Universidade do Minho
Escola de Letras, Artes e Ciências Humanas

Análise e Visualização de Dados

Projeto Integrado II

Trabalho Final

Mestrado em Humanidades Digitais

Docentes:

José João Almeida

Álvaro Iriarte Sanromán

Aluno:

Filipe Macedo PG48478

Data:

30 de maio de 2023

Resumo

Este relatório de projeto apresenta detalhadamente uma análise de quatro obras de Camilo Castelo Branco, servindo-se para tal da linguagem de programação *Python*.

Tem como objetivo final a análise e apresentação desses dados extraídos, que incluem pessoas, lugares, organizações, lemas, expressões e palavras-chave. Foi igualmente realizada uma análise de sentimento de cada uma das quatro obras. Os resultados são depois apresentados em listas com o top 10, e transformados em gráficos, também em *Python*, para haver uma maior automatização das tarefas.

Com esta extração, podemos não só obter uma visão do estilo literário do autor, mas também demonstrar o potencial da análise e visualização de textos nas Humanidades Digitais, usando técnicas de processamento de linguagem natural.

Introdução

Camilo Castelo Branco é um dos escritores mais conceituados da literatura portuguesa, sendo reconhecido pela sua vasta obra literária passional, que foi influenciada em grande parte pela sua vida, considerada por muitos como trágica.

Esta análise foca-se em duas obras de Camilo: “A mulher fatal” e “Maria da Fonte”. Através de ferramentas de processamento de linguagem natural, foi possível extrair e apresentar vários tipos de informação de cada obra, o que nos permite ter uma melhor visão destas obras de Camilo.

Metodologia

Como foi referido acima, esta tarefa foi realizada com *Python*. Para tal, foram importadas algumas bibliotecas e módulos para que a análise fosse possível:

```
import spacy
from collections import Counter
from nltk.corpus import stopwords
import string
import re
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
import matplotlib.pyplot as plt
import webbrowser
from pathlib import Path
import pandas as pd

# Load the Portuguese language model
nlp = spacy.load("pt_core_news_sm")

# List of stopwords in Portuguese
stop_words = set(stopwords.words("portuguese"))

# Define the output directory for the generated plots and HTML page
output_dir = Path("output")

def analyze_file(filename):
    text = read_file(filename)
    cleaned_text = clean_text(text)
    doc = nlp(cleaned_text)
```

No final do programa, adicionou se a análise de cada um dos ficheiros desejados, para que estes fossem lidos em *Python*:

```
analyze_file("C:/Filipe/MHD_Laptop/Analise e Viz.
Dados/FinalP/Camilo/Camilo/Obra/Camilo-A_mulher_fatal.txt")
```

Em caso de utilização do programa num computador ou utilizador diferente, para análise e teste do mesmo, o *path* terá de ser alterado manualmente.

I. Extração dos Dados:

- Extração de Entidades:
  - Extração de Pessoas:

```
#Extração de pessoas/personagens presentes nas obras
```

```
def extract_people(doc):
    people = []
    for entity in doc.ents:
        if entity.label_ == "PER":  # Se a entidade for uma pessoa
            people.append(entity.text)  # Adicionar o texto da entidade à
lista
    return Counter(people)  # Retornar um objeto Counter com a contagem das
pessoas
```

Com o objetivo de extrair todas as pessoas mencionadas na obra, criou-se a função *"extract_people"*, que extrai todas as entidades do tipo "PER" (person) dos documentos pretendidos, presentes na biblioteca portuguesa do *spaCy.* Caso estas estejam etiquetadas como pessoas, são anexadas a uma lista e depois devolvidas quando o programa for executado:

- Extração de Lugares:

```
#Extração de lugares
def extract_places(doc):
    places = []
    for entity in doc.ents:
        if entity.label_ == "LOC":  # Se a entidade for um local
            places.append(entity.text)  # Adicionar o texto da entidade à
lista
    return Counter(places)
```

Esta extração é exatamente igual à anterior, mas refere-se aos locais ("LOC").

- Extração de Organizações:

```
#Extração de organizações
def extract_orgs(doc):
    orgs = []
    for entity in doc.ents:
        if entity.label_ == "ORG":  # Se a entidade for uma organização
            orgs.append(entity.text)  # Adicionar o texto da entidade à lista
    return Counter(orgs)
```

Uma vez mais o mesmo passo, referindo-se desta vez às organizações presentes nas obras ("ORG").

- Extração dos Lemas:

```
#Extração dos lemas com as frequências
def extract_lemmas(doc):
    lemmas = []
    for token in doc:
```

```
        if token.is_alpha and not token.is_stop:  # Se o token for uma palavra
e não for uma stop word
            lemmas.append(token.lemma_)  # Adicionar o lemma do token à lista
    return Counter(lemmas)
```

Fazendo uso uma vez mais do *spaCy*, criou-se uma função para extrair os lemas. O código exclui *stop words* e carateres não alfabéticos, filtrando assim os lemas requeridos. Por fim, é devolvida uma lista com os lemas, que mais tarde tem os dados transportados para um gráfico.

- ❖ Extração de MWEs (Multi-Word Expressions):

Com o objetivo de extrair as "*multi-word expressions*" das obras, é criada uma lista que armazena as expressões.

Esta expressão é então devolvida na lista final, como resultado da função:

```
def extract_mwe(doc):
    mwe = []
    for chunk in doc.noun_chunks:
        if len(chunk) > 1:  # Se o chunk tiver mais de uma palavra
            mwe.append(chunk.text)  # Adicionar o texto do chunk à lista
    return Counter(mwe)
```

- ➢ Extração de Palavras-Chave:

Esta função permite filtrar as *stops words* e carateres não alfabéticos, deixando os *tokens* restantes. Estas palavras-chaves permitem uma análise posterior de cada obra.

```
def extract_keywords(doc):
    keywords = []
    for chunk in doc.noun_chunks:
        if not any(token.is_stop for token in chunk) and chunk.root.is_alpha:
            # Se nenhum dos tokens do chunk for uma stop word e a raiz do
chunk for uma palavra
            keywords.append(chunk.text)  # Adicionar o texto do chunk à lista
    return Counter(keywords)
```

- ➢ Extração de Datas:

A função de extração de datas não funcionou corretamente. Consistia em filtrar entidades das obras e extrair apenas as que correspondessem à etiqueta *"DATE"*. Com isto em mente, o objetivo seria extrair essas datas e juntá-las a uma lista.

```
def extract_dates(doc):
```

```
    dates = []
    for entity in doc.ents:
        if entity.label_ == "DATE":  # Se a entidade for uma data
            dates.append(entity.text)  # Adicionar o texto da entidade à lista
    return Counter(dates)
```

- Análise de Sentimento:

A tarefa requeria também uma análise de sentimento de cada obra. Com isto em mente, utilizou-se a biblioteca *"VaderSentiment"*. Esta função assume a obra como *input*, e testa a sua polaridade, para extrair o sentimento geral de cada uma das obras, que é devolvido no final.

```
def sentiment_analysis(text):
    analyzer = SentimentIntensityAnalyzer()
    sentiment_scores = analyzer.polarity_scores(text)
    compound_score = sentiment_scores["compound"]
    if compound_score >= 0.05:
        sentiment = "positive"
    elif compound_score <= -0.05:
        sentiment = "negative"
    else:
        sentiment = "neutral"
    return sentiment
```

- Dados pré-limpeza e estruturação:

De foi criada uma função que permitisse analisar as obras pretendidas. Utilizando o spaCy, ficheiros desejados são lidos e depois são extraídas as entidades desejadas.

```
def analyze_file(filename):
    text = read_file(filename)
    cleaned_text = clean_text(text)
    doc = nlp(cleaned_text)

    people = extract_people(doc)  # Extrair pessoas
    places = extract_places(doc)  # Extrair locais
    orgs = extract_orgs(doc)  # Extrair organizações
    lemmas = extract_lemmas(doc)  # Extrair lemas
    mwe = extract_mwe(doc)  # Extrair MWE (Multi-Word Expressions)
    keywords = extract_keywords(doc)  # Extrair palavras-chave
    dates = extract_dates(doc)  # Extrair datas
    sentiment = sentiment_analysis(cleaned_text)
```

II. Limpeza:

- Tentativas falhadas:

❖ Os dados extraídos apresentaram bastantes erros na filtragem. Foram utilizadas bastantes bibliotecas e módulos diferentes, mas nenhuma funcionou com o sucesso pretendido:

❖ *Fuzzywuzzy*:

Esta biblioteca foi testada para fundir alguns dos nomes presentes no texto, que aparecem no próprio top 10 de alguns dos gráficos. Esta contém um algoritmo que calcula a similaridade entre *strings* das obras, fazendo com que haja uma fusão de entidades para evitar que estas se repitam.

➢ *Stopwords*:

Para tentar filtrar *stopwords* do texto, experimentou-se também a importação de listas completas de *stopwords* portuguesas, como a deste link: https://gist.github.com/alopes/5358189 mas, uma vez mais, não foram obtidos os resultados pretendidos.

III. Visualização:

➢ Exibição dos top 10:

No final, para que houvesse uma maior facilidade de limpeza e visualização dos dados, foi realizada apenas uma extração do top 10 de cada entidade. Estes *prints* incluem um *counter* que extrai apenas o top 10 das entidades desejadas, de forma a simplificar e facilitar a análise:

```
# Imprimir os resultados da análise

print(f"\n===== Análise de {filename} =====")
print("Top 10 Pessoas:")
for person, count in Counter(people).most_common(10):
    print(f"{person}: {count}")
print("\nTop 10 Locais:")
for place, count in Counter(places).most_common(10):
    print(f"{place}: {count}")
print("\nTop 10 Organizações:")
for org, count in Counter(orgs).most_common(10):
    print(f"{org}: {count}")
print("\nTop 10 Lemas:")
for lemma, count in Counter(lemmas).most_common(10):
```

```
        print(f"{lemma}: {count}")
    print("\nTop 10 MWE:")
    for mwe_phrase, count in Counter(mwe).most_common(10):
        print(f"{mwe_phrase}: {count}")
    print("\nTop 10 Palavras-chave:")
    for keyword, count in Counter(keywords).most_common(10):
        print(f"{keyword}: {count}")
    print("\nTop 10 Datas:")
    for date, count in Counter(dates).most_common(10):
        print(f"{date}: {count}")
    print("\nAnálise de Sentimento:")
    print(f"Sentimento geral: {sentiment}")
```

- Exportação para CSV e Excel:

Em forma de consolidar os dados extraídos, foi criada a função para exportar os mesmos dados para Excel e CSV.

```
export_to_csv(people, "Pessoas.csv")
    export_to_csv(places, "Locais.csv")
    export_to_csv(orgs, "Organizacoes.csv")
    export_to_csv(lemmas, "Lemas.csv")
    export_to_csv(mwe, "MWE.csv")
    export_to_csv(keywords, "PalavrasChave.csv")
    export_to_csv(dates, "Datas.csv")
    export_to_excel({
        "Pessoas": Counter(people).most_common(10),
        "Locais": Counter(places).most_common(10),
        "Organizações": Counter(orgs).most_common(10),
        "Lemas": Counter(lemmas).most_common(10),
        "MWE": Counter(mwe).most_common(10),
        "Palavras-chave": Counter(keywords).most_common(10),
        "Datas": Counter(dates).most_common(10)
    }, "ResultadosAnalise.xlsx")

def export_to_csv(data, filename):
    data_with_columns = [(item, count) for item, count in data.items()]
    df = pd.DataFrame(data_with_columns, columns=["Item", "Contagem"])  #
Criar um DataFrame com as colunas "Item" e "Contagem"
    df.to_csv(output_dir / filename, index=False)  # Exportar o DataFrame para
CSV


def export_to_excel(data_dict, filename):
    with pd.ExcelWriter(output_dir / filename) as writer:
```

```
        for sheet_name, data in data_dict.items():
            df = pd.DataFrame(data, columns=["Item", "Contagem"])
```

➢ Criação e Exportação para HTML:

Para facilitar as visualizações dos gráficos é criada uma página HTML que é automaticamente aberta, onde expõe os resultados da análise.

```
    def generate_html_page(sentiment):

    html_content = """
        <html>
        <head>
            <title>Resultados da Análise</title>
            <style>
                body {
                    font-family: Arial, sans-serif;
                    margin: 20px;
                }
                h1 {
                    text-align: center;
                    margin-bottom: 30px;
                }
                h2 {
                    margin-top: 50px;
                    margin-bottom: 10px;
                }
                img {
                    display: block;
                    margin-left: auto;
                    margin-right: auto;
                    margin-top: 20px;
                    max-width: 80%;
                    height: auto;
                }
            </style>
        </head>
        <body>
        """
    html_content += f"<h1>Sentimento Geral: {sentiment}</h1>"  # Incluir o
sentimento geral no título

    image_files = sorted(output_dir.glob("*.png"))  # Obter a lista de
arquivos de imagem PNG ordenados
    for image_file in image_files:
```

```
        plot_title = image_file.stem.replace("_", " ")  # Obter o título do
gráfico a partir do nome do arquivo
        image_path = image_file.relative_to(output_dir)  # Obter o caminho
relativo da imagem em relação ao diretório de saída
        html_content += f"<h2>{plot_title}</h2>"
        html_content += f'<img src="{image_path}" alt="{plot_title}"><br><br>'

    html_content += "</body></html>"

    with open(output_dir / "resultados_analise.html", "w", encoding="utf-8")
as file:
        file.write(html_content)

    webbrowser.open_new_tab(output_dir / "resultados_analise.html")
")
```

➢ Análise de sentimento:

A análise de sentimento foi a única das tarefas realizadas que não teve os dados transportados para gráficos. Ao correr o programa, é apresentada a análise de sentimento geral de cada uma das obras, antes de serem apresentados os gráficos. De seguida, é apresentada a análise de sentimento de cada uma das quatro obras:

❖ "Camilo-A_mulher_fatal":

```
Sentiment Analysis of C:/Filipe/MHD_Laptop/Analise e Viz. Dados/FinalP/Camilo/Camilo/Obra/Camilo-A_mulher_fatal.txt: negativo
```

❖ "Camilo-Maria_da_Fonte":

```
Sentiment Analysis of C:/Filipe/MHD_Laptop/Analise e Viz. Dados/FinalP/Camilo/Camilo/Obra/Camilo-Maria_da_Fonte.txt: positivo
```

➢ Gráficos:

Para apresentar uma leitura final dos dados extraídos com uma maior facilidade, foram também criados gráficos para a sua apresentação:

```
def create_bar_plot(data, title, filename):
    categories = [item[0] for item in data]  # Obter as categorias do gráfico
a partir dos dados fornecidos
    counts = [item[1] for item in data]  # Obter as contagens do gráfico a
partir dos dados fornecidos

    plt.figure(figsize=(10, 6))
```

```
    plt.barh(range(len(categories)), counts, align="center")  # Criar um
gráfico de barras horizontais
    plt.yticks(range(len(categories)), categories)  # Definir os rótulos do
eixo y como as categorias
    plt.xlabel("Frequência")  # Definir o rótulo do eixo x como "Frequência"
    plt.title(title)  # Definir o título do gráfico
    plt.tight_layout()
    plt.savefig(filename)  # Gravar gráfico como imagem
    plt.close()

#
    create_bar_plot(Counter(people).most_common(10), f"Top 10 Pessoas em
{filename}", f"{Path(filename).stem}_pessoas.png")

    create_bar_plot(Counter(places).most_common(10), f"Top 10 Locais em
{filename}", f"{Path(filename).stem}_locais.png")
    create_bar_plot(Counter(orgs).most_common(10), f"Top 10 Organizações em
{filename}", f"{Path(filename).stem}_organizacoes.png")
    create_bar_plot(Counter(lemmas).most_common(10), f"Top 10 Lemas em
{filename}", f"{Path(filename).stem}_lemas.png")
    create_bar_plot(Counter(mwe).most_common(10), f"Top 10 MWE em {filename}",
f"{Path(filename).stem}_mwe.png")
    create_bar_plot(Counter(keywords).most_common(10), f"Top 10 Palavras-chave
em {filename}", f"{Path(filename).stem}_palavraschave.png")
    create_bar_plot(Counter(dates).most_common(10), f"Top 10 Datas em
{filename}", f"{Path(filename).stem}_datas.png")
```

Com o intuito de tornar esta tarefa completamente automatizada em *Python*, seguem todos os gráficos extraídos após correr o programa:

- "Camilo-A_mulher_fatal":

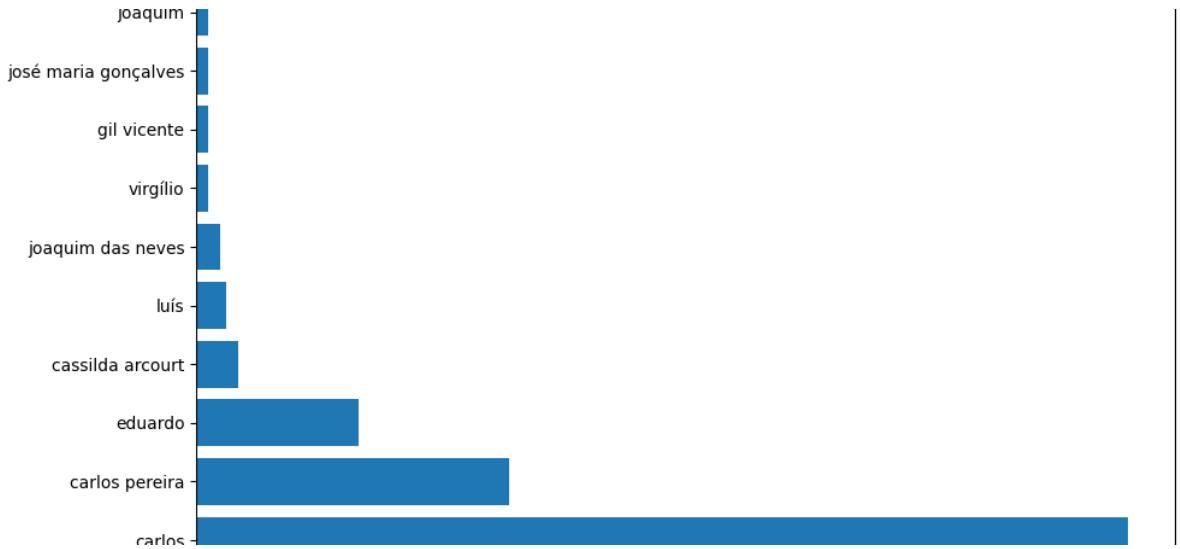

Figura 1-Top10 People

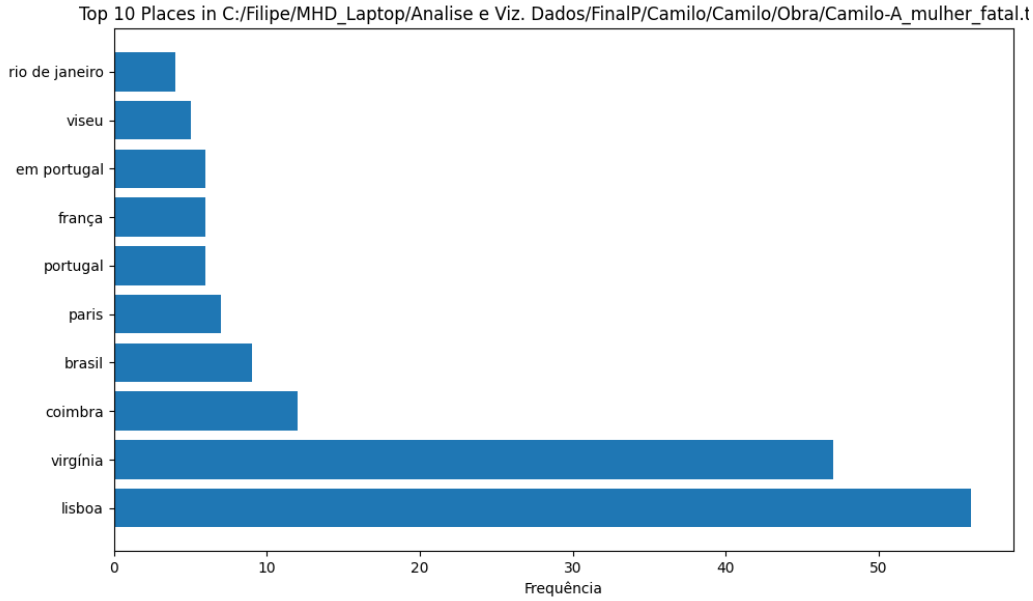

Figura 2-Top10 Places

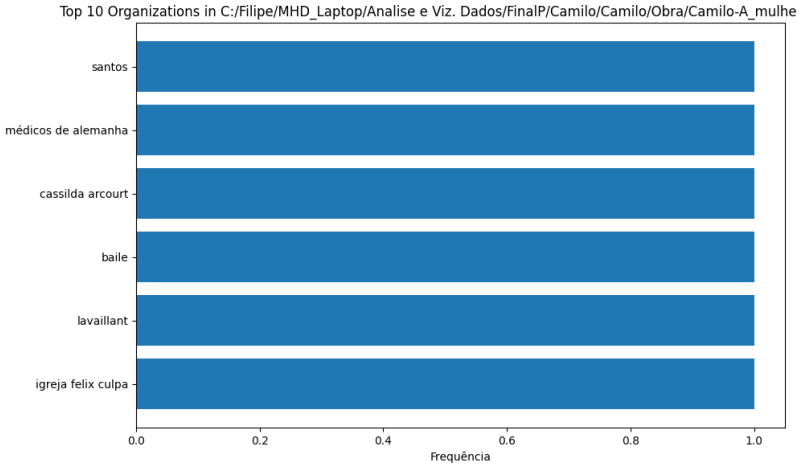

Figura 3-Top10 Orgs

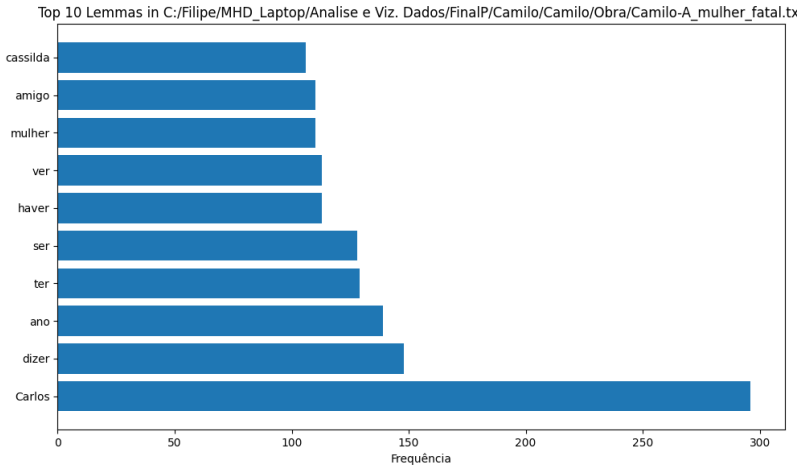

Figura 4-Top10 Lemmas

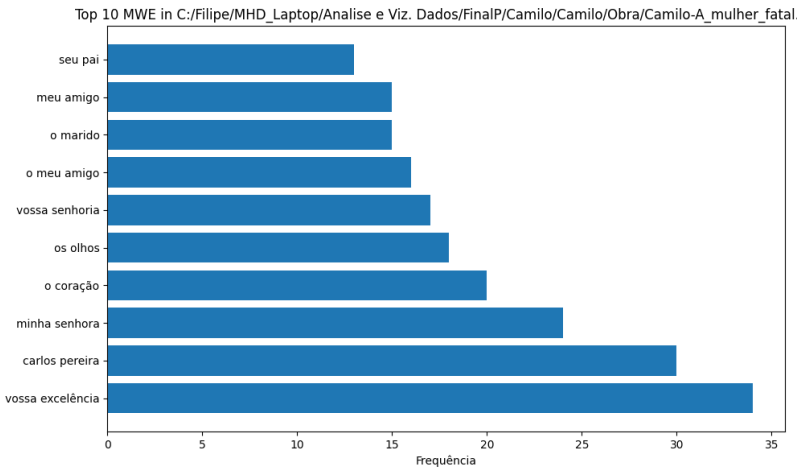

Figura 5-Top10 MWEs

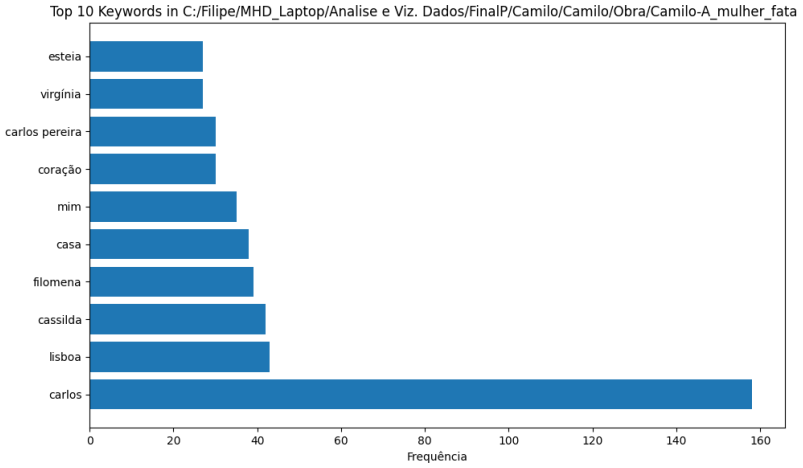

Figura 6-Top10 Keywords

❖ "Camilo-Maria_da_Fonte":

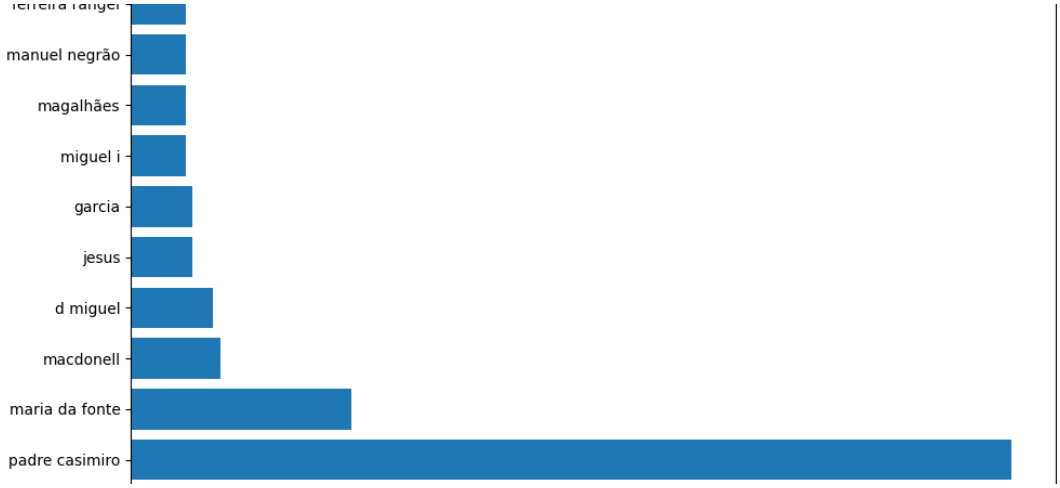

Figura 7-Top10 People

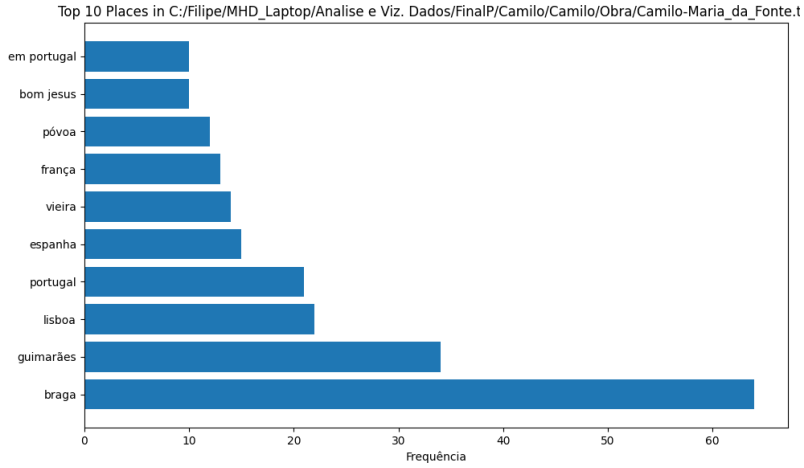

Figura 8-Top10 Places

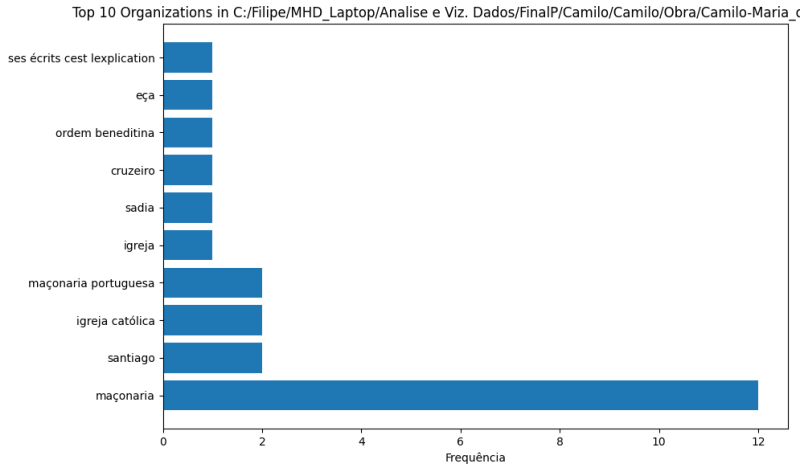

Figura 9-Top10 Orgs

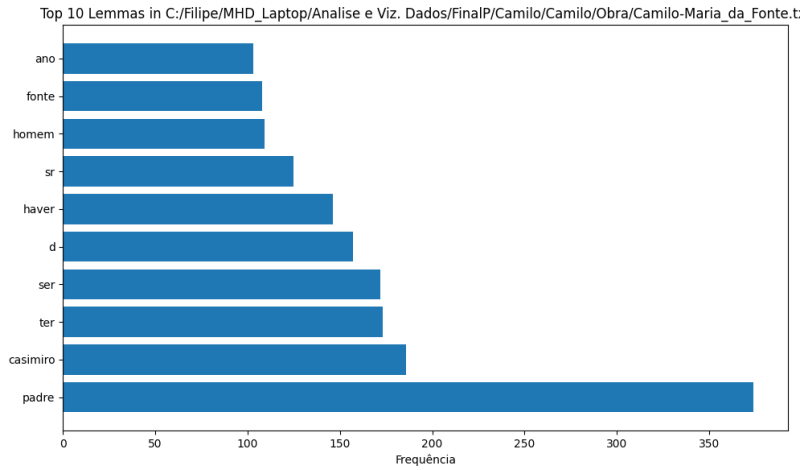

Figura 10-Top10 Lemmas

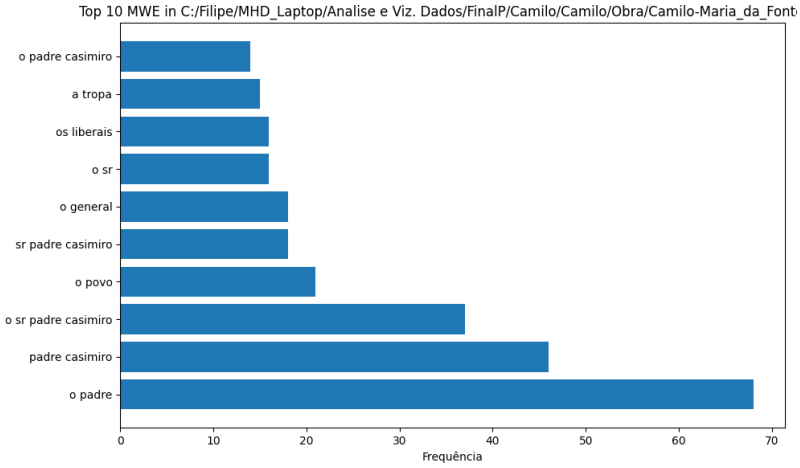

Figura 11-Top10 MWEs

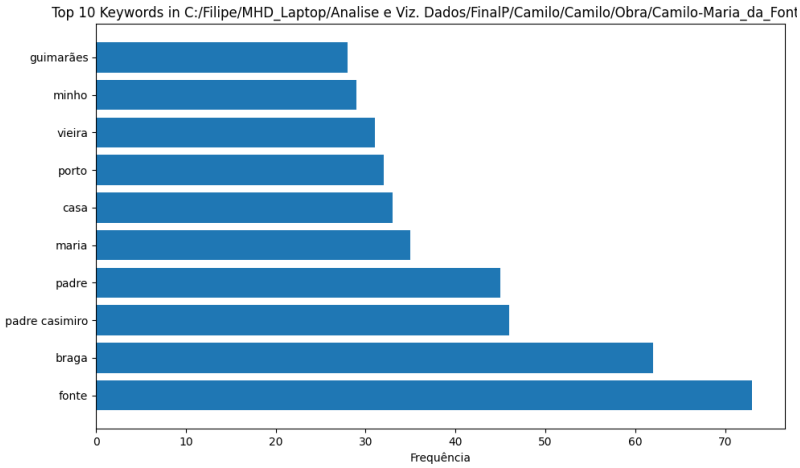

Figura 12-Top10 Keywords

Conclusão

Resumidamente, esta tarefa permitiu analisar com maior exatidão e qualidade algumas das obras de Camilo Castelo Branco, usando várias ferramentas para que isto fosse possível. Utilizando *Python* com modelos e bibliotecas implementados, conseguimos assim ter uma visão mais minuciosa da escrita de Camilo.

Referências

Frazão, D. (2023, April 25). Biografia de Camilo castelo branco. eBiografia. https://www.ebiografia.com/camilo_castelo_branco/

NLTK. (n.d.). https://www.nltk.org/index.html

Simplified text processing. TextBlob. (n.d.). https://textblob.readthedocs.io/en/dev/#

Spacy · industrial-strength natural language processing in python. · Industrial-strength Natural Language Processing in Python. (n.d.). https://spacy.io/

Visualization with python. Matplotlib. (n.d.). https://matplotlib.org/

Hutto, C.J. & Gilbert, E.E. (2014). VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. Eighth International Conference on Weblogs and Social Media (ICWSM-14). Ann Arbor, MI, USA.