pressao_atual, pressao_maxima,
pressao_minima])

            cont += 1
# Call the function to convert the txt file to CSV
convert_txt_to_csv('dados_meteorologicos.txt', 'dados_meteorologicos.csv')
print("Data has been converted to dados_meteorologicos.csv")
```

```
# PASSO 3 - Adicionar em bd externo ao Colab - dados persistentes
# CORRECOES APLICADAS HORARIO E FORMATO DOS CAMPOS
import sqlite3
import pandas as pd
from google.colab import drive
drive.mount('/content/drive')
#! SALVAR OS DADOS PERSISTENTEMENTE NO DRIVE
conex = sqlite3.connect('/content/drive/MyDrive/Historico_meteorologia.db')
cursor = conex.cursor()

fixed_consulta_crtb = """
CREATE TABLE IF NOT EXISTS Historico_meteorologia (
    estacao INTEGER,
    datahora TEXT,
    chuva_periodo_atual REAL,
    chuva_periodo_anterior REAL,
    chuva_zeramento TEXT,
    temperatura_atual REAL,
    temperatura_maxima REAL,
    temperatura_minima REAL,
    vento_direcao TEXT,
    vento_velocidade REAL,
    vento_rajada REAL,
    UNIQUE(estacao, datahora)
);
"""

cursor.execute(fixed_consulta_crtb)
conex.commit()
df = pd.read_csv('dados_meteorologicos.csv')
existing_data = pd.read_sql_query("SELECT estacao, datahora FROM
Historico_meteorologia", conex)
df = df[~df[['estacao', 'datahora']].apply(tuple,
1).isin(existing_data[['estacao', 'datahora']].apply(tuple, 1))]

df.to_sql('Historico_meteorologia', conex, if_exists='append', index=False)
```

```
conex.close()
drive.flush_and_unmount()
# PASSO 4 - Verificar os dados OK
import sqlite3
import pandas as pd
from google.colab import drive
drive.mount('/content/drive')
#! SALVAR OS DADOS PERSISTENTEMENTE NO DRIVE
conex = sqlite3.connect('/content/drive/MyDrive/Historico_meteorologia.db')

cursor = conex.cursor()
consulta_all = """
SELECT * FROM Historico_meteorologia
"""
cursor.execute(consulta_all)

resultados=cursor.fetchall()
for linha in resultados:
    print(linha)

conex.close()
drive.flush_and_unmount()
```

```
# PASSO 5
# CRIAR TABELA ESTACAO POSTO PARA O BD RELACIONAL

import sqlite3
import pandas as pd
from google.colab import drive
drive.mount('/content/drive')
#! SALVAR OS DADOS PERSISTENTEMENTE NO DRIVE
conex = sqlite3.connect('/content/drive/MyDrive/Historico_meteorologia.db')
cursor = conex.cursor()
strinsql = """
CREATE TABLE IF NOT EXISTS ESTACOES (
  ESTACAO_NOME TEXT,
  ESTACAO_ID INTEGER PRIMARY KEY);
"""
cursor.execute(strinsql)
conex.commit()

df = pd.read_csv('estacoes.csv')
#conversao de nomes de campos
df = df.rename(columns={'ESTACAO': 'ESTACAO_NOME', 'POSTO': 'ESTACAO_ID'})
```

```
#FILTRO
existing_ids = pd.read_sql_query("SELECT ESTACAO_ID FROM ESTACOES",
conex)['ESTACAO_ID'].tolist()
df = df[~df['ESTACAO_ID'].isin(existing_ids)]

df.to_sql('ESTACOES', conex, if_exists='append', index=False)
conex.close()
drive.flush_and_unmount()
```

Apêndice 1.2: Códigos-fonte em Python para coleta de dados e geração de Bases de Dados (versão local)

```
# VERSAO 2.0 LOCAL
# COLETA DOS DADOS DAS ESTACOES
# 18-02-2025 SMA

#DESATIVA WARNINGS
import warnings
warnings.filterwarnings("ignore")

import psycopg2
from sqlalchemy import create_engine
from bs4 import BeautifulSoup
import csv, requests, datetime
from requests_html import HTMLSession
import asyncio
import time
import os
import nest_asyncio
from requests_html import AsyncHTMLSession
import re
import ast
import pytz
import pandas as pd
import threading

# Conectar ao banco de dados PostgreSQL usando psycopg2
dbname = 'postgres'
user = 'postgres'
password = 'Lab@2024'
host = 'localhost'
port = '5432'

def conectaBD():
    pg_conn = psycopg2.connect(
        dbname=dbname,
        user=user,
        password=password,
        host=host,
        port=port
    )
    pg_cursor = pg_conn.cursor()
    return pg_conn, pg_cursor

def encerra_conex(pg_conn):
```

```
    pg_conn.commit()
    pg_conn.close()

# Criar um engine do SQLAlchemy usando os mesmos parâmetros de conexão
def cr_engine():
    engine =
create_engine(f'postgresql+psycopg2://{user}:{password}@{host}:{port}/{dbname}
')
    return engine

global timesleep
timesleep = 100
reg=0
registros = 0
nest_asyncio.apply()
#from urllib.parse import urljoin
def numeroestacao(strURL,identif):
    text=strURL
    pos = text.find(identif)
    if pos != -1:
        return text[pos+1:]
    else:
        return text

# OBTEM A LISTA DE ESTACOES E SALVA EM ARQUIVO
def obter_estacoes():
    # simular browser
    headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36"
    }

    url = 'https://www.cgesp.org/v3/estacoes-meteorologicas.jsp'
    response = requests.get(url, headers=headers, timeout=10)
    print(response)
    try:
        # Send a GET request to the URL
        response = requests.get(url, headers=headers)
        response.raise_for_status()  # Check for HTTP errors
        # Parse the HTML content
        soup = BeautifulSoup(response.text, 'html.parser')
        # Find the <ul> element with class 'arial-data' and id 'lista-
estacoes'
        ul_element = soup.find('ul', {'class': 'arial-data', 'id': 'lista-
estacoes'})
        # Extract the data from the <ul> element
```

```
        data = []
        for li in ul_element.find_all('li'):
            a_tag = li.find('a')
            if a_tag:
                station_name = a_tag.text.strip()
                station_url =numeroestacao(a_tag['href'],"=")
                data.append((station_name, station_url))
        # Write the extracted data to a CSV file
        with open('./TMP/estacoes.csv', 'w', newline='', encoding= 'latin-
1')as file:# 'utf-8') as file:
            writer = csv.writer(file)
            writer.writerow(['ESTACAO', 'POSTO'])
            writer.writerows(data)
    except requests.exceptions.RequestException as e:
        print(f"Erro ao obter estações: {e}")
        print("Tentando novamente em 60 segundos...")
        time.sleep(60)
        obter_estacoes()  # Recursive call to retry
    print("estacoes.csv foi atualizado")

#SCRAPING
async def page_scraper(url):
    '''
    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 "
                      "(KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36"
    }
    proxies = {
    "http": "http://localhost:8888",
    "https": "http://localhost:8888",  # Optional: if you later support HTTPS
    }
    '''

    global registros
    #print(f"pagescraper called:{url}")
    estacao = numeroestacao(url, "=")
    session = AsyncHTMLSession()
    #response = await session.get(url, headers=headers, proxies=proxies)
    response = await session.get(url)
    try:
        response.html.arender(timeout=60, sleep=1)
    except TimeoutError:
        print(f"Timeout for {url}. Skipping...")
        time.sleep(5)
        return
    if response.status_code == 200:
```

```
        soup = BeautifulSoup(response.html.html, 'html.parser')
        main_filename = './TMP/' + estacao + '.html'
        with open(main_filename, 'w', encoding='latin-1')as file:#'utf-8') as
file:
            file.write(response.html.html)
        with open(main_filename, 'r', encoding='latin-1')as file:#'utf-8') as
file:
            html_content = file.read()
        #print("arquivo saida:",main_filename) #debug off
        soup = BeautifulSoup(html_content, 'html.parser')
        # Find the table within the specified div
        try:
            table = soup.find('div', {'id': 'instrumentos'}).find('table')
            if table:
                headers = [header.text for header in table.find_all('th')]
                rows = []
                for row in table.find_all('tr')[1:]:
                    cells = row.find_all('td')
                    row_data = []
                    for cell in cells:
                        inner_table = cell.find('table')
                        if inner_table:
                            inner_rows = inner_table.find_all('tr')
                            for inner_row in inner_rows:
                                inner_cells = inner_row.find_all('td')
                                for inner_cell in inner_cells:
                                    row_data.append(inner_cell.text.strip())
                        else:
                            row_data.append(cell.text.strip())
                    rows.append(row_data)
                    registros+=1
            else:
                print("tabela instrumentos não encontrada")
                return
        except Exception as e:
            print("Erro: ", e)
            return
    else:
        print(f"Pagina nao acessivel. Codigo Erro: {response.status_code}")
        return

    #extrai os dados do arquivo baixado e salva em ARQUIVO TXT
    datahora = str(datetime.datetime.now())
    with open('./TMP/dados_meteorologicos.txt', 'a', newline='',
encoding='latin-1')as file:#'utf-8') as file:
        file.write("\nDados do scraping:" + "\n")
        file.write(estacao + "\n")
```

```
        file.write(datahora + "\n")
        file.write(str(rows))
    #print("Dados escritos em dados_meteorologicos.txt")

# funcao que lista todas as ESTACOES e salva em .csv
# funcao serializada. descomente o caractere a esquerda para testar
async def obtem_dados_estacoes():
    with open('./TMP/estacoes.csv', 'r', newline='', encoding='latin-1')as
file:#'utf-8') as file:
        reader = csv.reader(file)
        # Skip the header row
        next(reader, None)
        for row in reader:
            endereco = "https://www.cgesp.org/v3/estacao.jsp?POSTO=" + row[1]
            #print(f"DEBUG endereco:  {endereco} \n nome: {row[0]}")
            await page_scraper(endereco)  # Await page_scraper

def trata_dados(elemento):
  try:
    resultado=elemento.rsplit(" ",1)
    return resultado[0]
  except:
    return None

def pick_element(data, title):
    for sublist in data:
        if title in sublist:
            index = sublist.index(title)
            return sublist[index + 1] if index + 1 < len(sublist) else None
    return None

def convert_txt_to_csv(txt_file, csv_file):
    global reg
    #with open(txt_file, 'r', encoding='utf-8') as infile, \
    #        open(csv_file, 'w', newline='', encoding='utf-8') as outfile:
    with open(txt_file, 'r', encoding='latin-1') as infile, \
            open(csv_file, 'w', newline='', encoding='latin-1') as outfile:
        writer = csv.writer(outfile)
        writer.writerow(['estacao', 'datahora', 'chuva_periodo_atual',
                         'chuva_periodo_anterior', 'chuva_zeramento',
                         'temperatura_atual', 'temperatura_maxima',
'temperatura_minima',
                        #'umidade_atual', 'umidade_maxima', 'umidade_minima',
                         'vento_direcao', 'vento_velocidade',
'vento_rajada',])  # Header row
        cont = 0
```

```
        estacao = None
        datahora = None
        rows = None

        for line in infile:
            line = line.strip()
            # print(line)
            if line.startswith("Dados do scraping:"):
                cont = 0
            elif line:  # If the line is not empty
                if cont == 1:
                    estacao = line
                elif cont == 2:
                    try:
                        dt_object = datetime.datetime.strptime(line, '%Y-%m-%d
%H:%M:%S.%f')
                    except ValueError:
                        dt_object = datetime.datetime.strptime(line, '%Y-%m-%d
%H:%M:%S')
                    #correcao timezone = GMT +6
                    local_dt =
pytz.timezone('America/Sao_Paulo').localize(dt_object)
                    adjusted_dt = local_dt - datetime.timedelta(hours=3)
                    datahora = adjusted_dt.astimezone(pytz.utc)

                elif cont == 3:
                    # Safely evaluate the string representation of the list
                    rows = ast.literal_eval(line)
                    # Extract the desired values using pick_element
                    chuva_periodo_atual = trata_dados(pick_element(rows, "Per.
Atual:"))
                    chuva_periodo_anterior = trata_dados(pick_element(rows,
"Per. Anterior:"))
                    chuva_zeramento = trata_dados(pick_element(rows,
"Zeramento:"))
                    temperatura_atual = trata_dados(pick_element(rows,
"Atual:"))
                    temperatura_maxima = trata_dados(pick_element(rows,
"Máxima:"))
                    temperatura_minima = trata_dados(pick_element(rows,
"Mínima:"))
                    #umidade_atual = pick_element(rows, "Atual:") # DADOS COM
MUITAS FALHAS -- ignorados
                    #umidade_maxima = pick_element(rows, "Máxima:")
                    #umidade_minima = pick_element(rows, "Mínima:")
                    vento_direcao = trata_dados(pick_element(rows,
"Direção:"))
```

```
                    vento_velocidade = trata_dados(pick_element(rows,
"Velocidade:"))
                    vento_rajada = trata_dados(pick_element(rows, "Rajada:"))
                    #pressao_atual = pick_element(rows, "Atual:")
                    #pressao_maxima = pick_element(rows, "Máxima:")
                    #pressao_minima = pick_element(rows, "Mínima:")

                    # Write the extracted data to the CSV file
                    writer.writerow([estacao, datahora, chuva_periodo_atual,
                                     chuva_periodo_anterior, chuva_zeramento,
                                     temperatura_atual, temperatura_maxima,
temperatura_minima,
                                     #umidade_atual, umidade_maxima,
umidade_minima,
                                     vento_direcao, vento_velocidade,
vento_rajada,])
                                     #pressao_atual, pressao_maxima,
pressao_minima])
                    #numero de registros
                    reg+=1
            cont += 1

async def runcoleta(ts_anterior):
  #global registros, timesleep,ts_anterior
  obter_estacoes()
  await obtem_dados_estacoes()
  time.sleep(20)
  reg=0
  if not os.path.exists('./TMP/dados_meteorologicos.csv'):
      print("falha no arquivo")
      with open('./TMP/dados_meteorologicos.csv', 'w', newline='',
encoding='latin-1') as file:#'utf-8') as file:
          pass
  convert_txt_to_csv('./TMP/dados_meteorologicos.txt',
'./TMP/dados_meteorologicos.csv')
  print(f"{reg} linhas adicionadas a dados_meteorologicos.csv")
  pg_conn, pg_cursor = conectaBD()
  df = pd.read_csv('./TMP/dados_meteorologicos.csv', encoding='latin-1')
  existing_data = pd.read_sql_query("SELECT estacao, datahora FROM
Historico_meteorologia", pg_conn)
  df = df[~df[['estacao', 'datahora']].apply(tuple,
1).isin(existing_data[['estacao', 'datahora']].apply(tuple, 1))]
  consulta_verifica = """
    SELECT
        estacoes.estacao_nome, historico_meteorologia.chuva_periodo_atual,
        ((EXTRACT(EPOCH FROM NOW()) - EXTRACT(EPOCH FROM
TO_TIMESTAMP(datahora, 'YYYY-MM-DD HH24:MI:SS'))) / 60 - 180)  AS dif
```

```
    FROM
        Historico_meteorologia
    INNER JOIN
        estacoes
    ON
        Historico_meteorologia.estacao = estacoes.estacao_id
    WHERE
        historico_meteorologia.chuva_periodo_atual!='NaN' AND(EXTRACT(EPOCH
FROM NOW()) - EXTRACT(EPOCH FROM TO_TIMESTAMP(datahora, 'YYYY-MM-DD
HH24:MI:SS'))) / 60 - 180 < 300
    GROUP BY
        estacoes.estacao_nome,dif,datahora,chuva_periodo_atual
    HAVING
        chuva_periodo_atual > 1
  """

  pg_cursor.execute(consulta_verifica)
  resultados=pg_cursor.fetchall()
  #CONTROLE DE TIMESLEEP PARA HORÁRIOS COM / SEM CHUVA
  print(f"debug {len(resultados)}")
  if len(resultados) == 0:
    ts_anterior=timesleep
    timesleep = 600
    print("Ajustando timesleep para 10 minutos")
  if len(resultados) == 1:
    ts_anterior=timesleep
    timesleep = 100
    if ts_anterior != timesleep:
        print("Ajustando timesleep para 100 segundos")
  elif len(resultados)>10:
    timesleep = 60
    if ts_anterior != timesleep:
        print("Ajustando timesleep para 60 segundos")

  #salvar dados de forma persistente
  consulta_insere = """
  INSERT INTO historico_meteorologia (estacao, datahora, chuva_periodo_atual,
chuva_periodo_anterior, chuva_zeramento, temperatura_atual,
temperatura_maxima, temperatura_minima, vento_direcao, vento_velocidade,
vento_rajada)
  VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
  """
  for index, row in df.iterrows():  # Iterate through DataFrame rows
    try:  # Handle potential errors during insertion
        pg_cursor.execute(consulta_insere, (
            row['estacao'], row['datahora'], row['chuva_periodo_atual'],
            row['chuva_periodo_anterior'], row['chuva_zeramento'],
```

```
            row['temperatura_atual'], row['temperatura_maxima'],
            row['temperatura_minima'], row['vento_direcao'],
            row['vento_velocidade'], row['vento_rajada']
        ))
    except Exception as e:
        #print(f"Error inserting row: {row}")
        #print(f"Error details: {e}")
        pg_conn.rollback()  # Rollback on error to prevent partial inserts
        continue # Skip to the next row
    pg_conn.commit()
print("Dados inseridos com sucesso! ", time.ctime())
consulta_all = """
SELECT COUNT(*) FROM Historico_meteorologia
"""
pg_cursor.execute(consulta_all)
resultados=pg_cursor.fetchall()
#print("DEBUG resultados consulta:", resultados)
encerra_conex(pg_conn)
```

```
### COLETA DOS DADOS DE ALAGAMENTOS DO CGE - AUTOMATIZADO
# 18/02/2025 Via pc local
# 29.01.2025 SMA
# 30.01.2025 Testes OK

import urllib, re
import urllib.request
import socket
import psycopg2
from bs4 import BeautifulSoup
import datetime
import pandas as pd
import urllib
from sqlalchemy import create_engine

def conectaBD():
    # Conectar ao banco de dados PostgreSQL usando psycopg2
    dbname = 'postgres'
    user = 'postgres'
    password = 'Lab@2024'
    host = 'localhost'
    port = '5432'
    pg_conn = psycopg2.connect(
        dbname=dbname,
        user=user,
        password=password,
        host=host,
        port=port
    )
    pg_cursor = pg_conn.cursor()
    return pg_conn, pg_cursor

def encerra_conex(pg_conn):
    if pg_conn:
        pg_conn.commit()
        pg_conn.close()

# Criar um engine do SQLAlchemy usando os mesmos parâmetros de conexão
def cr_engine(user,password,host,port,dbname):
    engine =
create_engine(f'postgresql+psycopg2://{user}:{password}@{host}:{port}/{dbname}
')
    return engine

# Função para listar todas as datas entre duas datas fornecidas (inclusive)
(ger. copilot)
```

```
def list_dates_between(start_date_str, end_date_str):
    """Lists all dates between two given dates (inclusive).
    Args:
        start_date_str: Start date string in "YYYY-MM-DD" format.
        end_date_str: End date string in "YYYY-MM-DD" format.
    Returns:
        A list of date objects representing all dates between the start and
end dates.
    """
    # Convert date strings to datetime objects
    start_date = datetime.datetime.strptime(start_date_str, "%Y-%m-%d").date()
    end_date = datetime.datetime.strptime(end_date_str, "%Y-%m-%d").date()

    # Generate a list of dates
    dates = []
    delta = datetime.timedelta(days=1)  # Increment by 1 day
    current_date = start_date
    while current_date <= end_date:
        dates.append(current_date)
        current_date += delta
    return dates

# GERA A STRING PARA O FETCH
def montastringdata(dia, mes,ano):
  return
"https://www.cgesp.org/v3/alagamentos.jsp?dataBusca="+str(dia)+"%2F"+str(mes)+
"%2F"+str(ano)+"&enviaBusca=Buscar"
  pass

def sepstrhoralocal(strhl):
  pattern = r"De\s+(\d{2}:\d{2})\s+a\s+(\d{2}:\d{2})(.*)"
  match = re.search(pattern, strhl)
  var1='null'
  var2='null'
  var3='null'
  if match:
    var1 = match.group(1)  # 15:56
    var2 = match.group(2)  # 19:24
    var3 = match.group(3)  # AV ORDEM E PROGRESSO
    return (var1,var2,var3)
  else:
    print(f"debug: falha provavel string vazia")
    return (var1,var2,var3)

def sepstrsentref(strsr):
  try:
```

```
    # Split the string using 'Sentido:' as the delimiter
    parts = strsr.split("Sentido:")
    if len(parts) > 1:
      # Get the part after 'Sentido:'
      after_sentido = parts[1]
      # Split the remaining part using 'Referência:' as the delimiter
      parts2 = after_sentido.split("Referência:")
      varS = 'null'
      varS = 'null'
      if len(parts2) > 1:
        # Extract and strip the values
        varS = parts2[0].strip()
        varR = parts2[1].strip()
        return (varS, varR)
      else:
        return (varS, varR)
  except:
    pass

socket.setdefaulttimeout(10)

def obterdados(dia,mes,ano):
  conn, cursos = conectaBD()
  cursor = conn.cursor()
  url = montastringdata(dia,mes,ano)
  fileobj = urllib.request.urlopen(url)
  html_content = fileobj.read().decode('utf-8')  # Decode the byte string
  sopa = BeautifulSoup(html_content, 'html.parser') # Use the new variable
name
  table = sopa.find('div', class_='yui3-u col-alagamentos')
  naoharegistros = sopa.find('h2',string='Não há registros de alagamentos para
essa data.')
  if naoharegistros != None:
    print(f"debug: data sem registros {dia}/{mes}/{ano}" )
    return
  bairros = sopa.find_all('td', class_='bairro arial-bairros-alag linha-
pontilhada')
  NUM_BAIRROS=len(bairros)
  PTOS_BRT = sopa.find_all('td', class_='total-pts arial-bairros-alag')

  # Changed REGS_BAIRRO initialization to a list
  REGS_BAIRRO = []
  NOME_BAIRRO = []
  for i in range(NUM_BAIRROS):
    try:  # Add a try-except block for safety
      REGS_BAIRRO.append(int(PTOS_BRT[i].get_text().strip().split("pts.")[0]))
      NOME_BAIRRO.append(str((bairros[i].get_text().strip())))
```

```
    except (IndexError, ValueError):
      print(f"Warning: Could not extract value for bairro {i}")
  print(f"regs bairr:{NOME_BAIRRO}") #10
  print(f"regs bairr:{REGS_BAIRRO}") #10

  #contador registros do bairro (n-1) - o primeiro registro com bug, verificar
  ITEM_BAIRRO = 0 #contador dentro bairro
  NBAIRRO_ATUAL = 0 #contador bairros
  CONT_GERAL = 0 #contador geral (rows)
  provavelest = NOME_BAIRRO[0]
  all_data = []

  if table:  # Check if the table was found before proceeding
      for row in table.find_all('tr')[1:]:
          try:
            if  ITEM_BAIRRO>=(REGS_BAIRRO[NBAIRRO_ATUAL]) : #Se o elemento
dentro do bairro for superior a qt registros
              ITEM_BAIRRO=0
              if NBAIRRO_ATUAL < NUM_BAIRROS:
                NBAIRRO_ATUAL+=1
                provavelest = NOME_BAIRRO[NBAIRRO_ATUAL]
              else:
                  # Handle the case where el_bairro is out of bounds
                  print("el_bairro out of bounds for bairrodesc")
                  break  # Or handle it differently
          except IndexError as e:
            print(f"Fora dos limites{e}")
            break
          div = row.find('div', class_='ponto-de-alagamento')
          #ADICAO DADOS À LISTA
          if div:  # Check if the div was found
              CONT_GERAL+=1
              ITEM_BAIRRO+=1
              # Get the transitabilidade value
              transitabilidade = div.find('li')['class'][0]
              details = [li.get_text(strip=True) for li in div.find_all('li')]
              #print(details[2])
              horai, horaf, local = sepstrhoralocal(details[2])  # Changed to
sephoraflocal
              sentido, referencia = sepstrsentref(details[4])
              # Append to data list as a dictionary
#DHICORRIGIDO                #DHFCORRIGIDO                #Formato:
#"2024-12-27 08:16:00"
              all_data.append({
                  'endereco': local,
                  'referencia': referencia,
                  'provavelest': provavelest,
```

```
                'sentido': sentido,
                'transitabilidade': transitabilidade,
                'DHICORRIGIDO': str(ano) +"-" + str(mes) +"-"+ str(dia) +"
"+ horai,
                'DHFCORRIGIDO': str(ano) +"-" + str(mes) +"-"+ str(dia) +"
"+ horaf
            })

  df = pd.DataFrame(all_data)
  existing_data = pd.read_sql_query("SELECT endereco, referencia, DHICORRIGIDO
FROM ALAGAMENTOS", conn)
  df['DHICORRIGIDO'] = pd.to_datetime(df['DHICORRIGIDO'], format='%Y-%m-%d
%H:%M', errors='coerce')  # errors='coerce' handles invalid dates
  df['DHFCORRIGIDO'] = pd.to_datetime(df['DHFCORRIGIDO'], format='%Y-%m-%d
%H:%M', errors='coerce')  # errors='coerce' handles invalid dates
  print(df)
  print(df[['endereco', 'referencia', 'DHICORRIGIDO']].dtypes)

  try:
    df_no_duplicates = df.drop_duplicates(subset=['endereco', 'referencia',
'DHICORRIGIDO'])
    consulta_insere = """
    INSERT INTO ALAGAMENTOS (endereco, referencia, provavelest, sentido,
transitabilidade, DHICORRIGIDO, DHFCORRIGIDO)
    VALUES (%s, %s, %s, %s, %s, %s, %s)
    """
    pg_conn, pg_cursor = conectaBD()

    for index, row in df_no_duplicates.iterrows():
        try:
            pg_cursor.execute(consulta_insere, (
                row['endereco'], row['referencia'], row['provavelest'],
                row['sentido'], row['transitabilidade'],
                row['DHICORRIGIDO'], row['DHFCORRIGIDO']
            ))
            print("Data inserted successfully")
        except psycopg2.Error as e: # Catch psycopg2 errors
            print(f"Error inserting row: {row}")
            print(f"Error details: {e}")
            pg_conn.rollback()
            continue  # Skip to the next row
        except Exception as e: # Catch other errors
            print(f"Error inserting row: {row}")
            print(f"Error details: {e}")
            pg_conn.rollback()
            continue  # Skip to the next row
```

```
    pg_conn.commit()
    encerra_conex(pg_conn)

  except Exception as e:
            print("Erro: ", e)
            encerra_conex(pg_conn)
            return

# desatualizado com o frontend, adicionar em uma funcao principal se for
utilizar
def func():
  # Solicitar entrada do usuário para data inicial e final separadamente
  dia_i = int(input("Digite o dia inicial: "))
  mes_i = int(input("Digite o mês inicial: "))
  ano_i = int(input("Digite o ano inicial: "))

  dia_f = int(input("Digite o dia final: "))
  mes_f = int(input("Digite o mês final: "))
  ano_f = int(input("Digite o ano final: "))

  # Criar objetos de data a partir das entradas
  datai_object = datetime.date(ano_i, mes_i, dia_i)
  dataf_object = datetime.date(ano_f, mes_f, dia_f)

  print("Data inicial:", datai_object)
  print("Data final:", dataf_object)
  dates_between = list_dates_between(str(datai_object), str(dataf_object))
  # Imprimir as datas no formato DD/MM/YYYY e retornar três valores (dia, mês,
ano) sem zeros à esquerda
  for date in dates_between:
      day = date.day
      month = date.month
      year = date.year
      print(f"Dia: {day}, Mês: {month}, Ano: {year}")
      obterdados(day,month,year)
```

```python
# Interface para o usuário COLETA

import tkinter as tk
from prox_emulator import start_proxy_in_background
from tkinter import messagebox
from coleta import runcoleta
import asyncio
import threading
import time
import sys

import os
import nest_asyncio

class TextRedirector:
    def __init__(self, widget):
        self.widget = widget

    def write(self, text):
        self.widget.insert(tk.END, text)
        self.widget.see(tk.END)

    def flush(self):
        pass

def start_collection():
    global running
    running = True
    status_label.config(text="Status: Coletando dados", fg="green")
    messagebox.showinfo("Iniciando", "A coleta de dados foi iniciada.")
    thread = threading.Thread(target=run_async)
    thread.start()

def stop_collection():
    global running
    running = False
    status_label.config(text="Status: Parado", fg="red")
    messagebox.showinfo("Parando", "A coleta de dados foi interrompida.")

def run_async():
    asyncio.run(main())

def update_results(text):
    result_label.config(text=f"Última atualização: {text}")
```

```
async def main():
    global running, timesleep
    #proxy_server = start_proxy_in_background()
    counter = 0
    while running:
        timesleep = 60
        ts_anterior = timesleep
        print("INÍCIO: (hora atual sem fuso)", time.ctime())

        try:
            for i in range(10000):
                await runcoleta(60)  # Added await here
                print(f"iteracao {i}: sleep {timesleep}, {ts_anterior}")
                await asyncio.sleep(timesleep)  # Use asyncio.sleep instead of
time.sleep

                counter += 1
                output = f"Iteração {counter}: coletando dados...\n"
                print(output, end="")
                update_results(output.strip())

        except Exception as e:
            print(f"Erro durante a execução: {e}")
            break



def create_gui():
    global status_label, result_label, log_text
    root = tk.Tk()
    root.title("Monitoramento de Estações Meteorológicas")
    root.geometry("600x400")

    label = tk.Label(root, text="Controle de Coleta de Dados", font=("Arial",
14))
    label.pack(pady=10)

    status_label = tk.Label(root, text="Status: Parado", font=("Arial", 12),
fg="red")
    status_label.pack(pady=5)

    start_button = tk.Button(root, text="Iniciar Coleta",
command=start_collection, width=20)
    start_button.pack(pady=5)
```

```
    stop_button = tk.Button(root, text="Parar Coleta",
command=stop_collection, width=20)
    stop_button.pack(pady=5)

    result_label = tk.Label(root, text="Última atualização: -", font=("Arial",
10))
    result_label.pack(pady=5)

    log_text = tk.Text(root, height=10, width=70)
    log_text.pack(pady=5)

    sys.stdout = TextRedirector(log_text)
    sys.stderr = TextRedirector(log_text)

    exit_button = tk.Button(root, text="Sair", command=root.quit, width=20)
    exit_button.pack(pady=5)

    root.mainloop()

if __name__ == "__main__":
    running = False
    create_gui()
```

```python
# FRONTEND ALAGAMENTOS
# 01/04/2025 SMA

import tkinter as tk
from tkinter import ttk, messagebox
from datetime import datetime
import threading
from coleta_alagamentos import obterdados, list_dates_between

# Create main window
root = tk.Tk()
root.title("Coleta de Dados de Alagamentos")
root.geometry("600x400")

# Global state
is_running = False
dates_between = []

# Create main frame
main_frame = ttk.Frame(root, padding="10")
main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

def on_closing():
    global is_running
    if is_running:
        if not messagebox.askyesno("Confirmação", "A coleta ainda está em
andamento. Deseja sair?"):
            return  # Do nothing if the user cancels
        is_running = False  # Stop the background process
    root.destroy()  # Close the window

# Create date input fields
def create_date_inputs():
    global dia_i, mes_i, ano_i, dia_f, mes_f, ano_f

    initial_frame = ttk.LabelFrame(main_frame, text="Data Inicial",
padding="5")
    initial_frame.grid(row=0, column=0, padx=5, pady=5, sticky="nsew")

    dia_i = ttk.Entry(initial_frame, width=3)
    mes_i = ttk.Entry(initial_frame, width=3)
    ano_i = ttk.Entry(initial_frame, width=5)

    ttk.Label(initial_frame, text="Dia:").grid(row=0, column=0)
    dia_i.grid(row=0, column=1)
```

```
    ttk.Label(initial_frame, text="Mês:").grid(row=0, column=2)
    mes_i.grid(row=0, column=3)
    ttk.Label(initial_frame, text="Ano:").grid(row=0, column=4)
    ano_i.grid(row=0, column=5)

    final_frame = ttk.LabelFrame(main_frame, text="Data Final", padding="5")
    final_frame.grid(row=0, column=1, padx=5, pady=5, sticky="nsew")

    dia_f = ttk.Entry(final_frame, width=3)
    mes_f = ttk.Entry(final_frame, width=3)
    ano_f = ttk.Entry(final_frame, width=5)

    ttk.Label(final_frame, text="Dia:").grid(row=0, column=0)
    dia_f.grid(row=0, column=1)
    ttk.Label(final_frame, text="Mês:").grid(row=0, column=2)
    mes_f.grid(row=0, column=3)
    ttk.Label(final_frame, text="Ano:").grid(row=0, column=4)
    ano_f.grid(row=0, column=5)

# Create progress indicators
def create_progress_indicators():
    global progress_var, progress_bar
    progress_frame = ttk.LabelFrame(main_frame, text="Progresso", padding="5")
    progress_frame.grid(row=2, column=0, columnspan=2, padx=5, pady=5,
sticky="nsew")
    progress_var = tk.StringVar(value="0%")
    progress_bar = ttk.Progressbar(progress_frame, length=400,
mode='determinate')
    progress_bar.grid(row=0, column=0, padx=5, pady=5)
    ttk.Label(progress_frame, textvariable=progress_var).grid(row=0, column=1,
padx=5)

# Create buttons
def create_buttons():
    global start_button, stop_button

    button_frame = ttk.Frame(main_frame)
    button_frame.grid(row=3, column=0, columnspan=2, pady=10)

    start_button = ttk.Button(button_frame, text="Iniciar Coleta",
command=start_collection)
    start_button.grid(row=0, column=0, padx=5)

    stop_button = ttk.Button(button_frame, text="Parar",
command=stop_collection, state="disabled")
    stop_button.grid(row=0, column=1, padx=5)
```

```
# Log messages to status box
def log_message(message):
    status_text.insert(tk.END, f"{message}\n")
    status_text.see(tk.END)

# Start data collection
def start_collection():
    global is_running, dates_between

    try:
        datai = datetime(int(ano_i.get()), int(mes_i.get()), int(dia_i.get()))
        dataf = datetime(int(ano_f.get()), int(mes_f.get()), int(dia_f.get()))

        dates_between = list_dates_between(datai.strftime("%Y-%m-%d"),
dataf.strftime("%Y-%m-%d"))

        is_running = True
        start_button.config(state="disabled")
        stop_button.config(state="normal")

        # Start collection in a separate thread
        collection_thread = threading.Thread(target=run_collection)
        collection_thread.start()

    except ValueError:
        messagebox.showerror("Erro", "Datas inválidas!")

# Stop data collection
def stop_collection():
    global is_running
    is_running = False
    start_button.config(state="normal")
    stop_button.config(state="disabled")

# Run data collection
def run_collection():
    global is_running

    total_dates = len(dates_between)
    for i, date in enumerate(dates_between):
        if not is_running:
            break

        # Update progress
```

```
            progress = (i + 1) / total_dates * 100
            progress_bar["value"] = progress
            progress_var.set(f"{progress:.1f}%")

            # Log processing date
            log_message(f"Processando {date.strftime('%d/%m/%Y')}...")

            # Call data collection function
            try:
                obterdados(date.day, date.month, date.year)
            except Exception as e:
                log_message(f"Erro: {str(e)}")

            root.update()

    log_message("Coleta finalizada!")
    start_button.config(state="normal")
    stop_button.config(state="disabled")

# UI setup
create_date_inputs()
create_progress_indicators()
create_buttons()

# Status text area
status_text = tk.Text(main_frame, height=10, width=50)
status_text.grid(row=4, column=0, columnspan=2, pady=10)

# Start main loop
root.mainloop()

# Bind the close event (X button)
root.protocol("WM_DELETE_WINDOW", on_closing)
```

Apêndice 1.3: Códigos-fonte em Python para Classificação de textos e acesso ao Sharepoint (planilhas Excel)

```
# CLASSIFICADOR – acessoBD.py
# SMA 25/03/2025
# FUNCOES PARA ACESSO AS BASES DE DADOS E DADOS DE TESTE
# SEM USO CASO ACESSO DIRETO A PLANILHAS

import numpy as np
import pandas as pd
import psycopg2
import spacy
from preprocess import define_head

# Load spaCy model
nlp = spacy.load("pt_core_news_md")

def conecta_bd():
    # Conectar ao banco de dados PostgreSQL usando psycopg2
    try:
        pg_conn = psycopg2.connect(
            dbname='classificacao',
            user='postgres',
            password='Lab@2024',
            host='localhost',
            port='5432'
        )
        pg_cursor = pg_conn.cursor()
    except psycopg2.Error as e:
        print(f"Erro ao conectar ao banco de dados PostgreSQL: {e}")
        exit()
    return pg_conn, pg_cursor

def encerra_bd(pg_conn, pg_cursor):
    pg_cursor.close()
    pg_conn.close()
    return

def importar_dados(pg_conn):
    query = "SELECT * FROM temas"
    dataf = pd.read_sql(query, pg_conn)
    return dataf

def consulta_db(pg_conn, pg_cursor, predicted_label): # funcao para obter a
coluna do dado classificado
    query = f"SELECT * FROM temas WHERE nid = '{predicted_label}'"
```

```
    pg_cursor.execute(query)
    result = pg_cursor.fetchone()
    return result

def concatenate_columns(row):
    return ' '.join(row.dropna().astype(str))

# O DF['PROCESSED_TEXT'] CONTEM O TEXTO COMPLETO SEM ACRÔNIMOS
def inicializa_dataframe():
    conn, curs = conecta_bd()
    df = importar_dados(conn)
    df['head'] = df['servico'].apply(define_head)
    df['combined_text'] = df[['tema', 'assunto',
'servico']].apply(concatenate_columns, axis=1)
    encerra_bd(conn, curs)
    return df

def obter_dados_teste():
# Open the CSV file and read it line by line
    with open('dados_clas.csv', 'r',encoding='utf-8', errors='ignore') as
file:
        lines = file.readlines()
    # Create a DataFrame with each line as a single row in one column
    df = pd.DataFrame(lines, columns=['Text'])
    # Optionally, check the first few rows
    #print(df.head())
    return df
```

```python
# CLASSIFICADOR – dicionario.py

import re
import spacy

nlp = spacy.load("pt_core_news_md")
# Custom list of common Portuguese words
common_words = set([
    "a", "as", "à", "às", "ao", "aos", "aquela", "aquelas", "aquele",
"aqueles", "aquilo",
    "as", "às", "até", "com", "como", "da", "das", "de", "dela", "delas",
    "dele", "deles", "depois", "do", "dos", "e", "ela", "elas", "ele",
    "eles", "em", "entre", "entretanto", "era", "essa", "essas", "esse",
    "esses", "esta", "está", "estas", "este", "estes", "estão", "eu",
    "foi", "foram", "fosse", "há", "havia", "isto", "isso", "já", "lhe",
    "lhes", "mais", "mas", "me", "mesmo", "meu", "meus", "minha", "minhas",
    "muito", "na", "nas", "nem", "no", "nos", "nossa", "nossas", "nosso",
    "nossos", "nós", "num", "numa", "não", "o", "oi", "olá", "onde", "os",
"ou",
    "para", "pela", "pelas", "pelo", "pelos", "por", "porém","pouca",
"pouco","poucas","poucos"
    "quais", "qual", "quando","quanta","quanto","quantas","quantos",
    "que", "quem", "se", "seja", "sr","sra","sras","srs", "sem", "ser",
"será", "seu", "seus",
    "só", "sua", "suas", "também", "te", "tem", "tenho", "ter", "têm",
    "tinha", "tu", "tua", "tuas", "teu", "teus", "tu", "um", "uma", "você",
    "vocês", "vos"
])

def extrair_maiusculas(texto):
    padrao = r"[A-Z]{2,}"  # Encontra 2 ou mais letras maiúsculas consecutivas
    return re.findall(padrao, texto)

def extrair_pcomuns(texto):
    """
    Clean text by removing common words, hyphens, and special characters

    Args:
        texto (str): Input text to clean

    Returns:
        str: Cleaned text without common words and special characters
    """
    try:
        # Handle list input or convert to string
        if isinstance(texto, list):
```

```
            texto = ' '.join(map(str, texto))

        # Convert to lowercase and normalize text
        texto = str(texto).lower().strip()

        # Remove hyphens and standardize spacing
        texto = texto.replace(' - ', ' ').replace('-', ' ')

        # Split into words
        palavras = texto.split()

        # Remove common words and clean special characters
        cleaned_words = []
        for palavra in palavras:
            # Remove special characters
            palavra = re.sub(r'[^\w\s]', '', palavra)

            # Skip if word is empty or in common words list
            if not palavra or palavra in common_words:
                continue

            # Skip single characters
            if len(palavra) <= 1:
                continue

            cleaned_words.append(palavra)

        # Join words back together
        texto_limpo = ' '.join(cleaned_words)

        return texto_limpo

    except Exception as e:
        print(f"Error cleaning text: {e}")
        return ""


def atualiza_dicionario_acronimos(textos, acronym_dict):
    for text in textos:
        palavras = text.split()
        for i, palavra in enumerate(palavras):
            if re.match(r"[A-Z]{2,}", palavra) and palavra.lower() not in
common_words:
                # Get the previous three words and the next three words around
the acronym
```

```
                prev_words = palavras[max(0, i-3):i]
                next_words = palavras[i+1:i+4]
                filtered_next_words = [w for w in next_words if
w.startswith(palavra[0])]
                if len(prev_words) < 3:
                    prev_words = [""] * (3 - len(prev_words)) + prev_words
                if len(next_words) < 3:
                    next_words += [""] * (3 - len(next_words))
                context = {'prev_words': prev_words, 'next_words': next_words}
                if palavra in acronym_dict:
                    acronym_dict[palavra]['contexts'].append(context)
                else:
                    acronym_dict[palavra] = {'contexts': [context]}
    return acronym_dict

# REMOÇÃO DE PALAVRAS COM SIGNIFICADO DOS ACRÔNIMOS
def remove_psig_acron(palavra):
    doc = nlp(palavra)
    token = doc[0]
    token.similarity
    print(token.is_alpha,token.has_vector,token.vector_norm)
    return token.is_alpha,token.has_vector,token.vector_norm
```

```python
# CLASSIFICADOR - EXCEL.py

import win32com
import pandas as pd
from openpyxl import load_workbook, Workbook
from openpyxl.utils.exceptions import InvalidFileException

def resolve_shortcut(shortcut_path):
    """Resolve um atalho (.lnk) para obter o caminho real do arquivo."""
    shell = win32com.client.Dispatch("WScript.Shell")
    shortcut = shell.CreateShortcut(shortcut_path)
    return shortcut.TargetPath

def read_excel_tables(file_path, sheet_name=0, header=None):
    """
    Reads tables from an Excel file into a list of pandas DataFrames.

    Args:
        file_path (str): The path to the Excel file.
        sheet_name (str or int, optional): The name or index of the sheet to
read. Defaults to 0 (first sheet).
        header (int, list of int, or None, optional): Row number(s) to use as
column names, and the start of the data.
                                                    Defaults to None (pandas
infers header).

    Returns:
        list of pandas.DataFrame: A list containing the extracted tables as
DataFrames.
                                  Returns an empty list if no tables are found
or an error occurs.
    """
    try:
        # Read the entire sheet into a DataFrame
        df = pd.read_excel(file_path, sheet_name=sheet_name, header=header)

        # Find rows where all values are NaN (potential table separators)
        nan_rows = df.isna().all(axis=1)

        # Find the indices of these NaN rows
        nan_row_indices = nan_rows[nan_rows].index.tolist()

        # Split the DataFrame into tables based on NaN rows
        tables = []
        start_index = 0
```

```
        for end_index in nan_row_indices:
            table = df.iloc[start_index:end_index].dropna(axis=0,
how='all').dropna(axis=1, how='all') #remove rows and columns that are all NaN
            if not table.empty:
                tables.append(table)
            start_index = end_index + 1

        # Add the last table (if any)
        last_table = df.iloc[start_index:].dropna(axis=0,
how='all').dropna(axis=1, how='all')
        if not last_table.empty:
            tables.append(last_table)

        return tables

    except Exception as e:
        print(f"An error occurred: {e}")
        return []

def safe_excel_write(planilha, row, col, value):
    """Helper function to safely write values to Excel cells"""
    if isinstance(value, (list, np.ndarray)):
        value = str(value[0]) if len(value) > 0 else ""
    value = str(value).strip('[]\'\"')
    planilha.cell(row=row, column=col, value=value)

def list_sheets(filepath):
  """Lists the names of all sheets in an Excel file.

  Args:
    filepath: The path to the Excel (.xlsx) file.

  Returns:
    A list of sheet names, or None if an error occurs.
  """
  try:
    workbook = load_workbook(filepath)
    return workbook.sheetnames
  except FileNotFoundError:
    print(f"Error: File not found at {filepath}")
    return None
  except Exception as e:
    print(f"An error occurred: {e}")
    return None
```

```python
def load_excel_file(file_path, sheet_name):
    """
    Load Excel file once and return both openpyxl workbook and pandas
DataFrame

    Args:
        file_path (str): Path to Excel file
        sheet_name (str): Name of sheet to load

    Returns:
        tuple: (workbook, worksheet, dataframe)
    """
    try:
        # Load with openpyxl for writing
        workbook = load_workbook(file_path)
        worksheet = workbook[sheet_name]

        # Load same file with pandas for data analysis
        df = pd.read_excel(file_path, sheet_name=sheet_name)

        return workbook, worksheet, df

    except Exception as e:
        print(f"Error loading Excel file: {e}")
        return None, None, None

def add_values_to_table(filepath, sheet_name, table_name, data,row_index,
col_index):
    """
    Adds values to a table in an Excel sheet without altering its structure.

    Args:
        filepath: Path to the Excel file.
        sheet_name: Name of the sheet containing the table.
        table_name: Name of the table.
        data: A list of dictionaries, where each dictionary represents a row
of data.
              Keys in the dictionaries should match the table's column
headers.
    """
    try:
        workbook = load_workbook(filepath)
        sheet = workbook[sheet_name]

        print("debug", workbook,sheet)
        # Find the table
```

```
        for tbl in sheet.tables.values():
            if tbl.name == table_name:
                table = tbl
                break

        if not table:
            print(f"Error: Table '{table_name}' not found in sheet
'{sheet_name}'.")
            return

        sheet.cell(row=row_index, column=col_index, value=data)
        print("debug sc", sheet.cell(row=row_index, column=col_index,
value=data))
        try:
            workbook.save(filepath)
            print("Data added successfully.")
        except InvalidFileException as e:
            if "Registros Reparados" in str(e): #check the error message
                print("Warning: Excel file repaired during save. Please verify
the table.")
                workbook = load_workbook(filepath) #Reload the file to avoid
further issues.
                workbook.save(filepath + "_repaired.xlsx") # save the repaired
file with a new name.
                print(f"Repaired file saved to {filepath}_repaired.xlsx")
            else:
                print(f"An error occurred during save: {e}")
        except Exception as e:
            print(f"An error occurred during save: {e}")
        print("Data added successfully.")
    except FileNotFoundError:
        print(f"Error: File not found at {filepath}")
    except KeyError:
        print(f"Error: Sheet '{sheet_name}' not found.")
    except Exception as e:
        print(f"An error occurred: {e}")
```

```python
# CLASSIFICADOR funcoes_similaridade.py
import re
import spacy
import random
#from dicionario import extrair_maiusculas
from preprocess import preprocess_text_no_stopwords, encontra_potacron

nlp = spacy.load("pt_core_news_md")


def calculate_acronym_weight(acronym, df):
    """
    Calculate normalized weight for acronym based on:
    1. Relative frequency in dataset (0-0.6)
    2. Acronym specificity (0-0.4)

    Returns: float between 0 and 1
    """
    # Get occurrence statistics
    occurrences = df[df['acron'] == acronym].shape[0]
    total_records = len(df)
    max_occurrences = df['acron'].value_counts().max()

    # Normalize frequency (60% of weight)
    frequency_weight = (occurrences / max_occurrences) * 0.6

    # Calculate specificity weight (40% of weight)
    # More specific (longer) acronyms get higher weight
    max_acron_length = df['acron'].str.len().max()
    specificity_weight = (len(acronym) / max_acron_length) * 0.4

    # Combine weights
    total_weight = frequency_weight + specificity_weight

    return min(1.0, total_weight)

def max_similaridade_texto_nao_usar(pesquisa, df):
    """Finds the text with the highest similarity score to the search text."""
    MAX_SIM = 0
    MAX_ID_TXT = None
    for index, row in df.iterrows():
        combined_text = row['combined_text']
        if isinstance(combined_text, str) and isinstance(pesquisa, str):
            similarity_score = nlp(combined_text).similarity(nlp(pesquisa))
            if similarity_score > MAX_SIM:
```

```
                MAX_SIM = similarity_score
                MAX_ID_TXT = index
    return MAX_ID_TXT, MAX_SIM


def max_similaridade_texto(pesquisa, df):
    """
    Finds the text with the highest similarity score to the search text,
normalized to 0-1.
    """
    MAX_SIM = 0
    MAX_ID_TXT = None
    for index, row in df.iterrows():
        combined_text = row['combined_text']
        if isinstance(combined_text, str) and isinstance(pesquisa, str):
            similarity_score = nlp(combined_text).similarity(nlp(pesquisa))
            if similarity_score > MAX_SIM:
                MAX_SIM = similarity_score
                MAX_ID_TXT = index
    return MAX_ID_TXT, MAX_SIM

# Function to modify acronym matching to account for weights
def max_similaridade_acronimos(ACRONIM, df):
    """
    Calculate maximum similarity score for acronyms, ensuring score is between
0 and 1
    """
    MAX_SIM = 0
    MAX_ID_ACRON = None

    for acron, indices in ACRONIM:
        # Get normalized weight (already between 0 and 1)
        acronym_weight = calculate_acronym_weight(acron, df)

        for idx in indices:
            text_to_compare = df.loc[idx, 'combined_text']

            # Calculate base similarity
            base_similarity = nlp(text_to_compare).similarity(nlp(acron))

            # Apply weighted boost, ensuring result stays ≤ 1
            # Formula: base_sim + (1 - base_sim) * weight * boost_factor
            boost_factor = 4  # Controls how much weight affects final score
            weighted_boost = (1 - base_similarity) * acronym_weight *
boost_factor
            similarity_score = base_similarity + weighted_boost
```

```
            # Extra safety check (though mathematically unnecessary)
            similarity_score = min(1.0, similarity_score)

            print(f"DEBUG: base={base_similarity:.2f},
weight={acronym_weight:.2f}, final={similarity_score:.2f}")

            if similarity_score > MAX_SIM:
                MAX_SIM = similarity_score
                MAX_ID_ACRON = idx

    return MAX_ID_ACRON, MAX_SIM

def maxima_sim_txt_ou_acron(ACR,df,txt):
    #print(f"Máxima similaridade TXT OU ACR")
    ACR_VAL = 0
    MAX_ACR = 0
    row = None
    for ind, item in enumerate(ACR):
        lin_mx_sim_acr, ACR_VAL = max_similaridade_acronimos(ACR, df)
        if lin_mx_sim_acr is not None:
            if ACR_VAL > MAX_ACR:
                row = df.loc[lin_mx_sim_acr]
                #print("Acronym", ind, item, ACR_VAL)
                MAX_ACR = ACR_VAL
    #print(f"debug text pes:{pesquisa}")
    lin_mx_sim_txt, TXT_VAL = max_similaridade_texto(txt, df)
    if lin_mx_sim_txt is not None:
        row = df.loc[lin_mx_sim_txt]
        #print(row, TXT_VAL)
        #print(f"Máxima similaridade txt: '{txt}' and row {lin_mx_sim_txt}
valor={TXT_VAL}")
        if MAX_ACR > TXT_VAL:
            #obs lin é uma linha do dataframe
            return ACR_VAL, lin_mx_sim_acr, 1
            #print(f"Acronym-based match is stronger: {MAX_ACR} > {TXT_VAL,
row}")
        else:
            #obs lin é uma linha do dataframe
            return TXT_VAL, lin_mx_sim_txt, 0

def define_acronimos_texto(texto,df):
    pot_acronimos = encontra_potacron(texto)
    pot_acronimos = [item for sublist in pot_acronimos for item in sublist]
    # Find acronym matches
    ACR = []
```

```
    for _, item in enumerate(pot_acronimos):
        a = df[df['acron'] == item]
        if not a.empty:
            indice = df[df['acron'] == item].index
            ACR.append((item, indice))
    return ACR

# USAR MESMO REGISTRO PARA DEFINIR OS PESOS
def compare_text_self(row_series, pesos):
    results = []
    # obs utilizar coluna df['processed_text']
    tema = row_series['tema'] if isinstance(row_series['tema'], str) else ""
    assunto = row_series['assunto'] if isinstance(row_series['assunto'], str)
else ""
    servico = row_series['servico'] if isinstance(row_series['servico'], str)
else ""
    head = row_series['head'] if isinstance(row_series['head'], str) else ""
    combined_text = row_series['combined_text'] if
isinstance(row_series['combined_text'], str) else ""

    ctema = nlp(tema) if tema else nlp("")
    cassunto = nlp(assunto) if assunto else nlp("")
    cservico = nlp(servico) if servico else nlp("")
    chead = nlp(head) if head else nlp("")
    ccombined_text = nlp(combined_text) if combined_text else nlp("")

    #preprocessed_text = ccombined_text
    #input_doc = nlp(preprocessed_text)
    input_doc = ccombined_text
    row_values = [str(value).strip().lower() for value in row_series.values]
    # AQUI VARRE O TEXTO DE ENTRADA
    similarity = 0
    n_tokens = 0
    max_possible_similarity = sum(pesos)  # Maximum possible weighted sum

    for token in input_doc:
        if token.text.strip().lower() in row_values:
            n_tokens += 1
            # Calculate individual similarities
            similarities = [
                (ctema.similarity(token) * pesos[0] if ctema.has_vector and
token.has_vector else 0),
                (cassunto.similarity(token) * pesos[1] if cassunto.has_vector
and token.has_vector else 0),
                (cservico.similarity(token) * pesos[2] if cservico.has_vector
and token.has_vector else 0),
```

```
                (chead.similarity(token) * pesos[3] if chead.has_vector and
token.has_vector else 0),
                (ccombined_text.similarity(token) * pesos[4] if
ccombined_text.has_vector and token.has_vector else 0)
            ]

            # Normalize by dividing by max possible similarity
            token_similarity = sum(similarities) / max_possible_similarity
            similarity += token_similarity

    # Final normalization - divide by number of matching tokens
    if n_tokens > 0:
        similarity = similarity / n_tokens

    return similarity, results


def bagofwords_df(df, column='combined_text'):
    #print(f"debug bowdf{ df[column].apply(lambda text: set([token.lemma_ for
token in nlp(text) if token.is_alpha and not token.is_stop]))}")
    df['bag_of_words'] = df[column].apply(lambda text: set([token.lemma_ for
token in nlp(text) if token.is_alpha and not token.is_stop]))
    return df

def bagofwords_txt(text):
    conjunto = set([token.lemma_ for token in nlp(text) if token.is_alpha and
not token.is_stop])
    print("BAGOFW",conjunto)
    return conjunto

def extract_subject(text):
    doc = nlp(text)
    assunto = ""
    for token in doc:
        if "subj" in token.dep_:  # Looks for subjects in dependency parsing
            assunto = assunto + " " + str(token)
            #return token.text
    return assunto #"No subject found"

def jaccard_similarity(text1, text2):
    """Calcula a similaridade de Jaccard entre dois textos."""
    words_text1 = set(text1.split())
    words_text2 = set(text2.split())
    intersection = len(words_text1.intersection(words_text2))
    union = len(words_text1.union(words_text2))
    return intersection / union if union != 0 else 0
```

```python
def extract_subject_phrases(text, nlp, max_subjects=2, min_words=3):
    """
    Extract subject phrases with minimum 3 words from text

    Args:
        text (str): Input text to analyze
        nlp: spaCy language model
        max_subjects (int): Maximum number of subjects to return (default 2)
        min_words (int): Minimum words per subject phrase (default 3)

    Returns:
        list: Up to 2 subject phrases, each with at least 3 words
    """
    doc = nlp(str(text).lower())
    subjects = []

    # Get noun chunks and their root dependencies
    subject_candidates = []
    for chunk in doc.noun_chunks:
        # Only consider phrases with at least min_words
        phrase_words = chunk.text.split()
        if len(phrase_words) < min_words:
            # Try to expand short phrases
            expanded_phrase = expand_noun_phrase(chunk, doc, min_words)
            if expanded_phrase:
                phrase_words = expanded_phrase.split()
            else:
                continue

        score = calculate_phrase_score(chunk, len(phrase_words))

        subject_candidates.append({
            'text': ' '.join(phrase_words),
            'score': score,
            'length': len(phrase_words),
            'start': chunk.start
        })

    # Sort and filter candidates
    subject_candidates.sort(key=lambda x: (x['score'], x['length']),
reverse=True)

    # Take top 2 unique subjects meeting minimum length
    seen = set()
    for candidate in subject_candidates:
```

```
        if len(subjects) >= max_subjects:
            break

        if candidate['length'] >= min_words and candidate['text'] not in seen:
            subjects.append(candidate['text'])
            seen.add(candidate['text'])

    return subjects[:max_subjects]

def expand_noun_phrase(chunk, doc, min_words):
    """Helper function to expand short noun phrases"""
    words = chunk.text.split()
    left_idx = chunk.start - 1
    right_idx = chunk.end

    # Try to expand phrase to reach min_words
    while len(words) < min_words and (left_idx >= 0 or right_idx < len(doc)):
        if left_idx >= 0 and doc[left_idx].pos_ in ['ADJ', 'NOUN', 'PROPN']:
            words.insert(0, doc[left_idx].text)
            left_idx -= 1
        elif right_idx < len(doc) and doc[right_idx].pos_ in ['ADJ', 'NOUN',
'PROPN']:
            words.append(doc[right_idx].text)
            right_idx += 1
        else:
            break

    return ' '.join(words) if len(words) >= min_words else None

def calculate_phrase_score(chunk, length):
    """Helper function to calculate phrase importance score"""
    score = 0

    # Base score from dependency
    if chunk.root.dep_ in ['nsubj', 'nsubjpass']:
        score += 5
    elif chunk.root.dep_ in ['dobj', 'pobj']:
        score += 3
    else:
        score += 1

    # Length bonus (prefer longer phrases)
    score += min(length - 2, 3)  # Cap length bonus at 3

    # Position bonus (early in text)
```

```
    score += 2.0 / (chunk.start + 1)

    return score

def extract_subject_phrases_deprecated(text, nlp, max_subjects=5):
    """
    Extract up to 3 most relevant subject phrases from text

    Args:
        text (str): Input text to analyze
        nlp: spaCy language model
        max_subjects (int): Maximum number of subjects to return (default 3)

    Returns:
        list: Up to 3 most relevant subject phrases
    """
    doc = nlp(str(text).lower())
    subjects = []

    # Get noun chunks and their root dependencies
    subject_candidates = []
    for chunk in doc.noun_chunks:
        # Score based on position and dependency
        score = 0
        if chunk.root.dep_ in ['nsubj', 'nsubjpass']:  # Main subject
            score += 3
        elif chunk.root.dep_ in ['dobj', 'pobj']:  # Object
            score += 2
        else:
            score += 1

        # Boost score for chunks near the start
        score += 1.0 / (chunk.start + 1)

        subject_candidates.append({
            'text': chunk.text,
            'score': score,
            'length': len(chunk.text.split())
        })

    # Sort by score and length (prefer longer phrases)
    subject_candidates.sort(key=lambda x: (x['score'], x['length']),
reverse=True)

    # Take top N unique subjects
```

```
    seen = set()
    for candidate in subject_candidates:
        if len(subjects) >= max_subjects:
            break

        subject = candidate['text']
        if subject not in seen and len(subject.split()) >= 1:
            subjects.append(subject)
            seen.add(subject)

    return subjects[:max_subjects]

def extract_subject_phrases2(text, nlp):
    """
    Extract meaningful subject phrases from text using dependency parsing
    and noun chunk analysis
    """
    doc = nlp(str(text).lower())
    subjects = []

    # Get noun chunks (phrases)
    noun_chunks = list(doc.noun_chunks)

    # Extract subjects from dependency parsing
    for token in doc:
        if token.dep_ in ['nsubj', 'nsubjpass']:  # Subject dependencies
            # Get the full subject phrase
            phrase = ' '.join([t.text for t in token.subtree])
            subjects.append(phrase)

    # Add noun chunks if they're not already included
    for chunk in noun_chunks:
        if chunk.text not in subjects:
            subjects.append(chunk.text)

    return subjects

def compare_subjects(input_subjects, df_subjects, threshold=0.7):
    """
    Compare extracted subjects with database entries
    Returns similarity scores and matched subjects
    """
    matches = []
    for df_subject in df_subjects:
        for input_subject in input_subjects:
            # Calculate similarity
```

```
            similarity = nlp(input_subject).similarity(nlp(df_subject))
            if similarity >= threshold:
                matches.append({
                    'input_subject': input_subject,
                    'db_subject': df_subject,
                    'score': similarity
                })
    return sorted(matches, key=lambda x: x['score'], reverse=True)
```

```python
# CLASSIFICADOR FUZZY.py
# # 19.03.2025 -- apresenta maior dificuldade na classificacao
from fuzzywuzzy import fuzz
import spacy, time
import pandas as pd
from acessoBD import inicializa_dataframe
from preprocess import encontra_potacron, lemmatize_text, lemmatize_dataframe
from asyncio import run
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

nlp = spacy.load("pt_core_news_lg")  # Load Portuguese NLP model

def extract_nouns(text):
    """Extracts nouns and proper nouns from the given text."""
    doc = nlp(text)
    nouns = [token.text for token in doc if token.pos_ in ['NOUN', 'PROPN']]
    return " ".join(nouns)

def best_match_spacy(input_text, df,
                     primary_threshold=0.6, secondary_threshold=0.5,
tertiary_threshold=0.4,
                     acronym_weight=0.3, noun_weight=0.3, tfidf_weight=0.3,
                     field_weights=None):
    """
    Finds the top 3 best matches for input_text in the dataframe.
    Uses spaCy similarity, fuzzy matching, TF-IDF, noun-based similarity, and
acronym boosts.
    """
    if field_weights is None:
        field_weights = {'tema': 0.3, 'assunto': 0.2, 'servico': 0.3, 'head':
0.2}

    # Find potential acronyms in input text
    pot_acronimos = encontra_potacron(input_text)
    pot_acronimos = [item for sublist in pot_acronimos for item in sublist]

    ACRONIM = []
    for item in pot_acronimos:
        matched_acronym = df[df['acron'] == item]
        if not matched_acronym.empty:
            indices = matched_acronym.index
            ACRONIM.append((item, indices))

    # Preprocess input text
    input_text_proc = input_text.lower().strip()
```

```
    input_doc = nlp(input_text_proc)
    input_nouns = extract_nouns(input_text_proc)

    # Build corpus for BoW (Bag of Words)
    corpus = df['combined_text'].apply(lambda x: x.lower().strip()).tolist()
    corpus.append(input_text_proc)  # Add input_text to the corpus for
comparison

    # Create the BoW model using CountVectorizer
    vectorizer = CountVectorizer()
    bow_matrix = vectorizer.fit_transform(corpus)

    # Compute cosine similarity between the input text (last row) and all
other texts
    bow_similarities = cosine_similarity(bow_matrix[-1], bow_matrix[:-1])[0]

    results = []

    # Iterate through dataframe rows
    for i, (_, row) in enumerate(df.iterrows()):
        weighted_score = 0
        total_weight = 0

        # Compare input text with each field
        for field, weight in field_weights.items():
            field_text = str(row.get(field, "")).lower().strip()
            if not field_text:
                continue

            field_doc = nlp(field_text)
            spacy_sim = input_doc.similarity(field_doc)
            fuzzy_sim = fuzz.partial_ratio(input_text_proc, field_text) /
100.0
            field_sim = (spacy_sim + fuzzy_sim) / 2.0

            # Compare nouns
            field_nouns = extract_nouns(field_text)
            noun_bonus = noun_weight * (fuzz.partial_ratio(input_nouns,
field_nouns) / 100.0) if field_nouns else 0
            field_sim += noun_bonus

            weighted_score += weight * field_sim
            total_weight += weight

        final_score = weighted_score / total_weight if total_weight > 0 else 0
```

```
        # Add BoW score (Bag of Words)
        final_score += tfidf_weight * bow_similarities[i]

        # Acronym Matching Boost
        matched_acronyms = [acr for acr, idx in ACRONIM if acr in
str(row.get('acron', "")).lower()]
        if matched_acronyms:
            final_score += acronym_weight * len(matched_acronyms)

        # Store result if above tertiary threshold
        if final_score >= tertiary_threshold:
            match_text = " ".join([str(row.get(col, "")) for col in
field_weights.keys()]).strip()
            results.append((match_text, final_score, row['nid']))

    # Sort results by score in descending order
    results.sort(key=lambda x: x[1], reverse=True)

    # Select primary, secondary, and tertiary matches
    primary_match = results[0] if len(results) > 0 and results[0][1] >=
primary_threshold else None
    secondary_match = results[1] if len(results) > 1 and results[1][1] >=
secondary_threshold else None
    tertiary_match = results[2] if len(results) > 2 else None
    #print(f"primary_match: {primary_match}")
    return primary_match, secondary_match, tertiary_match

async def program():
    startboot_time = time.time()
    df = inicializa_dataframe()
    df = lemmatize_dataframe(df, ['acron', 'tema', 'assunto', 'servico',
'head','combined_text'])

    boot_endtime = time.time()
    duracao_boot = boot_endtime - startboot_time
    print(f"\033[90m ⏱️ Duração da inicialização: {duracao_boot:.4f}
segundos\033[0m")

    while True:
        pesquisa = input("Digite o texto a ser pesquisado: ")
        pesquisa = lemmatize_text(pesquisa)

        if pesquisa.lower() == 'sair':
            break
```

```
        else:
            start_time = time.time()
            primary, secondary, tertiary = best_match_spacy(pesquisa, df)

            # Display results
            if primary:
                print(f"🥇 Primary Match: {primary[0]}, Score:
{primary[1]:.2f}, NID: {primary[2]}")
            if secondary:
                print(f"🥈 Secondary Match: {secondary[0]}, Score:
{secondary[1]:.2f}, NID: {secondary[2]}")
            if tertiary:
                print(f"🥉 Tertiary Match: {tertiary[0]}, Score:
{tertiary[1]:.2f}, NID: {tertiary[2]}")

            end_time = time.time()
            elapsed_time = end_time - start_time
            print(f"⏱️ Tempo de execução: {elapsed_time:.4f} segundos")


if __name__ == "__main__":
    run(program())
```

```python
# CLASSIFICADOR JACCARD.py
import spacy, time
from fuzzywuzzy import fuzz
import pandas as pd
from dicionario import common_words
from acessoBD import inicializa_dataframe, obter_dados_teste
from asyncio import run
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from funcoes_similaridade import jaccard_similarity
from preprocess import process_txt, lemmatize_dataframe

#nlp = spacy.load("pt_core_news_md")
nlp = spacy.load("pt_core_news_lg")

def extract_nouns(text):
    doc = nlp(text)
    nouns = [token.text for token in doc if token.pos_ in ['NOUN', 'PROPN']]
    return " ".join(nouns)

def best_match_jaccard(input_text, df, main_threshold=0.6,
secondary_threshold=0.4, tertiary_threshold=0.2,
                       acronym_weight=0.2, noun_weight=0.2, tfidf_weight=0.3,
jaccard_weight=0.3,
                       field_weights=None, acronym_bonus=0.15):
    if field_weights is None:
        field_weights = {'tema': 0.3, 'assunto': 0.2, 'servico': 0.3, 'head':
0.2}
    #print(f"debug input_text: {input_text}")
    input_text_proc = input_text.lower().strip()
    input_text_doc = nlp(input_text_proc)
    input_text_proc = " ".join([token.text for token in input_text_doc if
token.text not in common_words])

    input_nouns = extract_nouns(input_text_proc)
    acronyms_in_input = set(input_text_proc.split()) &
set(df['acron'].dropna().unique())

    corpus = df['combined_text'].fillna("").str.lower().tolist()

    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(corpus)
    input_tfidf = vectorizer.transform([input_text_proc])
    tfidf_similarities = cosine_similarity(input_tfidf, tfidf_matrix)[0]

    best_match, best_score, best_nid = None, 0, None
```

```
    secondary_match, secondary_score = None, 0
    tertiary_match, tertiary_score = None, 0

    for i, (_, row) in enumerate(df.iterrows()):
        weighted_score = 0
        total_weight = 0

        for field, weight in field_weights.items():
            field_text = str(row.get(field, "")).lower().strip()
            if not field_text:
                continue

            field_doc = nlp(field_text)

            if field_doc.has_vector and input_text_doc.has_vector:
                spacy_sim = input_text_doc.similarity(field_doc)
            else:
                # Fallback to fuzzy matching if no vectors available
                spacy_sim = 0.0

            fuzzy_sim = fuzz.partial_ratio(input_text_proc, field_text) /
100.0
            #field_sim = (spacy_sim + fuzzy_sim) / 2.0
            field_nouns = extract_nouns(field_text)
            noun_bonus = noun_weight * fuzz.partial_ratio(input_nouns,
field_nouns) / 100.0 if field_nouns else 0

            # Weight fuzzy matching more heavily if no vectors available
            if not field_doc.has_vector or not input_text_doc.has_vector:
                field_sim = fuzzy_sim +(1-fuzzy_sim)*noun_bonus
            else:
                field_sim = (spacy_sim + fuzzy_sim) / 2.0

            weighted_score += weight * field_sim
            total_weight += weight

        final_score = weighted_score / total_weight if total_weight > 0 else 0
        tfidf_bonus = tfidf_weight * tfidf_similarities[i]
        #final_score += tfidf_bonus
        final_score = final_score + (1- final_score)*tfidf_bonus

        jaccard_sim = jaccard_similarity(input_text_proc, corpus[i])
        final_score += jaccard_weight * jaccard_sim

        acronyms_in_row = set(str(row.get('acron', "")).split())
```

```
        acronym_match_bonus = acronym_bonus if acronyms_in_input &
acronyms_in_row else 0
        #final_score += acronym_match_bonus
        final_score = final_score + (1-final_score)*acronym_match_bonus

        if final_score >= main_threshold and final_score > best_score:
            best_match = row['combined_text']
            best_score = final_score
            best_nid = row['nid']
        elif final_score >= secondary_threshold and final_score >
secondary_score:
            secondary_match = row['assunto']
            secondary_score = final_score
        elif final_score >= tertiary_threshold and final_score >
tertiary_score:
            tertiary_match = row['tema']
            tertiary_score = final_score

    return best_match, best_score, best_nid, secondary_match, secondary_score,
tertiary_match, tertiary_score

async def program():
    startboot_time = time.time()
    df = inicializa_dataframe()
    ## OBS VERIFICAR SE NAO PIOROU O SCORE:
    #df = lemmatize_dataframe(df, ['acron', 'tema', 'assunto', 'servico',
'head', 'combined_text'])
    df_lem = lemmatize_dataframe(df, ['acron', 'tema', 'assunto', 'servico',
'head','combined_text'])
    df_test = obter_dados_teste()
    boot_endtime = time.time()
    duracao_boot = boot_endtime - startboot_time
    print(f"\033[31m ⏱️ Duração da inicialização(JACCARD): {duracao_boot:.4f}
segundos\033[0m")
    while True:
        pesquisa = input("Digite o texto a ser pesquisado, '1' para testar com
dados do arquivo: ▨")
        if pesquisa == '1':
            start_time = time.time()
            for indice, linha in df_test.iterrows():
                text_value = linha['Text']
                match, score, nid, sec_match, sec_score, ter_match, ter_score
= best_match_jaccard(text_value, df)
                matchL, scoreL, nidL, sec_matchL, sec_scoreL, ter_matchL,
ter_scoreL = best_match_jaccard(text_value, df_lem)
```

```
                #print("Best Match\tScore\tNID\tSecondary Match\tSec.
Score\tTertiary Match\tTer. Score")  # Header row
                print(f"S/L:
{match}\t{score:.2f}\t{nid}\t{sec_match}\t{sec_score:.2f}\t{ter_match}\t{ter_s
core:.2f}")
                print(f"C/L:
{matchL}\t{score:.2f}\t{nidL}\t{sec_matchL}\t{sec_scoreL:.2f}\t{ter_matchL}\t{
ter_scoreL:.2f}")
                with open("saida_sproc_JACCARD.txt", "a", encoding="utf-8") as
f:  # Open file in append mode

f.write(f"{match}\t{score:.2f}\t{nid}\t{sec_match}\t{sec_score:.2f}\t{ter_matc
h}\t{ter_score:.2f}\n")
                nada = """
                if match:
                    print(f"🥇 Best Match (Combined Text): {match}, Score:
{score:.2f}, NID: {nid}")
                if sec_match:
                    print(f"🥈 Secondary Match (Assunto): {sec_match}, Score:
{sec_score:.2f}")
                if ter_match:
                    print(f"🥉 Tertiary Match (Tema): {ter_match}, Score:
{ter_score:.2f}")" \
                """
                texto = process_txt(text_value)


        else:
            if pesquisa.lower() == 'sair':
                break
            else:
                # SEM LEMATIZACAO
                start_time = time.time()
                match, score, nid, sec_match, sec_score, ter_match, ter_score
= best_match_jaccard(pesquisa, df)
                if match:
                    print(f"🥇 Best Match (Combined Text): {match}, Score:
{score:.2f}, NID: {nid}")
                if sec_match:
                    print(f"🥈 Secondary Match (Assunto): {sec_match}, Score:
{sec_score:.2f}")
                if ter_match:
                    print(f"🥉 Tertiary Match (Tema): {ter_match}, Score:
{ter_score:.2f}")
                end_time = time.time()
                elapsed_time = end_time - start_time
                print(f"⏱️ Tempo de execução: {elapsed_time:.4f} segundos")
```

```
                # COM LEMATIZACAO
                texto = process_txt(pesquisa)
                start_time = time.time()
                match, score, nid, sec_match, sec_score, ter_match, ter_score
= best_match_jaccard(texto, df_lem)
                if match:
                    print(f"🥇 Best Match (Combined Text): {match}, Score:
{score:.2f}, NID: {nid}")
                if sec_match:
                    print(f"🥈 Secondary Match (Assunto): {sec_match}, Score:
{sec_score:.2f}")
                if ter_match:
                    print(f"🥉 Tertiary Match (Tema): {ter_match}, Score:
{ter_score:.2f}")
                end_time = time.time()
                elapsed_time = end_time - start_time
                print(f"⏱️ Tempo de execução: {elapsed_time:.4f} segundos")

if __name__ == "__main__":
    run(program())
```

```python
# CLASSIFICADOR LEITOR_NOMES.py -- PROGRAMA PRINCIPAL

import os, re
import time
import spacy
import win32com.client
import warnings
import pandas as pd
import numpy as np
from tqdm import trange
from asyncio import run
from FUZZY import best_match_spacy
from JACCARD import best_match_jaccard
from openpyxl.utils.exceptions import InvalidFileException
from C_SCIKIT import class_scikit
from dicionario import extrair_pcomuns
from preprocess import lemmatize_dataframe,
preprocess_text_no_stopwords,remove_det_prep_punct
from funcoes_similaridade import define_acronimos_texto,
maxima_sim_txt_ou_acron, bagofwords_df, extract_subject_phrases ,
compare_subjects
from acessoBD import inicializa_dataframe, obter_dados_teste
from EXCEL import
load_excel_file,safe_excel_write,resolve_shortcut,load_workbook

def extract_nouns(text, nlp):
    """Extract nouns from text using spaCy"""
    text = str(text).lower().strip()
    text = text.replace(' - ', ' ').replace('-', ' ')
    # Process with spaCy
    doc = nlp(text)
    # Extract nouns with filtering
    nouns = [
        token.text for token in doc
        if token.pos_ in ['NOUN', 'PROPN']
        and token.text != '-'
        and len(token.text.strip()) > 1
    ]
    return nouns

def call_funcoes(df_sc,dfcls, dfclsbow, df_lem, actual_file_path, nlp):
    algo_results = {}
    table_name = "COLETA"
    nome_plan= "OUVIDORIA"
    dfcls['nouns'] = dfcls['combined_text'].apply(lambda x: extract_nouns(x,
nlp))
    #
```

```
    #for i1,linha1 in dfcls.iterrows():
    #    print("DEBUG dfcls", linha1['nouns'],linha1['nid'])
    # DEBUG dfcls ['trânsito', 'transporte', 'ponto', 'emergencial', 'paese']
21.16.5
    nada = """...
    DEBUG dfcls ['trânsito', 'transporte', 'ponto', 'ônibus', 'catraca',
'validador', 'terminal', 'recarga', 'falha'] 21.16.6
    DEBUG dfcls ['trânsito', 'transporte', 'ponto', 'ônibus', 'limpeza',
'higienização'] 21.16.7
    DEBUG dfcls ['trânsito', 'transporte', 'ponto', 'ônibus', 'reparo',
'problema'] 21.16.8
    DEBUG dfcls ['trânsito', 'transporte', 'ponto', 'ônibus', 'empresa']
21.16.9
    DEBUG dfcls ['trânsito', 'transporte', 'ponto', 'ônibus', 'implantação',
'ponto', 'abrigo'] 21.16.10
    DEBUG dfcls ['trânsito', 'transporte', 'ponto', 'ônibus', 'mudança',
'ponto', 'abrigo'] 21.16.11
    DEBUG dfcls ['trânsito', 'transporte', 'ponto', 'ônibus', 'ponto',
'ônibus', 'mudança', 'ponto', 'abrigo'] 21.16.12
    DEBUG dfcls ['trânsito', 'transporte', 'ponto', 'ônibus', 'ponto',
'ônibus', 'implantação', 'ponto', 'abrigo'] 21.16.13
    DEBUG dfcls ['trânsito', 'transporte', 'ponto', 'ônibus', 'ponto',
'ônibus', 'manutenção', 'conserto', 'limpeza', 'equipamento'] 21.16.14 ... """

    # Load Excel file once
    workbook, planilha, df_excel = load_excel_file(actual_file_path,
nome_plan)
    if workbook is None:
        print("Failed to load Excel file")
        return

    df_excel = pd.read_excel(actual_file_path)
    for indice, item in df_excel.iterrows():
        texto = str(item.iloc[4])
        print(f"\nCONSULTA: {texto}")
        nomes_notexto = extract_nouns(texto,nlp)
        #print(f"DEBUG nomes no texto{nomes_notexto}")
        ACRONIM = define_acronimos_texto(texto, dfcls)
        #texto=texto.lower()
        #texto=lemmatize_text(str(item.iloc[4]))

        texto=extrair_pcomuns(texto)
        #texto = remove_psig_acron(texto)
        txt_semstop = preprocess_text_no_stopwords(texto)
        #print("debug ASSUNTO",txt_semstop)
        #txt_lemat = lemmatize_text(txt_semstop)
        sem_numeros = re.sub(r'\d+', '', txt_semstop)
```

```
        text_semprep = remove_det_prep_punct(sem_numeros)
        texto = text_semprep.lower().strip()

        #texto = process_txt(texto) if isinstance(texto, str) else texto
        #if not texto:
        #    print(f"Warning: Empty processed text at row {indice + 2}")
        #    continue



        # Extract subjects from input text
        input_subjects = extract_subject_phrases(texto, nlp)
        print(f"\n🔍 Extracted subjects: {input_subjects}")

        best_matches = {}  # Track unique matches
        best_match_info = {
            'row': None,
            'nid': None,
            'tema' :None,
            'assunto' : None,
            'servico' : None,
            'score': 0
        }

        for indice, row in dfcls.iterrows():
            matches = compare_subjects(input_subjects, row['bag_of_words'])
            if matches:
                for match in matches:
                    match_key = (match['input_subject'], match['db_subject'])
                    if match_key not in best_matches or match['score'] >
best_matches[match_key]['score']:
                        best_matches[match_key] = match
                        # Update best match info if score is higher
                        if match['score'] > best_match_info['score']:
                            best_match_info['row'] = row
                            best_match_info['nid'] = row['nid']
                            best_match_info['score'] = match['score']
                            best_match_info['tema'] = row['tema']
                            best_match_info['assunto'] = row['assunto']
                            best_match_info['servico'] = row['assunto']

        # Convert dictionary values to list and sort
        best_matches_list = sorted(
            best_matches.values(),
            key=lambda x: x['score'],
            reverse=True
        )
```

```
        if best_matches_list:
            print("\n📊 Top subject matches:")
            print(f"\n🎯 Best matching row NID: {best_match_info['nid']}")
            print(f"Best matching row data:
{best_match_info['row']['servico']}")
            print(f"Best match score: {best_match_info['score']:.2f}")
            for match in best_matches_list[:5]:
                print(f"Input: {match['input_subject']}")
                print(f"Match: {match['db_subject']}")
                print(f"Score: {match['score']:.2f}\n")

        #print(f"\033[31m ⏱ debug SUBJS {primary}\033[0m")
        # TESTAR SCIKIT
        #sc_tema, sc_assunto, sc_servico, sc_nid, sc_val =
class_scikit(nomes_notexto, df_sc)
        #NOMES // outro df
        sc_tema, sc_assunto, sc_servico, sc_nid, sc_val =
class_scikit(nomes_notexto, dfcls,1)

        # TESTAR FUZZY
        primary, secondary, tertiary = best_match_spacy(texto, df_lem)
        #print("debug FUZZY",primary, secondary, tertiary)
        print(f"\033[36m ⏱ debug FUZZY {primary}\033[0m")
        #primary, secondary, tertiary = best_match_spacy(texto, dfcls)  # Pass
dfcls here
        # TESTAR JACCARD
        match, score, nid, sec_match, sec_score, ter_match, ter_score =
best_match_jaccard(texto, df_lem)
        #print("debug JACCARD",match, score, nid)
        print(f"\033[34m ⏱ debug JACCARD {match, score, nid}\033[0m")
        #match, score, nid, sec_match, sec_score, ter_match, ter_score =
best_match_jaccard(texto, dfcls)
        # TESTAR # #BAG OF WORDS
        VALOR_FC, LINHA_DF, ACR_OU_NAO = maxima_sim_txt_ou_acron(ACRONIM,
df_lem, texto)
        LIN = dfcls.loc[LINHA_DF]
        print(f"\033[33m ⏱ debug BOW  {LIN['servico'],LIN['nid'] ,VALOR_FC,
ACR_OU_NAO}\033[0m")
        #VALOR_FC, LINHA_DF, ACR_OU_NAO = maxima_sim_txt_ou_acron(ACRONIM,
dfcls, texto)

        # TESTAR C-MEANS -- resultados ruins 31/03/2025
        #print("TESTE DE CMEANS",best_match_cmeans(texto), dfcls)) # O
PREPROCESSAMENTO DO TEXTO PODE NÃO FUNCIONAR EM CMEANS
        #print("TESTE DE CMEANS",best_match_cmeans(nomes_notexto, dfcls,nlp))
```

```
        # SALVAR A PLANILHA
        try:
            if isinstance(LINHA_DF, int):
                LIN = dfcls.loc[LINHA_DF]
                # Ensure VALOR_FC is a valid number
                try:
                    #c1 = max(float(primary[1]), float(score),
float(VALOR_FC),float(sc_val))
                    #c1 = max(float(primary[1]), float(score),
float(VALOR_FC))
                    c1 = max(float(primary[1]), float(score), float(VALOR_FC),
float(best_match_info['score']))
                    print("DEBUG C1",float(primary[1]), float(score),
float(VALOR_FC),float(best_match_info['score']))
                except ValueError as e:
                    print(f"ValueError converting scores: {e}")
                    c1 = float(primary[1])  # Default to Fuzzy score if others
fail
                if c1 == (float(primary[1])):
                    print(f"🍎 Best Match FUZZY Score: {primary[1]:.2f};
{primary[2]};{primary[0]}")
                    planilha.cell(indice+2, 6,  f"{primary[0]}")
                    planilha.cell(indice+2, 7, primary[2])
                    planilha.cell(indice+2, 8, f"{primary[1]:.2f}")
                    planilha.cell(indice+2, 9,  secondary[0])
                    planilha.cell(indice+2, 10,  tertiary[0])
                    planilha.cell(indice+2, 11, "FUZZY")

                elif c1 == (float(score)):
                    print(f"🍎 Best Match JACCARD Score:
{score:.2f};{nid};{match}")
                    planilha.cell(indice+2, 6,  f"{match}")
                    planilha.cell(indice+2, 7,  nid)
                    planilha.cell(indice+2, 8,  f"{score:.2f}")
                    planilha.cell(indice+2, 9,  sec_match)
                    planilha.cell(indice+2, 10,  ter_match)
                    planilha.cell(indice+2, 11,"JACCARD")

                elif c1 == (float(VALOR_FC)):
                    print(f"🍎 Best Match BAG OF WORDS Score:
{VALOR_FC};{LIN['nid']};{LIN['servico']}")
                    planilha.cell(indice+2, 6,  f"{LIN['servico']}")
                    planilha.cell(indice+2, 7,   LIN['nid'])
                    planilha.cell(indice+2, 8,   f"{VALOR_FC:.2f}")
                    planilha.cell(indice+2, 9,LIN['assunto'])
```

```
                    planilha.cell(indice+2, 10,LIN['tema'])
                    planilha.cell(indice+2, 11,"BOW")

                elif c1 == (float(best_match_info['score'])):
                    print(f"🍒 Best Match ASSUNTOS Score:
{best_match_info['score']}")

safe_excel_write(planilha,indice+2,6,best_match_info['servico'])

safe_excel_write(planilha,indice+2,7,best_match_info['nid'])

safe_excel_write(planilha,indice+2,8,best_match_info['score'])

safe_excel_write(planilha,indice+2,9,best_match_info['assunto'])

safe_excel_write(planilha,indice+2,10,best_match_info['tema'])
                    safe_excel_write(planilha,indice+2,11,'ASSUNTO (variante
de BOW)')

                else:
                    print("No best match found. Adicionando resultados do
SCIKIT")
                    safe_excel_write(planilha, indice+2, 6, sc_servico)
                    safe_excel_write(planilha, indice+2, 7, sc_nid)
                    safe_excel_write(planilha, indice+2, 8, sc_val)
                    safe_excel_write(planilha, indice+2, 9, sc_assunto)
                    safe_excel_write(planilha, indice+2, 10, sc_tema)
                    safe_excel_write(planilha, indice+2, 11, "SCIKIT")

                # GRAVAR O EXCEL
                try:
                    workbook.save(actual_file_path)
                    print("Data added successfully.")
                except InvalidFileException as e:
                    if "Registros Reparados" in str(e): #check the error
message
                        print("Warning: Excel file repaired during save.
Please verify the table.")
                        workbook = load_workbook(actual_file_path) #Reload the
file to avoid further issues.
                        workbook.save(actual_file_path + "_repaired.xlsx") #
save the repaired file with a new name.
                        print(f"Repaired file saved to
{actual_file_path}_repaired.xlsx")
                    else:
                        print(f"An error occurred during save: {e}")
```

```
                except Exception as e:
                    print(f"An error occurred during save: {e}")
            else:
                print(f"LINHA_DF is not an integer: {LINHA_DF}")
            # Write back to the same sheet **without overwriting headers**
            #df_excel.to_excel(writer, sheet_name=sheet_name, index=True,
header=False, startrow=1)
        except KeyError as e:
            print(f"KeyError accessing LIN: {e}")
        except Exception as erro:
            print(f"General error in inner loop: {erro}")
    # Save the DataFrame back to the Excel file
    #df_excel.to_excel(actual_file_path, index=False)

async def main(dicion_size, flag_boot, actual_file_path):
    """Função principal que gerencia a inicialização e a classificação."""
    nlp = spacy.load("pt_core_news_lg" if dicion_size == "1" else
"pt_core_news_md")
    df_cls = inicializa_dataframe()
    df_clsbow = bagofwords_df(df_cls)
    dados=[]
    for i, row in df_cls.iterrows():
        dados.append([row['tema'], row['assunto'], row['servico'],row['nid']])
        # Create DataFrame
        df_sc = pd.DataFrame(dados, columns=["tema", "assunto",
"servico","nid"])

    df_lem = lemmatize_dataframe(df_cls, ['acron', 'tema', 'assunto',
'servico', 'head', 'combined_text'])
    print(f"call_funcoes")
    call_funcoes(df_sc,df_cls, df_clsbow, df_lem, actual_file_path, nlp)

if __name__ == "__main__":
    print(f"LEITOR")
    shortcut_path = r"PROTOTIPO_OUVIDORIA.lnk"
    actual_file_path = resolve_shortcut(shortcut_path)
    flag_boot = 1
    while True:
        #dicion_size = input("Defina o dicionário ptbr 1-LARGE 2-MEDIUM\n")
        #run(main(dicion_size, flag_boot, actual_file_path))
        run(main(1, flag_boot, actual_file_path))
        if input("Continuar (S/N)").strip().lower() != 's':
            break
```

```python
# CLASSIFICADOR preprocess.py
# FUNCOES GENÉRICAS DE PREPROCESSAMENTO DE TEXTO
# 21/03/2025 SMA

import spacy
import re

nlp = spacy.load("pt_core_news_md")

def consecutivos(text):
    """
    Finds consecutive uppercase letters (potential acronyms).
    """
    pattern = r'\b[A-Z]{2,}\.?[\b]*'
    result = re.findall(pattern, text)
    return result

def encontra_potacron(text):
    """
    Finds potential acronyms in the given text using regex for consecutive
uppercase letters,
    and returns a list of unique acronyms.
    """
    acronyms_in_text = []
    doc = nlp(text)
    for token in doc:
        possAcron = consecutivos(token.text)  # Use token.text instead of
token
        if possAcron and possAcron not in acronyms_in_text:
            acronyms_in_text.append(possAcron)
    return acronyms_in_text

def define_head(texto):
    """Defines the head of the text (usually the syntactic head)."""
    doc = nlp(texto)
    if doc:
        return doc[0].head.text
    return None

def remove_strings(text):
    """Removes strings with repeated characters."""
    if isinstance(text, str):
        text = re.sub(r'\b([a-zA-Z0-9])\1{3,}\b', '', text)  # Remove words
with 4 or more repeating characters
        return text
    else:
```

```
        raise TypeError(f"Expected a string or bytes-like object, got
'{type(text).__name__}'")

def preprocess_text(text, acronym_dict):
    """Preprocesses the text by lemmatizing and replacing acronyms."""
    doc = nlp(text)
    tokens = []
    for token in doc:
        if token.text in acronym_dict:
            tokens.append(acronym_dict[token.text])
        elif not token.is_stop and not token.is_punct:
            tokens.append(token.lemma_)
    return ' '.join(tokens)

def preprocess_text_no_stopwords(text):
    """Preprocesses the text removing stopwords and punctuation."""
    doc = nlp(text)
    tokens = [token.text for token in doc if not token.is_stop and not
token.is_punct]
    return ' '.join(tokens)

def remove_psig_acron(palavra):
    """Removes words without meaning (potential acronyms)."""
    doc = nlp(palavra)
    token = doc[0]
    if token.vector_norm < 50:  # Consider this a word without meaning
        return False
    else:
        return True

def atualiza_dicionario_acronimos(textos, acronym_dict):
    """Updates the acronym dictionary with potential acronyms and their
contexts."""
    for text in textos:
        palavras = text.split()
        ha_acron = False
        for i, palavra in enumerate(palavras):
            bool_sign = remove_psig_acron(palavra)
            if re.match(r"[A-Z]{2,}", palavra) and not bool_sign:
                ha_acron = True
                prev_words = palavras[max(0, i-3):i]
                next_words = palavras[i+1:i+4]
                filtered_next_words = [w for w in next_words if
w.startswith(palavra[0])]
                prev_words = [""] * (3 - len(prev_words)) + prev_words
                next_words += [""] * (3 - len(next_words))
```

```
                context = {'prev_words': prev_words, 'next_words': next_words}
                if palavra in acronym_dict:
                    acronym_dict[palavra]['contexts'].append(context)
                else:
                    acronym_dict[palavra] = {'contexts': [context]}
    return acronym_dict

def remove_det_prep_punct(text):
    """
    Removes determiners, prepositions, punctuation and numbers except 250,
156, and 2.

    Args:
        text (str): Input text to process

    Returns:
        str: Processed text with unwanted elements removed
    """
    text_rem = nlp(text)
    texto = ""

    # Define allowed numbers
    allowed_numbers = {'250', '156', '2'}

    for tokens in text_rem:
        # Check if token is an allowed number
        if tokens.text in allowed_numbers:
            texto = texto + " " + tokens.text
        # Check other tokens
        elif tokens.dep_ != "det" and tokens.dep_ !="prep" and tokens.dep_ !=
"punct":
            texto = texto + " " + str(tokens)

    if isinstance(texto, str):
        # Remove unwanted characters
        texto = re.sub(r"[/'-]", "", texto)
        # Remove all numbers except allowed ones
        texto = re.sub(r'\b(?!250\b)(?!156\b)(?!2\b)\d+\b', '', texto)
        # Clean up extra spaces
        texto = re.sub(r'\s+', ' ', texto).strip()

    return texto

def remove_det_prep_punct_naousar(text):
    text_rem = nlp(text)
    texto = ""
```

```
    for tokens in text_rem:
        if (tokens.dep_ != "det") & (tokens.dep_ !="prep") & (tokens.dep_ !=
"punct"):
            texto = texto + " " + str(tokens)
        if isinstance(texto, str):
        # Removing specific unwanted characters like '/', '-', and "'"
            texto = re.sub(r"[/'-]", "", texto)
    return texto

def lemmatize_text(text):
    """Lemmatizes the given text using spaCy."""
    doc = nlp(text)
    return " ".join([token.lemma_ for token in doc])

def lemmatize_dataframe(df, columns):
    """Applies lemmatization to specified columns in the dataframe."""
    for col in columns:
        if col in df.columns:
            df[col] = df[col].astype(str).apply(lemmatize_text)
    return df



# TODAS AS FUNCOES DE PROCESSAMENTO
def process_txt(texto):
    texto = remove_psig_acron(texto)
    txt_semstop = preprocess_text_no_stopwords(texto)
    #print("debug ASSUNTO",txt_semstop)
    txt_lemat = lemmatize_text(txt_semstop)
    sem_numeros = re.sub(r'\d+', '', txt_lemat)
    text_semprep = remove_det_prep_punct(sem_numeros)
    final = text_semprep.lower().strip()
    #print("debug FINAL",final)
    return final

# Utilizar a funcao reduzir_texto
def extract_nouns(text):
    doc = nlp(text)
    nouns = [token.text for token in doc if token.pos_ in ['NOUN', 'PROPN']]
    return " ".join(nouns)



# ESPECIFICAMENTE ESTA FUNCAO PROCURA NOMES E ASSUNTOS
def reduzir_texto(text):
    doc = nlp(text)
    subjects = [token.text for token in doc if token.dep_ in ["nsubj",
"nsubj_pass"]]
```

```
heads = [token.head.text for token in doc]
nouns = [token.text for token in doc if token.pos_ in ["NOUN", "PROPN"]]
reduced_text = list(set(subjects + heads + nouns))  # Remove duplicates
return " ".join(reduced_text)
```

```python
# CLASSIFICADOR C_SCIKIT.py

import pandas as pd
import numpy as np
from acessoBD import inicializa_dataframe
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import classification_report,accuracy_score
from sklearn.model_selection import train_test_split

# BETA -- MODIFICAR PARA CONFIGURAR UMA SÓ VEZ -- mover para inicializacao
prog principal
def class_scikit(txt_entrada,df,nomes):

    # CONCATENA EVENTUAIS LISTAS
    if isinstance(txt_entrada, list):
        txt_entrada = ' '.join(txt_entrada)

    # Initialize vectorizer and create TF-IDF features
    vectorizer = TfidfVectorizer()
    # Prepare training data
    if nomes == 1:
        # Convert lists in nouns column to strings
        train_data = df["nouns"].apply(lambda x: ' '.join(x) if isinstance(x,
list) else str(x))
        X = vectorizer.fit_transform(train_data)
    else:
        X = vectorizer.fit_transform(df["servico"])

    # Create three separate classifiers
    clf_tema = SVC(kernel="linear")
    clf_assunto = SVC(kernel="linear")
    clf_servico = SVC(kernel="linear")

    # Train all classifiers
    clf_tema.fit(X, df["tema"])
    clf_assunto.fit(X, df["assunto"])
    clf_servico.fit(X, df["servico"])
    servico_to_nid = dict(zip(df['servico'], df['nid']))
    #single_text = txt_entrada
    X_single = vectorizer.transform([txt_entrada])  # Wrap text in list

    # Make single prediction
    prediction_tema = clf_tema.predict(X_single)
    prediction_assunto = clf_assunto.predict(X_single)
    prediction_servico = clf_servico.predict(X_single)
```

```
    # Get confidence scores for single prediction
    decision_score_tema = clf_tema.decision_function(X_single)
    decision_score_assunto = clf_assunto.decision_function(X_single)
    decision_score_servico = clf_servico.decision_function(X_single)
    servico_nid = servico_to_nid.get(prediction_servico[0], 'Not found')

    # Calculate confidence scores
    tema_confidence = 1 / (1 + np.exp(-np.abs(decision_score_tema[0].max())))
    assunto_confidence = 1 / (1 + np.exp(-
np.abs(decision_score_assunto[0].max())))
    servico_confidence = 1 / (1 + np.exp(-
np.abs(decision_score_servico[0].max())))

    return prediction_tema, prediction_assunto, prediction_servico,
servico_nid, servico_confidence
```

Apêndice 1.4: Código-fonte em HTML com VBScript da interface integrada de acesso à união das bases de dados “LINDÃO” e “BDI”

```
<html xmlns:v="urn:schemas-microsoft-com:vml"
xmlns:o="urn:schemas-microsoft-com:office:office"
xmlns:w="urn:schemas-microsoft-com:office:word"
xmlns:m="http://schemas.microsoft.com/office/2004/12/omml"
xmlns="http://www.w3.org/TR/REC-html40">

<head>
<style>
        .red-text {
            color: red;
        }
        .blue-text {
            color: blue;
        }
        .green-text {
            color: green;
        }
</style>
<meta http-equiv=Content-Type content="text/html; charset=windows-1252">
<meta name=ProgId content=Word.Document>
<meta name=Generator content="Microsoft Word 14">
<meta name=Originator content="Microsoft Word 14">
<link rel=File-List href="inicio_arquivos/filelist.xml">
<link rel=Edit-Time-Data href="inicio_arquivos/editdata.mso">
<!--[if !mso]>
<style>
v\:* {behavior:url(#default#VML);}
o\:* {behavior:url(#default#VML);}
w\:* {behavior:url(#default#VML);}
.shape {behavior:url(#default#VML);}
</style>
<![endif]--><!--[if gte mso 9]><xml>
 <o:DocumentProperties>
  <o:Author>Silvio Miyadaira Amâncio</o:Author>
  <o:Template>Normal</o:Template>
  <o:LastAuthor>Silvio Miyadaira Amâncio</o:LastAuthor>
  <o:Revision>2</o:Revision>
  <o:TotalTime>3</o:TotalTime>
  <o:Created>2024-08-21T11:20:00Z</o:Created>
  <o:LastSaved>2024-08-21T11:20:00Z</o:LastSaved>
  <o:Pages>1</o:Pages>
  <o:Words>11</o:Words>
  <o:Characters>63</o:Characters>
  <o:Lines>1</o:Lines>
  <o:Paragraphs>1</o:Paragraphs>
  <o:CharactersWithSpaces>73</o:CharactersWithSpaces>
  <o:Version>14.00</o:Version>
 </o:DocumentProperties>
 <o:OfficeDocumentSettings>
  <o:AllowPNG/>
 </o:OfficeDocumentSettings>
</xml><![endif]-->
<link rel=dataStoreItem href="inicio_arquivos/item0007.xml"
target="inicio_arquivos/props008.xml">
```

```
<v:imagedata src="logo.png" o:title=""/>
</v:shape><![endif]--><![if !vml]><img width=182 height=144 src="logo.png"
v:shapes="Imagem_x0020_1"><![endif]></span></p>

<p class=MsoNormal><o:p> </o:p></p>

<p class=MsoNormal align=center style='text-align:center'><b
style='mso-bidi-font-weight:
normal'><span style='font-size:18.0pt;line-height:115%'>Pagina Inicial Consulta
Dupla<o:p></o:p></span></b></p>



<title>BDI BETA</title>

<style>
    table {
        width: 100%;
        border-collapse: collapse;
    }
    table, th, td {
        border: 2px solid black;
    }
    th, td {
        padding: 8px;
        text-align: center;
    }
    th {
        cursor: pointer;
    }
</style>
<script type="text/vbscript">
    'a corrigir 09/2024 -- ordenar tabela; -- campos trocados;
    'corrigidos;
    'retornar todos os registros;

        dim fso: set fso = CreateObject("Scripting.FileSystemObject")
        dim path: path = fso.GetAbsolutePathName(".")
        dim oRst, orst2, sConnect, StrQuery, chkExato,StrQuery2,tipotxt
        set oRst = CreateObject("ADODB.recordset")
        sConnect = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source="& path &
"\bdi.mdb;Persist Security Info=False"
        On Error Resume Next

        sub fcarrega()
            '27/08 bd esta travando por tamanho'
            document.getElementById("checkBoxLimit").checked=1
        end sub

        Sub BotaoA()
            cleartable()
            tipotxt=verificatxtbox()
            if tipotxt<>1 then
                msgbox "Insira um numero"
```

```
                exit sub
            else
                if document.getElementById("checkBoxLimit").checked then

                    strquery="Select top 100 * from TABELA_GERAL where
[TABELA_GERAL].[ASSUNTO]=" & CLng(document.getElementById("txtPesq").value)  &
";"
                else
                'limitar; verificar posteriormente outras solucoes
                    strquery="Select top 250 * from TABELA_GERAL where
[TABELA_GERAL].[ASSUNTO]=" & CLng(document.getElementById("txtPesq").value)  &
";"
                end if
                writeTable(StrQuery)
            end if
        End Sub

        sub BotaoALVARA()
            dim nregs
            nregs=qtRegistros()
            tipotxt=verificatxtbox()
            cleartable()
            if tipotxt<>1 then
                msgbox "insira um numero"
                exit sub
            else
                if document.getElementById("checkBoxLimit").checked then
                    strQuery=cstr("Select top 100 * from TUDO where
[TUDO].[ALVARA_NUMERO]=" & CLng(document.getElementById("txtPesq").value)  &
";")
                else
                    strQuery=cstr("Select top 250 * from TUDO where
[TUDO].[ALVARA_NUMERO]=" &  CLng(document.getElementById("txtPesq").value)  &
";")
                 end if
                writeTable2(strquery)
            end if
        end sub

        Sub BotaoDd()
            cleartable()
            if tipotxt<>2 then
                msgbox "Insira uma data"
                exit sub
            else
                if document.getElementById("checkBoxLimit").checked then
                    strquery="Select TOP 100 * from TABELA_GERAL where
[TABELA_GERAL].[DATAd]LIKE'%" & CSTR(document.getElementById("txtPesq").value)
& "%';"
                    strquery2="Select TOP 100 * from TUDO where
[TUDO].[DATAd]LIKE'%" & CSTR(document.getElementById("txtPesq").value)  & "%';"
                else
                    strquery="Select  top 250  * from TABELA_GERAL where
[TABELA_GERAL].[DATAd]LIKE'%" & CSTR(document.getElementById("txtPesq").value)
```

```
& "%';"
                    strquery2="Select  top 250   * from TUDO where
[TUDO].[DATAd]LIKE'%" & CSTR(document.getElementById("txtPesq").value)  & "%';"
                end if
                writeTable(StrQuery)
                writeTable2(strQuery2)
            end if
        end sub

        Sub BotaoDr()
            cleartable()
            tipotxt=verificatxtbox()
            if tipotxt<>2 then
                msgbox "Insira uma data"
                exit sub
            else
                if document.getElementById("checkBoxLimit").checked then
                    strquery="Select  top 100   * from TABELA_GERAL where
[TABELA_GERAL].[DATAr]LIKE'%" & CSTR(document.getElementById("txtPesq").value)
& "%';"
                else
                    strquery="Select   top 250  * from TABELA_GERAL where
[TABELA_GERAL].[DATAr]LIKE'%" & CSTR(document.getElementById("txtPesq").value)
& "%';"
                end if
                writeTable(StrQuery)
            end if
        end sub

        Sub BotaoD()
            cleartable()
            tipotxt=verificatxtbox()
            if tipotxt<>2 then
                msgbox "Insira uma data"
                exit sub
            else
                if document.getElementById("checkBoxLimit").checked then
                    strquery="Select   top 100  * from TABELA_GERAL where
[TABELA_GERAL].[DESPACHO]LIKE'%" &
CSTR(document.getElementById("txtPesq").value)  & "%';"
                else
                    strquery="Select   top 250  * from TABELA_GERAL where
[TABELA_GERAL].[DESPACHO]LIKE'%" &
CSTR(document.getElementById("txtPesq").value)  & "%';"
                end if
                writeTable(StrQuery)
            end if
        end sub

        sub BotaoI()
            cleartable()
            tipotxt=verificatxtbox()
            if document.getElementById("checkBoxexato").Checked then
```

```
                if document.getElementById("checkBoxLimit").checked then

                    strquery="Select TOP 100 * from TABELA_GERAL where
[TABELA_GERAL].[INTERESSADO]='" & CSTR(document.getElementById("txtPesq").value)
 & "';"
                    strQuery2="Select TOP 100  *  from TUDO where
[TUDO].[INTERESSADO]=" & CSTR(document.getElementById("txtPesq").value)  & ";"
                Else
                    strquery="Select  top 250   * from TABELA_GERAL where
[TABELA_GERAL].[INTERESSADO]='" & CSTR(document.getElementById("txtPesq").value)
 & "';"
                    strquery="Select  top 250   * from TUDO where
[TUDO].[INTERESSADO]='" & CSTR(document.getElementById("txtPesq").value)  & "';"
                end If
            Else
                if document.getElementById("checkBoxLimit").checked then

                    strquery="Select TOP 10 * from TABELA_GERAL where
[TABELA_GERAL].[INTERESSADO]LIKE'%" &
CSTR(document.getElementById("txtPesq").value)  & "%';"
                    strQuery2="Select top 10 * from TUDO where
[TUDO].[INTERESSADO]LIKE'%" & CSTR(document.getElementById("txtPesq").value)  &
"%';"
                else
                    strquery="Select  top 250   * from TABELA_GERAL where
[TABELA_GERAL].[INTERESSADO]LIKE'%" &
CSTR(document.getElementById("txtPesq").value)  & "%';"
                    strQuery2="Select   top 250  * from TUDO where
[TUDO].[INTERESSADO]LIKE'%" & CSTR(document.getElementById("txtPesq").value)  &
"%';"
                end if
            end If
            writeTable(StrQuery)
            writeTable2(StrQuery2)
        end sub

        sub BotaoL()
            cleartable()
            tipotxt=verificatxtbox()
            if document.getElementById("checkBoxexato").Checked then
                if document.getElementById("checkBoxLimit").checked then
                    strquery="Select top 100 * from TABELA_GERAL where
[TABELA_GERAL].[LOGRADOURO]='" &  CSTR(document.getElementById("txtPesq").value)
 & "';"
                    strquery="Select top 100 * from TUDO where
[TUDO].[ENDERECO]='" &  CSTR(document.getElementById("txtPesq").value)  & "';"
                else
                    strquery="Select   top 250  * from TABELA_GERAL where
[TABELA_GERAL].[LOGRADOURO]='" &  CSTR(document.getElementById("txtPesq").value)
 & "';"
                    strquery="Select  top 250   *  from TUDO where
[TUDO].[ENDERECO]='" &  CSTR(document.getElementById("txtPesq").value)  & "';"
                end if
            Else
```

```
                if document.getElementById("checkBoxLimit").checked then
                    strquery="Select  TOP 100 * from TABELA_GERAL where
[TABELA_GERAL].[LOGRADOURO]LIKE '%" &
CSTR(document.getElementById("txtPesq").value)  & "%';"
                    strQuery2="Select  TOP 100 * from TUDO where
[TUDO].[ENDERECO]LIKE '%" &  CSTR(document.getElementById("txtPesq").value)  &
"%';"
                else
                    strquery="Select   top 250  * from TABELA_GERAL where
[TABELA_GERAL].[LOGRADOURO]LIKE '%" &
CSTR(document.getElementById("txtPesq").value)  & "%';"
                    strQuery2="Select   top 250  * from TUDO where
[TUDO].[ENDERECO]LIKE '%" &  CSTR(document.getElementById("txtPesq").value)  &
"%';"
                end if
            end if
            writeTable(StrQuery)
            writeTable2(strquery2)
        end sub

        sub cleartable()
            'ok
            tableHTML=""
            table2HTML=""
            document.getElementById("saidaTab").innerHTML = tableHTML
            document.getElementById("saida2Tab").innerHTML = table2HTML
        end sub

        function formataNum(strEntr,tam)
            dim nFormat
            nFormat = Right(String(tam, "0") & strEntr, tam)
            formataNum=nFormat
        end function

        function verificatxtbox()
        'msgbox "debug verificatxtbox"
            if document.getElementById("txtPesq").value= "" then
                msgbox "Valor nao pode ser vazio"
                verificatxtbox=0
            else
                if isnumeric(document.getElementById("txtPesq").value)then
                    verificatxtbox=1 'numero
                elseif isdate(document.getElementById("txtPesq").value) then
                    verificatxtbox=2 'data
                else
                    verificatxtbox=3 'outros, princ. string
                end if
            end if
        end function

        function qtRegistros()
            'if (document.getElementById("checkBoxLimit").checked=1) then
                msgbox 1
            'else (if document.getElementById("checkBoxLimit2").checked=1) then
```

```
                msgbox 2
            'else if
((document.getElementById("checkBoxLimit2").checked=0)&&(document.getElementById
("checkBoxLimit2").checked=0)) then
                msgbox 3
            'end if
        end function

        function fclimit(num)
            if num=0 then
                document.getElementById("checkBoxLimit").checked=1
                document.getElementById("checkBoxLimit2").checked=0

            else
                document.getElementById("checkBoxLimit").checked=0
                document.getElementById("checkBoxLimit2").checked=1
            end if
        end function

        Sub ordenaTab1(columnIndex)
        'msgbox "debug ordena1"
        'similar ao exemplo, mas com 5 colunas
            Dim table, rows, switching, i, x, y, shouldSwitch
            Set table = Document.getElementById("idhtmltab1")
            'msgbox "tabela " &  table
            switching = True
            Do While switching
                switching = False
                Set rows = table.getElementsByTagName("TR")
                For i = 1 To rows.length - 2
                    shouldSwitch = False
                    Set x = rows(i).getElementsByTagName("TD")(columnIndex)
                    'msgbox "debug sw" & x
                    Set y = rows(i + 1).getElementsByTagName("TD")(columnIndex)
                    If x.innerText > y.innerText Then
                        shouldSwitch = True
                        Exit For
                    End If
                Next
                If shouldSwitch Then
                    rows(i).parentNode.insertBefore rows(i + 1), rows(i)
                    switching = True
                End If
            Loop
        End Sub

        function fcOrdenaE()
            ordenaTab1(0)
        END function

        function fcOrdenaC()
            ordenaTab1(1)
        end function
```

```
function fcOrdenaI()
    ordenaTab1(2)
end function

'campo removido
'function fcOrdenaTL()
'    ordenaTab1()
'end function

function fcOrdenaL()
    ordenaTab1(3)
end function

Sub ordenaTab2(columnIndex)
'similar ao exemplo, mas com 5 colunas
'funcao walkaround
    Dim table, rows, switching, i, x, y, shouldSwitch
    Set table = Document.getElementById("idhtmltab2")
    'msgbox "tabela 2" &  table
    switching = True
    Do While switching
        switching = False
        Set rows = table.getElementsByTagName("TR")
        For i = 1 To rows.length - 2
            shouldSwitch = False
            Set x = rows(i).getElementsByTagName("TD")(columnIndex)
            'msgbox "debug sw" & x
            Set y = rows(i + 1).getElementsByTagName("TD")(columnIndex)
            If x.innerText > y.innerText Then
                shouldSwitch = True
                Exit For
            End If
        Next
        If shouldSwitch Then
            rows(i).parentNode.insertBefore rows(i + 1), rows(i)
            switching = True
        End If
    Loop
End Sub

function fcOrdena2AL()
    ordenaTab2(0)
END function

function fcOrdena2AA()
    ordenaTab2(1)
end function

function fcOrdena2I()
    ordenaTab2(2)
end function

function fcOrdena2E()
    ordenaTab2(3)
```

```
        end function

        Sub writeTable(StrQuery)
        'passagem de parametros falhou, solucao criar duas funcoes
            set oRst = CreateObject("ADODB.recordset")
            oRst.Open StrQuery, sConnect
            Dim tableHTML
            tableHTML = "<p class='blue-text'><br><br><table id='idhtmltab1'
border='1'><em><b>TABELA DE ASSUNTOS</b></em></p>"
            tableHTML=tableHTML & "<tr>"
            dim cont, contmod, SDBse, SDBmil, SDBcem, SDBano,SDBdv, SDBcapa,
SDBAnoc,SDBTL, SDBLOG,varEscrevertab ,varEscreverfmt
            cont=1    'mod 14
            tableHTML=tableHTML & "<th onclick='fcOrdenaE'>ETIQUETA</th>"
            tableHTML=tableHTML & "<th onclick='fcOrdenaC'>CAPA</th>"
            tableHTML=tableHTML & "<th onclick='fcOrdenaI'>INTERESSADO</th>"
            'OBS**removido** tableHTML=tableHTML & "<th
onclick='fcOrdenaTL'>TL</th>"
            tableHTML=tableHTML & "<th onclick='fcOrdenaL'>LOGRADOURO</th>"
            tableHTML=tableHTML & "</tr>"

            cont=1    'mod 14
            varEscrevercap=0
            varEscreverfmt=0
            Do Until oRst.EOF
                tableHTML = tableHTML & "<tr>"
                'tableHTML=tableHTML & "<tr>"
                For Each field In oRst.Fields
                    contmod=cont mod (14)
                    if contmod=1 then
                        SDBse=formataNum(field.Value,2)
                    elseif contmod=2 then
                        SDBmil=formataNum(field.Value,3)
                    elseif contmod=3 then
                        SDBcem=formataNum(field.Value,3)
                    elseif contmod=4 then
                        SDBano=formataNum(field.Value,2)
                    elseif contmod=5 then
                        SDBdv=formataNum(field.Value,2)
                        varEscrevercap=1
                    elseif contmod=6 then
                        SDBcapa=formataNum(field.Value,6)
                    elseif contmod=7 then
                        SDBAnoc=formataNum(field.Value,2)

                        varEscreverfmt=1
                    elseif contmod=12 then
                        tableHTML = tableHTML & "<td>" & field.value & "</td>"
                    elseif contmod=13 then
                        SDBTL=field.Value
                    elseif contmod=0 then
                        tableHTML = tableHTML & "<td>" & SDBTL &" " &
field.Value & "</td>"
                    end if
```

```
                    if varEscrevercap=1 then
                    'trocado, corrigido 09.2024
                        tableHTML = tableHTML & "<td class='red-text'>" & SDBse
&"-"& SDBmil &"." & SDBcem & "-" & SDBano & "*" & SDBdv & "</td>"

                        varEscrevercap=0
                    end if

                    if varEscreverfmt=1 then
                    'trocado, corrigido 09.2024
                        tableHTML = tableHTML & "<td>" & SDBcapa &"/"& SDBanoc
&"</td>"
                        varEscreverfmt=0
                    end if
                    cont=cont+1
                Next
            oRst.MoveNext
            tableHTML=tableHTML & "</tr>"
            Loop
            document.getElementById("saidaTab").innerHTML = tableHTML
        End Sub

        If Err.Number <> 0 Then
            MsgBox "Error #" & Err.Number & ": " & Err.Description
            Err.Clear
        Else
            'MsgBox "Connection successful!"
        End If

        Sub writeTable2(StrQuery)
        'funcao walkaround
            set oRst = CreateObject("ADODB.recordset")
            oRst.Open StrQuery, sConnect
            Dim table2HTML
            table2HTML = "<p class='blue-text'><br><br><table id='idhtmltab2'
border='1'><em><b>TABELA DE ALVARAS</b></em></p>"
            table2HTML=table2HTML & "<tr>"
            'removidos em 08.2024 -- campos não utilizados
            'table2HTML=table2HTML & "<th onclick=''>Alvara</th>"
            'table2HTML=table2HTML & "<th onclick=''>ID</th>"
            table2HTML=table2HTML & "<th onclick='fcOrdena2AL'>ALVARA # </th>"
            table2HTML=table2HTML & "<th onclick='fcOrdena2AA'> ANO ALVARA
</th>"
            table2HTML=table2HTML & "<th onclick='fcOrdena2I'> INTERESSADO
</th>"
            table2HTML=table2HTML & "<th onclick='fcOrdena2E'> ENDERECO </th>"

            table2HTML=table2HTML & "</tr>"
            cont=1    'mod 14
            Do Until oRst.EOF
                table2HTML = table2HTML & "<tr>"
                For Each field In oRst.Fields
                    contmod=cont mod (12)
```

```
                    'if contmod=1 then
                    '    table2HTML = table2HTML & "<td>" & field.value &
"</td>"
                    'elseif contmod=5 then
                    '    table2HTML = table2HTML & "<td>" & field.value &
"</td>"
                    if contmod=7 then
                        table2HTML = table2HTML & "<td>" & field.value & "</td>"

                    elseif contmod=9 then
                        table2HTML = table2HTML & "<td>" & field.value & "</td>"

                    elseif contmod=11 then
                        table2HTML = table2HTML & "<td>" & field.value & "</td>"

                    elseif contmod=0 then
                        table2HTML = table2HTML & "<td>" & field.value & "</td>"

                    end if
                    cont=cont+1
                Next
            oRst.MoveNext
            table2HTML=table2HTML & "</tr>"
            Loop
            document.getElementById("saida2Tab").innerHTML = table2HTML
        End Sub

        If Err.Number <> 0 Then
            MsgBox "Error #" & Err.Number & ": " & Err.Description
            Err.Clear
        Else
            'MsgBox "Connection successful!"
        End If
    </script>

<body onload="fcarrega()">
    <div style="text-align: center;">
        <!-- tirei <h1> TESTE BDI </h1>       -->
        <input type="text" id="txtPesq" >
        <label for="checkBoxexato">Exatamente</label>
        <input type ="checkbox" id="checkBoxexato" value = 0>
        <label for="checkBoxLimit">       Limitar a 100
resultados</label>
        <input type ="checkbox" id="checkBoxLimit" onclick='fclimit(0)' value =
"0">
        <label for="checkBoxLimit2">       Maximo</label>

        <input type ="checkbox" id="checkBoxLimit2" onclick='fclimit(1)' value =
"1">
        <br> <br>
        <input type="button" value="ASSUNTO" onclick="BotaoA()">
        <input type="button" value="ALVARA" onclick="BotaoALVARA()">
        <input type="button" value="DATA R" onclick="BotaoDr()">
        <input type="button" value="DATA D" onclick="BotaoDd()">
```

```
        <input type="button" value="DESPACHO" onclick="BotaoD()">
        <input type="button" style="color:blue;" value="INTERESSADO"
onclick="BotaoI()">
        <input type="button" style="color:blue;" value="LOGRADOURO"
onclick="BotaoL()">
        <br>
        <br>
        <div id="saidaTab"> </div> <                    -->
        <div id="saida2Tab"> </div>
    </div>
</body>
</head>
</html>
```

Apêndice 2.:

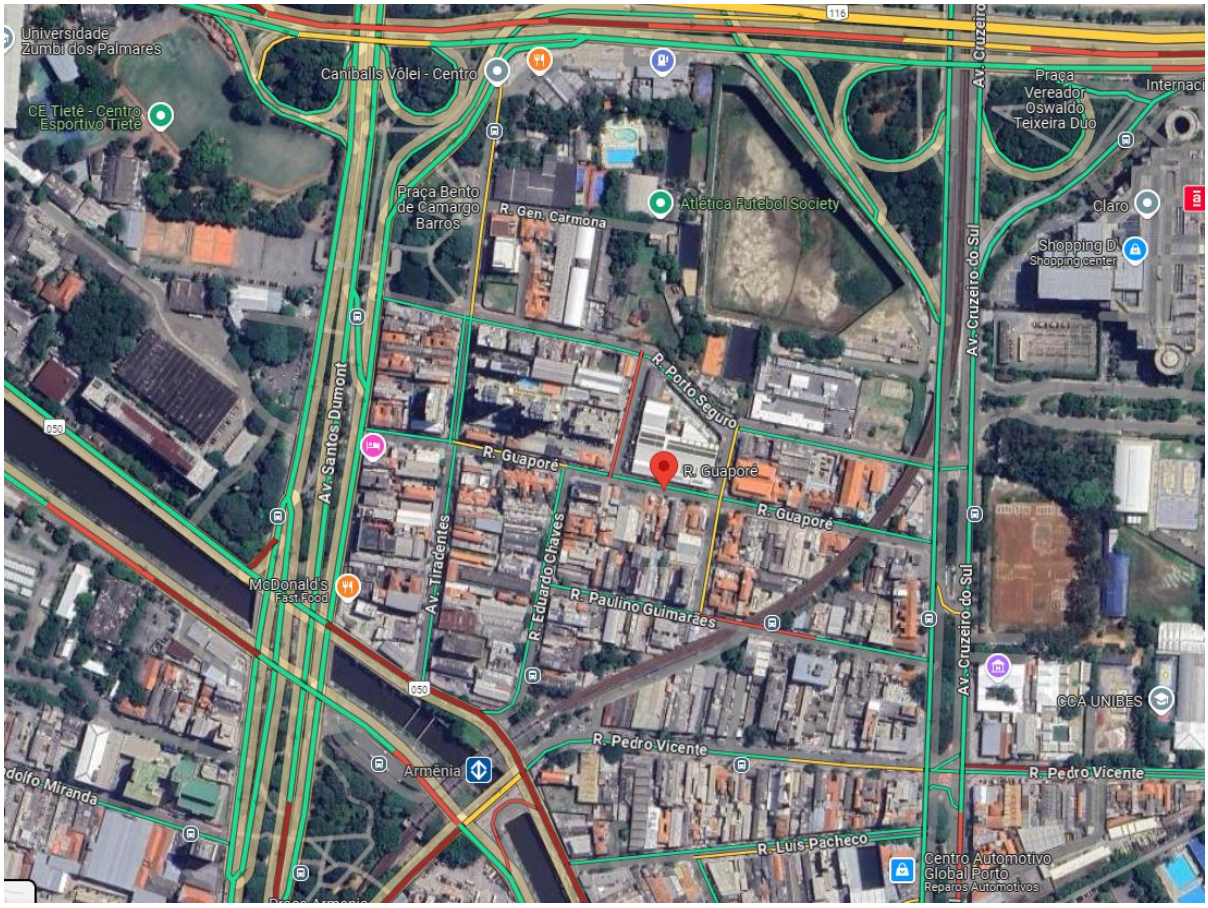

Fig 1. Exemplo de localidade com alagamento observada no *google maps*.

Fig 2. Estimativa (1) do congestionamento nos arredores para esta ocorrência

Fig 3. Estimativa (2) do congestionamento nos arredores para esta ocorrência:

Fig 4. Estimativa (3) do congestionamento nos arredores para esta ocorrência:

Apêndice 3.: Dicionário de Dados

TABELA	CAMPO	POS. ORDINAL	TIPO	NULLABLE ?
temas	tema	1	text	NO
temas	assunto	2	text	YES
temas	servico	3	text	YES
temas	nid	4	text	NO
temas	acron	5	text	YES
alagamentos	endereco	2	text	YES
alagamentos	referencia	3	text	YES
alagamentos	provavelest	6	text	YES
alagamentos	sentido	7	text	YES
alagamentos	transitabilidade	8	text	YES
alagamentos	dhicorrigido	9	text	YES
alagamentos	dhfcorrigido	10	text	YES
escola_infantil	tipo_escola	1	text	YES
escola_infantil	cdunidade	2	text	YES
escola_infantil	nome_escola	3	text	YES
escola_infantil	dre_escola	4	text	YES
escola_infantil	cdaluno	5	text	YES
escola_outros	cod	1	text	YES
escola_outros	cdunidade	2	text	YES
escola_outros	nome_escola	3	text	YES
escola_outros	dre_escola	4	text	YES
escola_outros	cdaluno	5	text	YES
escolas_loc	cod	1	integer	NO
escolas_loc	nome	2	text	YES
escolas_loc	lat	3	text	YES
escolas_loc	long	4	text	YES
estacoes	estacao_nome	1	text	YES
estacoes	estacao_id	2	integer	YES
estacoes	longitude_estim	3	text	YES
estacoes	latitude_estim	4	text	YES
historico_meteorologia	estacao	1	integer	YES
historico_meteorologia	datahora	2	text	YES
historico_meteorologia	chuva_periodo_atual	3	real	YES
historico_meteorologia	chuva_periodo_anterior	4	real	YES
historico_meteorologia	chuva_zeramento	5	text	YES
historico_meteorologia	temperatura_atual	6	real	YES
historico_meteorologia	temperatura_maxima	7	real	YES
historico_meteorologia	temperatura_minima	8	real	YES
historico_meteorologia	vento_direcao	9	text	YES

historico_meteorologia	vento_velocidade	10	real	YES
historico_meteorologia	vento_rajada	11	real	YES
inmet_2024	data	1	text	YES
inmet_2024	hora_utc	2	text	YES
inmet_2024	precipitacao_total	3	real	YES
inmet_2024	pressao_atmosferica	4	real	YES
inmet_2024	pressao_max	5	real	YES
inmet_2024	pressao_min	6	real	YES
inmet_2024	radiacao_global	7	real	YES
inmet_2024	temperatura_ar	8	real	YES
inmet_2024	ponto_orvalho	9	real	YES
inmet_2024	temperatura_max	10	real	YES
inmet_2024	temperatura_min	11	real	YES
inmet_2024	orvalho_max	12	real	YES
inmet_2024	orvalho_min	13	real	YES
inmet_2024	umidade_rel_max	14	real	YES
inmet_2024	umidade_rel_min	15	real	YES
inmet_2024	umidade_relativa	16	real	YES
inmet_2024	vento_direcao	17	real	YES
inmet_2024	vento_rajada	18	real	YES
inmet_2024	vento_velocidade	19	real	YES
inmet_2024	datahora_completa	20	text	YES
obitos	id_deleg	1	integer	YES
obitos	nome_delegacia	2	text	YES
obitos	numero_bo	3	integer	YES
obitos	ano_bo	4	integer	YES
obitos	data_obito	5	text	YES
obitos	data_sinistro	6	text	YES
obitos	tipo_via	7	text	YES
obitos	dia_obito	8	integer	YES
obitos	mes_obito	9	integer	YES
obitos	ano_obito	10	integer	YES
obitos	ano_mes_obito	11	text	YES
obitos	dia_semana	12	text	YES
obitos	hora_sinistro	13	text	YES
obitos	turno	14	text	YES
obitos	municipio	15	text	YES
obitos	regiao_administrativa	16	text	YES
obitos	logradouro	17	text	YES
obitos	numeral_km	18	text	YES
obitos	jurisdicao	19	text	YES
obitos	administracao	20	text	YES
obitos	conservacao	21	text	YES
obitos	tipo_local_sinistro	22	text	YES

obitos	latitude	23	real	YES
obitos	longitude	24	real	YES
obitos	meio_locomocao_vitima	25	text	YES
obitos	tipo_vitima	26	text	YES
obitos	local_obito	27	text	YES
obitos	tipo_sinistro	28	text	YES
obitos	sexo	29	text	YES
obitos	faixa_etaria	30	text	YES
obitos	idade_vitima	31	integer	YES
obitos	outro_veiculo_envolvido	32	text	YES
obitos	tempo_sinistro_obito	33	text	YES
risco	_id	1	integer	YES
risco	datadeabertura	2	text	YES
risco	canal	3	text	YES
risco	tema	4	text	YES
risco	assunto	5	text	YES
risco	servico	6	text	YES
risco	logradouro	7	text	YES
risco	numero	8	real	YES
risco	cep	9	real	YES
risco	subprefeitura	10	text	YES
risco	distrito	11	text	YES
risco	latitude	12	real	YES
risco	longitude	13	real	YES
risco	datadoparecer	14	text	YES
risco	statusdasolicitacao	15	text	YES
risco	orgao	16	text	YES
risco	dataavaliacaopesquisasatisfacao	17	text	YES
risco	nivelsatisfacao	18	text	YES
risco	qualidadeservico	19	text	YES
risco	atendeusolicitacao	20	text	YES
risco	compreensaodaresposta	21	text	YES
sinistros_fatais	id_delegacia	1	text	YES
sinistros_fatais	numero_bo	2	text	YES
sinistros_fatais	ano_bo	3	text	YES
sinistros_fatais	nome_delegacia	4	text	YES
sinistros_fatais	data_sinistro	5	text	YES
sinistros_fatais	dia_sinistro	6	text	YES
sinistros_fatais	mes_sinistro	7	text	YES
sinistros_fatais	ano_sinistro	8	text	YES
sinistros_fatais	ano_mes_sinistro	9	text	YES
sinistros_fatais	dia_semana	10	text	YES
sinistros_fatais	hora_sinistro	11	text	YES
sinistros_fatais	turno	12	text	YES

sinistros_fatais	municipio	13	text	YES
sinistros_fatais	regiao_administrativa	14	text	YES
sinistros_fatais	logradouro	15	text	YES
sinistros_fatais	numeral_km	16	text	YES
sinistros_fatais	jurisdicao	17	text	YES
sinistros_fatais	administracao	18	text	YES
sinistros_fatais	conservacao	19	text	YES
sinistros_fatais	tipo_local_sinistro	20	text	YES
sinistros_fatais	latitude	21	text	YES
sinistros_fatais	longitude	22	text	YES
sinistros_fatais	tipo_via	23	text	YES
sinistros_fatais	pedestre_envolvido	24	text	YES
sinistros_fatais	automovel_envolvido	25	text	YES
sinistros_fatais	bicicleta_envolvida	26	text	YES
sinistros_fatais	caminhao_envolvido	27	text	YES
sinistros_fatais	motocicleta_envolvida	28	text	YES
sinistros_fatais	onibus_envolvido	29	text	YES
sinistros_fatais	outros_veiculos_envolvidos	30	text	YES
sinistros_fatais	veiculo_envolvido_nao_disponivel	31	text	YES
sinistros_fatais	tempo_entre_sinistro_obito	32	text	YES
sinistros_fatais	atropelamento	33	text	YES
sinistros_fatais	capotamento	34	text	YES
sinistros_fatais	choque	35	text	YES
sinistros_fatais	colisao_frontal	36	text	YES
sinistros_fatais	colisao_lateral	37	text	YES
sinistros_fatais	colisao_transversal	38	text	YES
sinistros_fatais	colisao_traseira	39	text	YES
sinistros_fatais	outras_colisoes	40	text	YES
sinistros_fatais	engavetamento	41	text	YES
sinistros_fatais	tombamento	42	text	YES
sinistros_fatais	outros_sinistros	43	text	YES
sinistros_fatais	quantidade_vitimas_fatais	44	text	YES
sinistros_nao_fatais	id_sinistro	1	text	YES
sinistros_nao_fatais	data_sinistro	2	text	YES
sinistros_nao_fatais	dia_sinistro	3	text	YES
sinistros_nao_fatais	mes_sinistro	4	text	YES
sinistros_nao_fatais	ano_sinistro	5	text	YES
sinistros_nao_fatais	ano_mes_sinistro	6	text	YES
sinistros_nao_fatais	latitude	7	real	YES
sinistros_nao_fatais	longitude	8	real	YES
sinistros_nao_fatais	hora_sinistro	9	text	YES
sinistros_nao_fatais	municipio	10	text	YES
sinistros_nao_fatais	regiao_administrativa	11	text	YES
sinistros_nao_fatais	logradouro	12	text	YES

sinistros_nao_fatais	numero_km	13	text	YES
sinistros_nao_fatais	pedestre_envolvido	14	integer	YES
sinistros_nao_fatais	automovel_envolvido	15	integer	YES
sinistros_nao_fatais	bicicleta_envolvida	16	integer	YES
sinistros_nao_fatais	caminhao_envolvido	17	integer	YES
sinistros_nao_fatais	motocicleta_envolvida	18	integer	YES
sinistros_nao_fatais	onibus_envolvido	19	integer	YES
sinistros_nao_fatais	outros_veiculos_envolvidos	20	integer	YES
sinistros_nao_fatais	veiculo_envolvido_nao_disponivel	21	integer	YES
sinistros_nao_fatais	atropelamento	22	integer	YES
sinistros_nao_fatais	choque	23	integer	YES
sinistros_nao_fatais	colisao	24	integer	YES
sinistros_nao_fatais	outros	25	integer	YES
sinistros_nao_fatais	nao_disponivel	26	integer	YES
sinistros_nao_fatais	tipo_registro	27	text	YES