Agregação de dados de múltiplas fontes

Relatório Final do Projeto Integrador

Bruno Coutinho Pereira
Bruno Rafael Granja Alves
Carlos Ferreira Marta Carreira da Costa
Luís Filipe Gonçalves Fernandes

U.PORTO

Licenciatura em Engenharia Informática e Computação

Tutor na U.Porto: Bruno Lima
Orientador na empresa/Proponente: Pedro Pimenta

27 de Julho, 2025

Conteúdo

1 Introdução

1.1 Enquadramento

O presente relatório tem como objetivo descrever o desenvolvimento do Projeto Integrador "Agregação de Dados de Múltiplas Fontes", realizado no âmbito da Licenciatura em Engenharia Informática e Computação da U.Porto pelo grupo PE01, constituído por Bruno Pereira, Bruno Alves, Luís Fernandes e Carlos Costa.

O trabalho foi conduzido em parceria com o **D4Maia**[1], grupo de trabalho da Câmara Municipal da Maia criado em 2020 no contexto do projeto **BaZe – Balanço Zero de Carbono**[2]. O D4Maia mantém um *data lake*, demonstrado na figura 1, que integra atualmente mais de cinquenta fontes de dados heterogéneas (sensores de mobilidade, consumo de energia, qualidade do ar, dados sociodemográficos, entre outros), suportando decisões operacionais e estratégicas da autarquia.

Apesar da vasta quantidade de informação disponível, constatou-se que muitas fontes permanecem isoladas e com formatos incompatíveis, o que dificulta a sua análise integrada. Esta heterogeneidade — presença de diferentes esquemas, formatos (CSV, JSON, geoJSON, GTFS, Excel, APIs REST, etc.) e níveis de qualidade — gera redundâncias, inconsistências e lacunas nos dados municipais. A motivação deste projeto prende-se na necessidade de consolidar essas informações de modo a criar um repositório coerente, extensível e reutilizável, capaz de produzir insights mais fiáveis para a gestão urbana, como pode ser visto na figura 1.

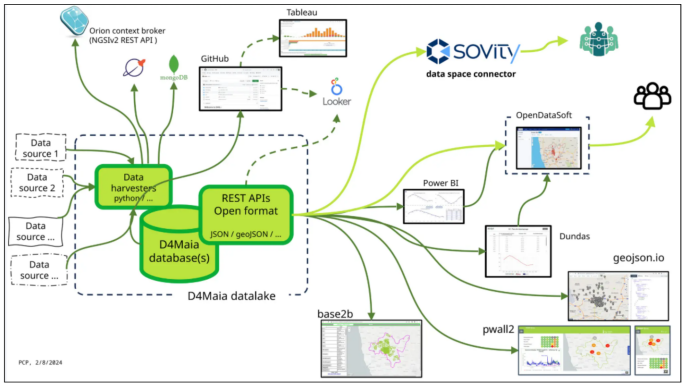


Figura 1: D4Maia Architeture (imagem fornecida pelo proponente)

1.2 Objetivos e resultados esperados

O principal objetivo do trabalho consistiu em desenvolver e otimizar **scripts em Python** para a recolha, limpeza e harmonização de dados provenientes de diversas fontes, utilizando métodos que serão detalhados posteriormente neste relatório. O foco do projeto foi nas instituições de ensino, dado o seu impacto social e potencial utilidade para a gestão municipal. Dado esse foco, foram disponibilizados dados adicionais relacionados com esse domínio, o que será expresso em um maior número de correspondências (*matchings*) entre esse tipo de instituições.

Foram definidos os seguintes objetivos para o trabalho:

- Padronizar e reconciliar registos divergentes, eliminando duplicidades e preenchendo lacunas de informação.
- Gerar uma tabela final agregada que consolide, de forma clara e estruturada, todos os atributos relevantes das escolas do concelho da Maia.
- Documentar o processo de forma detalhada, garantindo reprodutibilidade e possibilidade de extensão para novos domínios e futuros trabalhos.

1.3 Estrutura do relatório

Para além da presente introdução, o documento encontra-se dividido em mais 3 capítulos.

1. **Metodologia e Atividades Desenvolvidas**: descreve o método de trabalho, as principais tarefas realizadas e as ferramentas utilizadas ao longo do projeto.
2. **Desenvolvimento da Solução**: detalha os requisitos, arquitetura, tecnologias empregues, implementação dos scripts e validação dos resultados obtidos.
3. **Conclusões**: sumariza os resultados alcançados, reflete sobre as lições aprendidas e propõe possíveis extensões e melhorias para trabalhos futuros.

2 Metodologia utilizada e principais atividades desenvolvidas

2.1 Metodologia utilizada

Para o desenvolvimento do projeto, adotámos a seguinte abordagem: o **Scrum**. Esta metodologia baseia-se em ciclos de desenvolvimento iterativo e incremental, acompanhados de reuniões semanais com o proponente, nas quais foi possível apresentar o progresso alcançado, esclarecer dúvidas e recolher feedback contínuo, permitindo ajustar o rumo do trabalho sempre que necessário.

Relativamente à gestão e partilha de recursos, foram utilizadas duas ferramentas:

- **GitHub** - Para a partilha de ficheiros [3]
- **Google Colab** - Para o desenvolvimento do código [4]

2.2 Intervenientes, papéis e responsabilidades

Os membros do grupo são:

- **Bruno Coutinho Pereira** – responsável pela filtragem, correspondência de strings (string matching) e análise de resultados.
- **Bruno Rafael Granja Alves** – responsável pela importação e recolha de tabelas, filtragem, correspondência via coordenadas e via strings, análise de resultados e construção da tabela final.
- **Carlos Ferreira Marta Carreira da Costa** – responsável pela filtragem, correspondência de coordenadas e elaboração da tabela final.
- **Luís Filipe Gonçalves Fernandes** – responsável pela filtragem, correspondência de coordenadas e análise de resultados.

Em termos de **stakeholders**, contámos com a colaboração dos trabalhadores da Câmara Municipal da Maia, em particular **Pedro Pimenta**, que nos prestou feedback regular sobre o trabalho desenvolvido e esteve disponível para o esclarecimento de dúvidas ao longo do projeto.

Por fim, tivemos também a intervenção de **Bruno Lima** (tutor da FEUP), que participou em algumas reuniões com o objetivo de agilizar o projeto, prestando apoio em questões relacionadas com os requisitos formais da FEUP, como o formato do poster, do relatório, entre outros aspetos.

2.3 Atividades desenvolvidas

2.3.1 Análise inicial dos dados e preparação do ambiente de trabalho

Após a decisão sobre o caso de estudo a abordar, iniciámos a análise dos ficheiros fornecidos, relacionados com as escolas. Esta análise teve como objetivo

facilitar a identificação e o manuseamento dos diferentes conjuntos de dados ao longo do projeto. Para isso, atribuímos nomes de referência (placeholders) mais descritivos aos ficheiros, nomeadamente:

- **Contadores**,
- **Equipamento**,
- **Energia**,
- **Imóveis**,
- **Edifícios**.

Importa referir que estes nomes não correspondem aos nomes originais dos ficheiros, mas foram definidos para facilitar a comunicação durante o desenvolvimento.

Durante a análise inicial, surgiram diversas dúvidas relativamente ao conteúdo dos ficheiros e ao modo como os dados deveriam ser tratados. Estas foram esclarecidas através de trocas de mensagens com o proponente, e envolveram, entre outras, as seguintes questões:

- Interpretação dos títulos das colunas;
- Possibilidade de eliminar linhas duplicadas;
- Utilização de filtros com base na tipologia presente em alguns ficheiros;
- Estratégia a adotar para lidar com lacunas nos dados (por exemplo, casos em que os ficheiros apresentavam uma interrupção e retomavam n linhas mais abaixo).

Com base nesta análise preliminar, foi criado um ambiente de trabalho colaborativo em **Google Colab**, complementado com o uso de **Jupyter Notebook**. Esta escolha visou:

- Reduzir a necessidade de recompilar todo o código após cada modificação (via Jupyter Notebook);
- Permitir o acesso e a colaboração em tempo real entre os membros do grupo e o proponente (via Google Colab).

Posteriormente, foram configurados *tokens* e *secrets* para permitir o acesso ao repositório GitHub onde se encontravam os ficheiros de dados, garantindo que todos os elementos podiam trabalhar de forma integrada.

Para garantir esse acesso aos ficheiros privados armazenados no GitHub, foi necessário gerar um *token* pessoal de acesso e configurá-lo como *secret* no ambiente do Google Colab. Este procedimento consistiu nos seguintes passos:

1. Aceder a *Settings* → *Developer Settings* no GitHub;

2. Selecionar *Tokens Classic* e clicar em *Generate new token (classic)*;
3. Definir as permissões adequadas e gerar o token;
4. Copiar o token e adicioná-lo como *secret* no Google Colab com o nome `passToken`;
5. Autorizar o notebook a aceder ao *secret* criado.

Após a configuração do acesso, foram instaladas as bibliotecas necessárias ao projeto com o seguinte comando:

```
pip install RapidFuzz
```

Listing 1: Instalação de dependências

Seguidamente, foram realizados os *imports* das bibliotecas que seriam utilizadas ao longo do projeto:

```
from google.colab import userdata
import pandas as pd
import requests
from io import BytesIO
import geopandas as gpd
from shapely.geometry import Polygon, Point
import ipywidgets as widgets
from rapidfuzz import process, fuzz
from IPython.display import display, HTML
from math import radians, sin, cos, sqrt, atan2
import numpy as np
import unicodedata
```

Listing 2: Imports utilizados

Com o token previamente armazenado como *secret*, foi possível aceder e carregar os ficheiros diretamente do repositório GitHub através da API:

```
# Aceder ao token armazenado como secret
GITHUB_TOKEN = userdata.get('passToken')

# URL da API para aceder ao ficheiro
FILE_URL = "https://api.github.com/repos/xxx/xxx/
    Contadores.xlsx"
headers = {"Authorization": f"token {GITHUB_TOKEN}"
    }

# Pedido a API do GitHub
response = requests.get(FILE_URL, headers=headers)

if response.status_code == 200:
    file_data = response.json()
    file_download_url = file_data["download_url"]

    file_response = requests.get(file_download_url)
    if file_response.status_code == 200:
        contadores = pd.read_excel(BytesIO(
            file_response.content))
```

```
            else:
                print("Erro ao descarregar o ficheiro:",
                    file_response.status_code)
        else:
            print("Erro ao obter informacoes do ficheiro:",
                 response.status_code)
```

Listing 3: Exemplo de carregamento de ficheiro do GitHub

Este procedimento foi aplicado de forma semelhante aos restantes ficheiros de dados, alterando apenas o nome do ficheiro na URL correspondente.

2.3.2 Filtragem inicial dos dados

Após a criação do notebook no Google Colab, procedeu-se à implementação do código responsável pela filtragem inicial dos ficheiros, tendo em conta as respostas às questões anteriormente colocadas ao proponente.

Devido à organização distinta de cada ficheiro, foram desenvolvidos métodos específicos para cada conjunto de dados:

- No caso do ficheiro *Contadores*, eliminaram-se as colunas consideradas irrelevantes para a identificação de escolas.
- Para o ficheiro *Equipamento*, realizou-se a remoção de todas as entradas duplicadas.
- A filtragem do ficheiro *Imóveis* foi efetuada em duas etapas: inicialmente, eliminaram-se colunas com conteúdo redundante; posteriormente, aplicou-se um filtro baseado em palavras-chave previamente definidas, através da análise das strings contidas nas colunas.
- Por fim, os ficheiros *Energia* e *Edifícios* foram submetidos a processos de filtragem semelhantes, que consistiram na remoção de colunas desnecessárias e em branco, seguida da aplicação de filtros na coluna denominada *Tipologia*. Para o ficheiro *Energia*, eliminaram-se as entradas cuja tipologia não fosse "Escolar"ou "Santa Casa". Para o ficheiro *Edifícios*, mantiveram-se apenas as entradas com tipologia "Escolar".

```
filtered_equipamentos = equipamentos.
    drop_duplicates(subset=["CodigoMISIescola", "
    NomeGrauEnsino"])
filtered_equipamentos = filtered_equipamentos.
    reset_index(drop=True)
```

Listing 4: Filtragem da tabela de Equipamento

```
filtered_imoveis = imoveis.drop(columns=['CONTA'])

keywords = [
    r"\bescola\b",
```

```
        r"\be\.?b\.?\s*\d?\b",  # Captura EB e as suas
            variacoes (pontos/espacos apos as letras,
            numeros no final)
        r"\bcreche\b",
        r"\binfant[ a ]rio\b",  # Captura Infantario
            com ou sem acento
        r"\bjardim\s*(?:de\s*)?inf[ a ]ncia\b"  #
            Captura Jardim de infancia com ou sem
            acento; "de" e opcional
    ]

    include_pattern = "|".join(keywords)

    filtered_imoveis_2 = filtered_imoveis[
        filtered_imoveis["DESIGNA AO"].astype(str).str
            .contains(include_pattern, case=False, na=
            False, regex=True)
    ]
```

Listing 5: Filtragem da tabela de Imoveis

```
    #Energia
    empty_columns_energetica = energetica.columns[
        energetica.isnull().all()]
    filtered_energetica = energetica.drop(
        empty_columns_energetica, axis=1)


    filtered_energetica_2 = filtered_energetica[
        filtered_energetica["Tipologia"].isin(["
        Escolares", "Santa Casa"])]

    #Edificios
    filtered_edificios = edificios.drop(columns=["Ano
        de  constru   o"])
    empty_columns_edificios = edificios.columns[
        edificios.isnull().all()]
    filtered_edificios = filtered_edificios.drop(
        empty_columns_edificios, axis=1)

    filtered_edificios_2 = filtered_edificios[
        filtered_edificios["Tipologia"].isin(["
        Escolares"])]
```

Listing 6: Filtragem da tabela de Energia e Edificios

Na reunião seguinte após este progresso, foi-nos alertado que é suposto relatar todo o progresso que tenhamos feito, quer ele tenha tido sucesso ou não.

2.3.3 Matching entre tabelas

Após a conclusão desta fase, foram colocadas novas questões ao proponente, nomeadamente relativamente ao agendamento da próxima reunião, uma vez que ainda não tinha sido definida uma data. Adicionalmente, surgiram dúvidas relacionadas com as possíveis abordagens para realizar a interseção entre as diferentes tabelas.

Neste contexto, foi também documentado o primeiro experimento realizado no âmbito do processo de correspondência entre tabelas (*matching*). Este consistiu na aplicação de um limiar de similaridade (*threshold*) de 85 na comparação entre os ficheiros `filtered_imoveis` e `filtered_edificios`, sendo que a designação `filtered_X` refere-se aos ficheiros após a aplicação do processo de filtragem descrito anteriormente.

Foram realizados alguns ajustes nos dados de modo a viabilizar a comparação entre as diferentes tabelas. Para a correspondência entre entradas, recorreu-se à métrica de distância de Levenshtein.

```
THRESHOLD = 100

matches_list = []

processed_matches = set()

filtered_imoveis['N.   INVENTARIO'] =
    filtered_imoveis['N.   INVENTARIO'].astype(str)
    .str.strip()
filtered_edificios['N.   Invent rio'] =
    filtered_edificios['N.   Invent rio'].astype(
    str).str.strip()

filtered_imoveis['N.   INVENTARIO'] =
    filtered_imoveis['N.   INVENTARIO'].str.replace
    (r'\.0$', '', regex=True)

for _, imoveis_row in filtered_imoveis.iterrows():
    number_imoveis = imoveis_row['N.   INVENTARIO']
    designation_imoveis = imoveis_row['DESIGNA AO'
        ]

    matches = process.extract(number_imoveis,
        filtered_edificios['N.   Invent rio'],
        scorer=fuzz.ratio, limit=None)

    for match in matches:
        if match[1] >= THRESHOLD and match[0] != '
            nan':
            matching_edificios = filtered_edificios
                [filtered_edificios['N.
                Invent rio'] == match[0]]
            if number_imoveis not in
                processed_matches:
                for _, edificios_row in
                    matching_edificios.iterrows():
                    matches_list.append({
                        'number_imoveis':
                            number_imoveis,
                        'designation_imoveis':
                            designation_imoveis,
                        'number_edificios': match
                            [0],
                        'designation_edificios':
                            edificios_row['
                            Designa  o'],
```

```
                'match_score': match[1]
            })
            processed_matches.add(match[0])
```

Listing 7: Matching entre Imoveis e Edificios

Após o esclarecimento quanto à metodologia a adotar, procedeu-se à atualização do processo de correspondência (*matching*) entre tabelas, tendo sido reportadas as alterações implementadas. Foi igualmente identificado que alguns procedimentos anteriores tinham sido realizados de forma manual, dificultando a escalabilidade do código para novas tabelas. Nesse sentido, foi sugerida a utilização de expressões regulares como forma de generalizar e tornar mais eficaz o processo de filtragem e integração de dados.

Adicionalmente, foi-nos recomendado tornar mais explícitas as motivações por detrás de certas modificações realizadas, bem como incluir indicadores nos resultados das junções de tabelas que permitissem identificar a origem de cada informação. Foi também salientada a importância de evitar a reutilização de parâmetros idênticos em diferentes contextos, uma vez que nem todas as tabelas contêm os mesmos campos. Como boa prática, sugeriu-se a segmentação do código em blocos distintos no Colab, com o objetivo de aumentar a clareza do processo de tratamento de dados.

Com base nessas orientações, procedeu-se à revisão do código, integrando expressões regulares nas etapas de filtragem, explicando as alterações realizadas e tornando o processo mais flexível. Em seguida, realizou-se uma nova etapa de *matching*, agora com base no código postal, utilizando um limiar de similaridade de 100. Para facilitar a interpretação dos resultados, foram introduzidas cores nas tabelas cruzadas, assinalando visualmente a origem dos dados.

Foi também testada uma abordagem preliminar baseada em coordenadas geográficas (latitude e longitude), mas esta revelou-se ineficaz devido à elevada presença de valores nulos nas colunas correspondentes.

Após a obtenção de resultados considerados próximos do objetivo inicial definido pelo grupo, decidiu-se integrar dados adicionais provenientes da plataforma BaZe, contendo informação relativa à designação e coordenadas geográficas das instituições.

```
url = "https://baze.cm-maia.pt/BaZe/api/xxx"

response = requests.get(url)
data = response.json()

entries = data["features"]

rows=[]
for entry in entries:
    designacao = entry["properties"].get("
        popupContent", "")
    longitude = entry["geometry"]["coordinates"][0]
    latitude = entry["geometry"]["coordinates"][1]
    rows.append({
        "designa  o": designacao,
        "longitude": longitude,
```

```
            "latitude": latitude
        })
```

Listing 8: Recolha da tabela BaZe

```
contadores_2_coord['key'] = 1
baze['key'] = 1
cross = pd.merge(contadores_2_coord, baze, on='key'
    ).drop('key', axis=1)

cross['distance_m'] = cross.apply(
    lambda row: haversine(row['Lon'], row['Lat'],
        row['latitude'], row['longitude']),
    axis=1
)

contadores_baze_2 = cross[cross['distance_m'] <=
    distance_threshold]

contadores_baze_2 = contadores_baze_2.rename(
    columns={
    'Lon':'Lat_contadores_2',
    'Lat':'Lon_contadores_2',
    'desig_edificio':'designacao_contadores_2',
    'Nome da Escola':'Escola_contadores_2',
    'longitude': 'Lon_baze',
    'latitude':'Lat_baze',
    'designation': 'designacao_baze'
})
```

Listing 9: Match tabela Contadores e BaZe

Posteriormente, realizou-se um ajuste na forma de cálculo da distância entre coordenadas, substituindo a medição em graus pela aplicação da fórmula de Haversine. Esta alteração permitiu expressar a distância em metros, proporcionando uma maior precisão nas correspondências geográficas entre os registos.

```
def haversine(lat1, lon1, lat2, lon2):
R = 6371000
dlat = radians(lat2 - lat1)
dlon = radians(lon2 - lon1)

a = sin(dlat/2)**2 + cos(radians(lat1)) * cos(
    radians(lat2)) * sin(dlon/2)**2
c = 2 * atan2(sqrt(a), sqrt(1 - a))

return R * c
```

Listing 10: Formula de Haversine

Posteriormente, foi disponibilizada uma nova versão do ficheiro **Contadores**, que incluía agora uma das colunas anteriormente incompletas — nomeadamente, as coordenadas geográficas. Adicionalmente, foi-nos demonstrado o processo para recolher informações provenientes de fontes externas, nomeadamente da plataforma **OpenStreetMaps (OSM)**, com o intuito de enriquecer os dados existentes.

Com a disponibilização desta nova versão dos dados, procedeu-se a uma comparação entre os registos antigos e os atualizados, com o objetivo de avaliar o impacto das alterações introduzidas, para além do preenchimento das coordenadas. Verificou-se que o método de filtragem previamente implementado continuava a ser eficaz face à nova estrutura dos dados. Durante esta análise, constatou-se que o número de correspondências era inferior, o que se justificava pela existência de valores nulos em colunas cruciais durante o processo de comparação entre tabelas.

Seguidamente, introduziu-se a tabela de escolas extraída do **OpenStreetMaps**, com o intuito de explorar diferentes estratégias de fusão de dados. Foram testadas três abordagens distintas quanto à ordem de integração das fontes de dados:

1. Iniciar com o conjunto base, seguido da junção com a tabela `BaZe`, e por fim com a tabela do `OSM` (resultando numa evolução de 73 para 77 registos);

2. Iniciar com o conjunto base, seguido do `OSM`, e por fim da `BaZe` (108 para 81 registos);

3. Iniciar com a tabela do `OSM`, seguida da `BaZe` e, por último, adicionar o conjunto base (49 para 91 registos).

A análise destes resultados permitiu perceber que a ordem de integração dos dados tem impacto significativo no número de correspondências finais. Este comportamento motivou a necessidade de estudar estratégias mais robustas e generalizáveis, capazes de produzir resultados consistentes independentemente da ordem de agregação das diferentes fontes.

```
contadores_baze_2['key'] = 1
escolas_OSM_matching['key'] = 1
cross = pd.merge(contadores_baze_2,
    escolas_OSM_matching, on='key').drop('key',
     axis=1)

cross['distance_m_2'] = cross.apply(
    lambda row: min(haversine(row['
        Lat_contadores_2'], row['
        Lon_contadores_2'], row['Latitude'],
        row['Longitude']), haversine(row['
        Lat_baze'], row['Lon_baze'], row['
        Latitude'], row['Longitude'])),
    axis=1
)

contadores_baze_escolas_OSM_2 = cross[cross['
    distance_m_2'] <= distance_threshold]

contadores_baze_escolas_OSM_2 =
    contadores_baze_escolas_OSM_2.rename(
    columns={
    'Name':'Name_escolas_OSM',
    'Latitude':'Latitude_escolas_OSM',
```

```
        'Longitude':'Longitude_escolas_OSM'
    })
```

Listing 11: Match tabela Contadores+BaZe com OSM

2.3.4 Construção de tabela resultado

Posteriormente, foi implementada uma funcionalidade que permite o descarregamento automático dos dados provenientes do **OpenStreetMaps**, facilitando assim a atualização e integração contínua dessas informações no processo de análise.

Com o auxílio de técnicas baseadas em IA, foi adicionada uma nova coluna à tabela de correspondência (*matching*), destinada a classificar os resultados como *verdadeiro positivo*, *falso positivo* ou *não identificado* — este último correspondente a casos em que não foi possível determinar, com segurança, a validade da correspondência.

Este desenvolvimento levantou novas questões relacionadas com a necessidade de generalização de *strings*, de forma a normalizar os dados textuais e permitir uma análise mais eficaz e consistente.

```
replacements = {
    'escola ': 'e',
    ' escolar': 'e',
    'esc ': 'e',
    'secundaria': 's',
    'basica': 'b',
    'sec ': 's',
    'centro ': 'c',
    'jardim': 'j',
    'infancia':'i',
    ' de ': ' ',
    ' do ':' ',
    ' e ': ' ',
    ' com ': ' ',
    'n.o ': '',
    'ciclo ': '',
    '1.o': '1',
    '1o': '1',
}

def normalize(text):
    if pd.isna(text):
        return ""
    text = ''.join(c for c in unicodedata.
        normalize('NFKD', str(text)) if not
        unicodedata.combining(c))

    text = text.lower()

    for old, new in replacements.items():
        text = text.replace(old, new)

    return ' '.join(text.split())
```

```
threshold_c_b_e = 60
```

Listing 12: Normalização de dados

```
results = []

for i in range(len(
    contadores_baze_escolas_OSM_2)):
    row = contadores_baze_escolas_OSM_2.iloc[i]

    designacao_contadores_2 = normalize(row['
        designacao_contadores_2'])
    Escola_contadores_2 = normalize(row['
        Escola_contadores_2'])
    designacao_baze = normalize(row['designacao
        '])
    Name_escolas_OSM = normalize(row['
        Name_escolas_OSM'])

    if pd.isna(Name_escolas_OSM) or (pd.isna(
        designacao_contadores_2) and pd.isna(
        Escola_contadores_2) and pd.isna(
        designacao_baze)):
        results.append('non_identified')
        continue

    scores = []
    scores.append(fuzz.ratio(
        designacao_contadores_2,
        Name_escolas_OSM))
    scores.append(fuzz.ratio(
        Escola_contadores_2, Name_escolas_OSM))
    scores.append(fuzz.ratio(designacao_baze,
        Name_escolas_OSM))

    if max(scores) >= threshold_c_b_e:
        results.append('true_positive')
    else:
        results.append('false_positive')

contadores_baze_escolas_OSM_2['result'] =
    results

result_counts = contadores_baze_escolas_OSM_2['
    result'].value_counts()
```

Listing 13: Identificação de TP e FP

De seguida, procedeu-se à redução do *threshold* da correspondência geográfica de 100 para 70 metros, com o intuito de aumentar a sensibilidade do processo de correspondência. Paralelamente, foi aplicada uma filtragem adicional às *strings*, com o objetivo de generalizar termos menos relevantes e, assim, melhorar a fiabilidade da classificação automática na coluna de *falsos positivos*.

Como resultado dessas alterações, obtiveram-se os seguintes valores nas três abordagens testadas:

- Abordagem 1: 48 *True Positives* e 1 *False Positive*;
- Abordagem 2: 36 *True Positives* e 0 *False Positives*;
- Abordagem 3: 30 *True Positives* e 21 *False Positives*.

Verificou-se uma discrepância significativa na terceira abordagem, justificada pelo elevado número de entradas com valores nulos ou indefinidos no conjunto de dados inicial, o que contribuiu para um aumento considerável de falsos positivos.

Posteriormente, integrou-se no ambiente do Colab o método automatizado de recolha de dados provenientes do **OpenStreetMaps**, permitindo uma maior flexibilidade e atualização contínua da base de dados.

Finalmente, foi produzido um ficheiro **Excel** com o resultado final da análise, contendo o nome do edifício para todos os registos não nulos das tabelas utilizadas, bem como a respetiva designação e o método de correspondência aplicado. Esta tabela final foi disponibilizada em duas versões: uma versão completa, contendo todos os dados agregados, e uma versão alternativa, otimizada para facilitar a leitura e interpretação dos resultados.

```
valid_contadores_2 = filtered_contadores_2.dropna(
    subset=['Nome da Escola'])

valid_edificios = filtered_edificios.dropna(subset
    =['Designa  o'])

tabela_resultado = pd.concat([
    pd.Series(edificios_imoveis['
        designation_imoveis'], name='Nome do
        edificio').to_frame().assign(Flag='I'),
    pd.Series(valid_edificios['Designa  o'], name
        ='Nome do edificio').to_frame().assign(Flag
        ='E'),
    pd.Series(valid_contadores_2['Nome da Escola'],
         name='Nome do edificio').to_frame().assign
        (Flag='C'),
    pd.Series(baze['designa  o'], name='Nome do
        edificio').to_frame().assign(Flag='B'),
    pd.Series(escolas_OSM['Name'], name='Nome do
        edificio').to_frame().assign(Flag='OSM')
], ignore_index=True)
```

Listing 14: Construção de Tabela resultado

3 Desenvolvimento da solução

3.1 Requisitos

Os requisitos foram definidos na primeira reunião feita com o proponente na Câmara Municipal da Maia e ao longo do semestre através de meetings semanais.

3.1.1 Requisitos funcionais

- O sistema deve permitir importar ficheiros excel a partir de um repositório github
- O sistema deve identificar correspondências entre edifícios com base num critério comum
- No final deve ser produzida uma tabela que inclua os nomes dos edifícios conciliados

3.1.2 Requisitos não funcionais

- Este projeto deve ser executável no google Colab
- A tabela final tem de ser distinguível quanto à identificação das tabelas provenientes

3.1.3 Restrições

A autenticação via github no google Colab devido ao repositório ser privado.

3.2 Arquitetura e tecnologias

3.2.1 Arquitetura

A arquitetura desenvolvida seguiu uma abordagem estruturada em diferentes etapas, tendo como principal objetivo a conciliação das tabelas provenientes de fontes diferentes.

O processo iniciou-se com a importação das tabelas fornecidas através do github.

Seguiu-se a recolha de dados adicionais através de scripts que consultaram fontes externas.

Após a importação foi realizada uma etapa para a limpeza e filtragem dos dados, que consistiu na remoção de colunas irrelevantes para facilitar a identificação das colunas-chave que foram usadas como referência no processo de correspondência entre tabelas.

A fase seguinte foi o matching entre tabelas, onde se aplicaram duas abordagens:

- A distância de Levenshtein [5] que permite comparar strings onde a métrica utilizada calcula o número mínimo de inserções, remoções e substituições para as strings se tornarem iguais.
- A fórmula de Haversine [6] que foi usada para calcular a distância em metros entre coordenadas geográficas (latitude e longitude), figura 2.

A solução final consistiu numa tabela resultado que agregou todas as correspondências entre as várias tabelas. Esta tabela inclui os vários nomes que um dado edifício possa ter, estando a conciliação das várias fontes completa.

$$a = \sin^2\left(\frac{\Delta \text{lat}}{2}\right) + \cos(\text{lat}_1) \cdot \cos(\text{lat}_2) \cdot \sin^2\left(\frac{\Delta \text{lon}}{2}\right)$$

$$c = 2 \cdot \text{atan2}\left(\sqrt{a}, \sqrt{1-a}\right)$$

$$d = R \cdot c$$

Figura 2: Formula de Haversine (via Medium)

3.2.2 Tecnologias

A linguagem de programação escolhida foi Python pois é uma linguagem que facilita a manipulação de dados através de módulos que nos ajudaram em todo o processo do projeto.

O trabalho foi desenvolvido em jupyter notebook pois deste modo permite correr o código célula a célula, o que é uma grande vantagem dada a importação das várias tabelas e recolha de dados que só é preciso correr uma vez. Outras razões foram também o suporte de markdown o que permite explicar o código de uma maneira estruturada, e também a facilidade em visualizar os dados e as tabelas.

Utilizamos o google Colab como plataforma para correr o notebook, principalmente pela possibilidade de partilha e colaboração entre os alunos e o instrutor.

Lista de módulos que foram utilizados:

- pandas [7] - o módulo mais importante ao longo do projeto, foi usado para ler tabelas excel usando a função `read_excel()`, e as tabelas foram usadas na estrutura Dataframe, ajudou para fazer a limpeza de células nulas com a função `isna()`, para combinar tabelas de forma eficiente através da função `merge()` e para a construção da tabela final que concatenou todas as tabelas utlizadas para matching usando a função `concat()`

- requests [8] - foi usado para fazer um HTTP request à API do github para importar os ficheiros com as tabelas

- io [9] - foi utilizado o submódulo BytesIO que permitiu carregar diretamente o conteúdo das tabelas para um Dataframe.

- google.colab - foi usado o submódulo userdata para aceder de forma segura aos secrets do Colab permitindo que passwords não ficassem visíveis no código e o submódulo files para permitir a transferência de ficheiro do Colab para o sistema local do utilizador

- ipywidgets [10]- que gera interfaces interativas como botões para dar ao utilizador a opção de fazer a transferência

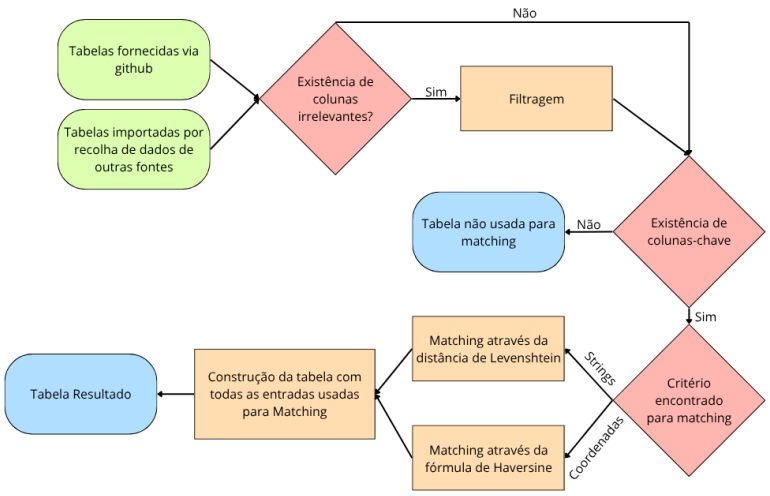


Figura 3: Gráfico de fluxo da Arquitetura

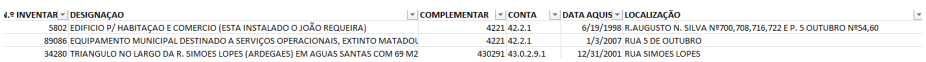
- IPython.display [11] - foram usados os submódulos display e HTML que facilitam a visualização e análise das tabelas pois permitem que estas sejam apresentadas na sua totalidade
- math [12] - foram utilizadas as funções radians, sin, cos, sqrt e atan2 por serem funções trigonométricas que ajudam na implementação da fórmula de Haversine
- rapidfuzz [13] - módulo usado para a comparação de strings onde foi utilizado o submódulo fuzz que fornece várias métricas mas a escolhida foi o `ratio` que calcula a distância de Levenshtein entre duas strings, o número mínimo de inserções, remoções e substituições para estas se tornarem iguais, e o submódulo process através da função `extract()` que retorna as correspondências tendo em conta a métrica usada
- unicodedata [14] - para a normalização dos dados, remoção de acentos e cedilhas

3.3 Solução desenvolvida

A figura 3 identifica a arquitetura que foi usada identificando os conceitos gerais, seguindo-se agora uma explicação detalhada com exemplos nas decisões e processos.

Primeiramente, na decisão de haver ou não colunas irrelevantes, temos um caso de colunas duplicadas na figura 4 onde está representada parte de uma das

tabelas que foi fornecida, onde a coluna COMPLEMENTAR e CONTA têm os mesmo valores.



Nº INVENTAR	DESIGNAÇAO	COMPLEMENTAR	CONTA	DATA AQUIS	LOCALIZAÇÃO
5802	EDIFICIO P/ HABITAÇAO E COMERCIO (ESTA INSTALADO O JOÃO REQUEIRA)	4221	42.2.1	6/19/1998	R.AUGUSTO N. SILVA Nº700,708,716,722 E P. 5 OUTUBRO Nº54,60
89086	EQUIPAMENTO MUNICIPAL DESTINADO A SERVIÇOS OPERACIONAIS, EXTINTO MATADOU	4221	42.2.1	1/3/2007	RUA 5 DE OUTUBRO
34280	TRIANGULO NO LARGO DA R. SIMOES LOPES (ARDEGAES) EM AGUAS SANTAS COM 69 M2	430291	43.0.2.9.1	12/31/2001	RUA SIMOES LOPES

Figura 4: Excerto da tabela imóveis

A filtragem consiste na remoção destas mesmas colunas irrelevantes.

Nas colunas-chave temos um exemplo onde estão presentes as colunas latitude e longitude em duas tabelas como nas figuras 5 e 6.

	designação	longitude	latitude
0	EB Bajouca	-8.636671	41.274883
1	EB Ferreiró	-8.600348	41.273765
2	EB Ferronho	-8.613663	41.278121
3	EB Castelo da Maia	-8.614067	41.259409

Figura 5: Excerto da tabela baze

	Name	Type	Latitude	Longitude
0	Jardim de Infância da Santa Casa da Misericórdia	kindergarten	41.186491	-8.683652
1	Jardim de Infância Fonte dos Dois Amigos	kindergarten	41.186652	-8.686198
2	Infantário Pica Pau Amarelo	kindergarten	41.218561	-8.546001

Figura 6: Excerto da tabela do OpenStreetMaps

Sendo então o critério escolhido coordenadas geográficas é feito o matching através da fórmula de Haversine, figura 7, tendo em conta um threshold adequado para a distância entre os pontos previamente escolhido.

A tabela resultante, figura 8, tem como colunas os nomes dos edifícios e os critérios usados de cada tabela.

A construção da tabela final foi feita a partir de todos os matchings realizados, onde se juntaram todos os valores das tabelas utilizadas como base da tabela.

```
escolas_OSM_matching['key'] = 1
baze['key'] = 1
cross = pd.merge(escolas_OSM_matching, baze, on='key').drop('key', axis=1)

cross['distance_m'] = cross.apply(
    lambda row: haversine(row['Latitude'], row['Longitude'], row['latitude'], row['longitude']),
    axis=1
)

baze_escolas_OSM = cross[cross['distance_m'] <= distance_threshold]

baze_escolas_OSM = baze_escolas_OSM.rename(columns={
    'Name':'Name_escolas_OSM',
    'Latitude':'Lat_escolas_OSM',
    'Longitude':'Lon_escolas_OSM',
    'longitude': 'Lon_baze',
    'latitude':'Lat_baze',
    'designation': 'designação_baze'
})
```

Figura 7: Matching feito por coordenadas

	Name_escolas_OSM	Lat_escolas_OSM	Lon_escolas_OSM	designação	Lon_baze	Lat_baze	distance_m
4253	Escola Básica do 1º Ciclo e Jardim de Infância...	41.229499	-8.614988	EB Cidade Jardim	-8.614799	41.229635	21.963135
4695	Escola Básica do 1º Ciclo e Jardim de Infância...	41.204380	-8.568072	EB Pícua	-8.568378	41.204116	38.947084
4741	Escola Básica de Moutidos	41.211528	-8.570596	EB Moutidos	-8.570920	41.211187	46.645859

Figura 8: Tabela resultante do matching de coordenadas

3.4 Validação

Ao longo do projeto fizemos uma avaliação experimental na qual consistiu em fazer um matching de três tabelas no qual o critério usado foi igual para todos, coordenadas geográficas dado um threshold adequado para a distância (na experiência foi usado um threshold de 70 metros).

Foram testadas as três formas possíveis de matching:

- Caso 1: (Tabela A * Tabela B) * Tabela C
- Caso 2: (Tabela A * Tabela C) * Tabela B
- Caso 3: (Tabela B * Tabela C) * Tabela A

Os resultados foram:

- 51 entradas no matching intermédio, 49 no final
- 54 entradas no matching intermédio, 36 no final
- 45 entradas no matching intermédio, 51 no final

Dada a variação do número de entradas dos três casos, conclui-se que não houve associatividade e isto deve-se ao critério de matching ser a aproximação de coordenadas.

De seguida foi feita uma normalização dos dados onde juntamente com a remoção de acentos, cedilhas e ignorar maiúsculas foi utilizada uma lista de termos após uma análise aos dados que prioriza os termos principais que diferenciam as designações corretamente (ex: Gueifães, Maia, etc) diminuindo os termos gerais (ex: Escola = E, Básica = B, etc), isto para comparar os nomes a fim de identificar os verdadeiros e falsos positivos.

Os resultados foram:

- Caso 1: das 49 entradas, houve 1 falso positivo
- Caso 2: das 36 entradas, não houve falsos positivos
- Caso 3: das 51 entradas, houve 21 falsos positivos

No caso 1, o falso positivo foi entre o edifício "EB Crestins"e "Creche/Pré-escolar de Crestins", pois embora sejam recintos muito próximos são diferentes tipos de ensino logo dois edifícios diferentes.

Já no caso 3, a grande quantidade de falsos positivos deve-se ao facto de uma das tabelas, a última a ser feito o matching, conter valores nulos.

Para a solução final, qualquer pessoa pode servir deste trabalho seja para estudar a metodologia que foi feita seja para verificar que tecnologias foram usadas para a conciliação dos dados.

No entanto, o utilizador principal deste projeto são as pessoas da Câmara Municipal da Maia que forneceram as tabelas, pois realizada esta agregação dos nomes dos edifícios o próximo passo pertence a eles para chegar a um acordo de quais designações usar.

Para a ótica deles, a tabela resultado é constituída por uma coluna que indica todos os nomes dos edifícios de todas as tabelas em que ocorreu matching onde a cor da célula indica qual a tabela de onde são provenientes. Nas colunas que se seguem, para cada tabela existe uma secção de colunas, uma para o nome do edifício e as outras para as colunas-chave que foram usadas para matching. Estas colunas são também identificadas pela cor das tabelas de onde são provenientes respetivamente como na primeira coluna para facilitar a leitura.

4 Conclusões

4.1 Resultados alcançados

Inicialmente os dados continham 730 entradas e após o matching obteve-se um resultado final de 318 entradas referentes a escolas.

O trabalho do grupo foi distribuído da seguinte maneira:

- **Bruno Coutinho Pereira** – responsável pela filtragem, correspondência de strings (string matching) e análise de resultados. (20%)
- **Bruno Rafael Granja Alves** – responsável pela importação e recolha de tabelas, filtragem, correspondência via coordenadas e via strings, análise de resultados e construção da tabela final.(40%)

- **Carlos Ferreira Marta Carreira da Costa** – responsável pela filtragem, correspondência de coordenadas e elaboração da tabela final. (20%)
- **Luís Filipe Gonçalves Fernandes** – responsável pela filtragem, correspondência de coordenadas e análise de resultados. (20%)

4.2 Lições aprendidas

No que diz respeito à colaboração em grupo, vários fatores contribuíram para a aprendizagem coletiva. Ao utilizar ferramentas como o GitHub, o Google Colab e o Jupyter Notebook, foram aplicados os conhecimentos dos membros na criação colaborativa de código. Além disso, o registo de quaisquer avanços no logbook e a realização de reuniões semanais com o proponente, nas quais o seu feedback e sugestões foram dados, proporcionaram uma experiência de trabalho num projeto para um cliente, duma maneira que os típicos projetos da licenciatura não conseguem replicar. De facto, destacam-se diferenças chave entre projetos académicos e projetos em contexto empresarial:

- o facto de não existir um enunciado claro com critérios de avaliação definidos à partida, mas sim um conjunto de requisitos vagos que evoluíram ao longo do tempo;
- a comunicação frequente com stakeholders, como parte do processo iterativo de reuniões regulares e feedback contínuo, o que leva à necessidade de reformular abordagens com base em novas informações e limitações, ao invés de haver uma única submissão para avaliação;
- a primazia da documentação (que em projetos académicos tende a ser secundária) com alto grau de frequência e clareza, para que outros possam entender, utilizar ou continuar o trabalho.

Isto permitiu o desenvolvimento de diversas soft skills, entre as quais:

- **Comunicação eficaz**, essencial para esclarecer dúvidas com o proponente e ter um bom entendimento dos seus objetivos e para coordenar tarefas entre os membros do grupo;
- **Gestão de tempo e organização**, dado o planeamento de tarefas e cumprimento de prazos definidos;
- **Adaptabilidade** ao lidar com ficheiros diferentes, dados incompletos e critérios de matching diferentes;
- **Resolução de problemas**, visto que foram necessárias várias iterações e testes para encontrar abordagens robustas no processo de conciliação de dados;
- **Trabalho em equipa**, crucial para dividir responsabilidades e tarefas e manter eficiência do grupo;

- **Pensamento crítico** na tomada de decisões relativas à análise de resultados e escolha de parâmetros adequados ao matching.

De um ponto de vista técnico, a utilização de Python e de tecnologias adjacentes, como os módulos Pandas, Unicodedata e RapidFuzz, bem como conceitos como as distâncias de Haversine e Levenshtein e a junção de dados permitiram aprofundar os conhecimentos do grupo em análise e manipulação de dados, uma área de grande importância na Engenharia Informática.

Assim, este projeto permitiu adquirir aprendizagens valiosas que serão certamente úteis para o percurso académico e profissional de todos os membros do grupo, tornando-os melhores engenheiros informáticos.

4.3 Trabalho futuro

A quem continuar este trabalho, seja o D4Maia ou colegas nossos numa futura iteração da Projeto Integrador, recomendamos que, a partir do conjunto de técnicas que usamos na conciliação de dados, procurem explorar mais aprofundadamente o problema. Consideramos que há duas maneiras principais de o fazer:

- Aplicar estas técnicas a diferentes áreas do Data Lake, de modo a garantir uma conciliação de dados de maior escala;
- Expandir o conjunto de técnicas, explorando novos métodos para garantir uma conciliação de dados ainda mais fiel à realidade.

Referências

[1] Portal d4maia – data lake municipal. `https://baze.cm-maia.pt`.

[2] Projeto baze – balanço zero de carbono. `https://www.cm-maia.pt/institucional/inovacao-e-desenvolvimento/programas-e-projetos/baze`.

[3] Github - pi2025 repository. `https://github.com/pedroccpimenta/PI2025-DataHarmonization`.

[4] Colab- notebook com o código completo. `https://colab.research.google.com/drive/1bCDrviszfxoA2VWW1zN-e-prRok7Fzmv?usp=sharing`.

[5] Allan Fernando O de Andrade, Adrieli Kethin Santos, Frank Willian C de Oliveira Marcelo, and F Terenciani. Utilizaçao da distância de levenshtein na aproximaçao de palavras em integraçoes de sistemas.

[6] Nitin R Chopde and Mangesh Nichat. Landmark based shortest path detection by using a* and haversine formula. *International Journal of Innovative Research in Computer and Communication Engineering*, 1(2):298–302, 2013.

[7] Wes McKinney. *Python para análise de dados: Tratamento de dados com Pandas, NumPy e IPython*. Novatec Editora, 2018.

[8] Rakesh Vidya Chandra and Bala Subrahmanyam Varanasi. *Python requests essentials*. Packt Publishing Birmingham, 2015.

[9] Python Software Foundation. io — Core tools for working with streams.

[10] Python Software Foundation. ipywidgets: Jupyter interactive widgets.

[11] Cyrille Rossant. *Learning IPython for interactive computing and data visualization*. Packt Publishing Ltd, 2015.

[12] Python Software Foundation. math — Mathematical functions.

[13] Aoshuang Ye, Lina Wang, Lei Zhao, Jianpeng Ke, Wenqi Wang, and Qinliang Liu. Rapidfuzz: Accelerating fuzzing via generative adversarial networks. *Neurocomputing*, 460:195–204, 2021.

[14] Python Software Foundation. unicodedata — Unicode Database.