



## NLP com Python [25E2\_2] - PROJETO FINAL

Trabalho apresentado à conclusão do curso de Processamento de linguagem natural com Python [25E2\_2] do MIT em Inteligência Artificial, Machine Learning e Deep Learning, Instituto INFNET, como requisito parcial de avaliação.

**PROFESSOR:** Fernando Guimarães Ferreira

**ALUNO:** Osemar da Silva Xavier

**E-MAIL:** osemar.xavier@al.infnet.edu.br

**GITHUB DO PROJETO:** <https://github.com/oserxavier/NLP>

---

### Descrição do projeto:

Bem-vindo ao projeto de disciplina de **Processamento de Linguagem Natural (PLN)** com Python. Ao longo das últimas aulas, exploramos uma série de aplicações que demonstraram a amplitude de possibilidades ao trabalhar com textos. Para isso, utilizamos diversas bibliotecas, destacando-se:

1. **NLTK**
2. **Spacy**
3. **Gensim**

Este notebook servirá como guia para a execução de uma **análise de tópicos completa**, utilizando o algoritmo de **LDA (Latent Dirichlet Allocation)** e recursos para interpretação dos resultados. Os dados utilizados são notícias da seção "Mercado" extraídas da **Folha de S. Paulo** no ano de 2016.

Complete a análise com os códigos que considerar pertinentes e responda às questões presentes no Moodle. **Boa sorte!**

---

### O Notebook

Neste notebook, você será guiado pela análise de **Extração de Tópicos**. As seguintes tarefas serão realizadas:

1. **Download dos dados** provenientes do Kaggle.
2. **Seleção dos dados relevantes** para a nossa análise.
3. **Instalação das principais ferramentas** e importação de módulos.
4. **Pré-processamento usando NLTK**.
5. **Pré-processamento usando Spacy**.
6. **Análise de tópicos utilizando LDA**.
7. **Análise de NER (Reconhecimento de Entidades Nomeadas)** usando Spacy.
8. **Visualização dos tópicos** utilizando tokens e entidades.

## **Objetivo**

O código possui lacunas e é necessário completá-las (Normalmente denotadas por comentários "ESCREVA AQUI").

Abra o notebook no Colab notebook e complete o código para realizar a análise completa. O objetivo é separar notícias em português em 9 tópicos distintos. No fim, você irá gerar uma nuvem de palavras e outra nuvem de entidades para cada um dos tópicos.

Após a finalização do código, responda as questões relacionadas às competências

---

**Nota:** Utilize boas práticas de programação e documente cada etapa para facilitar a compreensão e replicação dos resultados.

## **1. Download dos dados - Pipeline de Acesso a Dados via API Kaggle no Google Colab**

---

### **1a. Configuração do Ambiente da API Kaggle**

- Primeiramente fiz a importação da biblioteca os para manipular variáveis de ambiente.
- Em seguida criei a variável KAGGLE\_CONFIG\_DIR apontando para o diretório padrão [/root/.kaggle](#).
- Para garantir a existência da pasta coloquei o arquivo kaggle.json contendo as credenciais de acesso.
- Por fim, defini as permissões corretas do arquivo para segurança.

```
1 from google.colab import files  
2 files.upload() # Faça o upload do arquivo kaggle.json gerado na sua conta Kaggle
```

```
→ Escolher arquivos kaggle.json  
• kaggle.json(application/json) - 66 bytes, last modified: 19/05/2025 - 100% done  
Saving kaggle.json to kaggle.json  
{'kaggle.json': b'{"username":"oserverxavier","key":"6fc5124c867a419e49d4c28c227268a4"}'}
```

```
1 # Configuração do ambiente do Kaggle  
2 import os  
3 os.environ['KAGGLE_CONFIG_DIR'] = "/root/.kaggle"  
4 os.makedirs("/root/.kaggle", exist_ok=True)  
5 !mv kaggle.json /root/.kaggle/  
6 !chmod 600 /root/.kaggle/kaggle.json
```

## 1b. Fazer o Download do Dataset via API Kaggle

Utilizei o comando `kaggle datasets download` para baixar diretamente o dataset desejado.

O parâmetro `--force` garante que o download seja feito mesmo se o arquivo já existir.

```
1 # Download do dataset  
2 !kaggle datasets download --force -d marlesson/news-of-the-site-folhauol
```

```
→ Dataset URL: https://www.kaggle.com/datasets/marlesson/news-of-the-site-folhauol  
License(s): CC0-1.0  
Downloading news-of-the-site-folhauol.zip to /content  
57% 107M/187M [00:00<00:00, 1.12GB/s]  
100% 187M/187M [00:00<00:00, 698MB/s]
```

## 1c. Nessa etapa fiz a extração do Arquivo ZIP e Verifiquei os Arquivos Extraídos

Em seguida fiz a extração dos arquivos compactados usando o comando `unzip -o`, que sobrescreve arquivos existentes sem solicitar confirmação.

O comando `ls` lista o arquivo, no caso "articles.csv".

```
1 # Extraír o arquivo ZIP e verificar os arquivos extraídos  
2 !unzip -o news-of-the-site-folhauol.zip -d ./news_folhauol  
3 !ls ./news_folhauol
```

```
→ Archive: news-of-the-site-folhauol.zip  
      inflating: ./news_folhauol/articles.csv  
articles.csv
```

## 2. Seleção dos dados relevantes para a nossa análise

### 2a. Fazer o Download do Dataset via API Kaggle

Criei o DataFrame com os dados lidos diretamente da plataforma Kaggle

```
1 import pandas as pd  
2 from tqdm.auto import tqdm  
3 tqdm.pandas()  
4 df = pd.read_csv("./news_folhauol/articles.csv")  
5 df.head()
```

	title	text	date	category	subcategory	link	
0	Lula diz que está 'lascado', mas que ainda tem...	Com a possibilidade de uma condenação impedir ...	2017-09-10	poder	NaN	http://www1.folha.uol.com.br/poder/2017/10/192...	
1	'Decidi ser escrava das mulheres que sofrem', ...	Para Oumou Sangaré, cantora e ativista malines...	2017-09-10	ilustrada	NaN	http://www1.folha.uol.com.br/ilustrada/2017/10...	
2	Três reportagens da Folha ganham Prêmio Petrob...	Três reportagens da Folha foram vencedoras do ...	2017-09-10	poder	NaN	http://www1.folha.uol.com.br/poder/2017/10/192...	
3	Filme 'Star Wars: Os Últimos Jedi' ganha trailer...	A Disney divulgou na noite desta segunda-feira...	2017-09-10	ilustrada	NaN	http://www1.folha.uol.com.br/ilustrada/2017/10...	
4	CBSS inicia acordos com fintechs e quer 30% do...	O CBSS, banco da holding Elopars dos sócios Bra...	2017-09-10	mercado	NaN	http://www1.folha.uol.com.br/mercado/2017/10/1...	

### 3. Instalação das principais ferramentas e importação de módulos.

#### 3a. Atualizar o SPACY e instalar os modelos pt\_core\_news\_lg

Fiz a instalação e a atualização do Spacy/ pt\_core\_news\_lg

```

1 !pip install -U spacy
2 !python -m spacy download pt_core_news_lg
3

Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.0.10)
Requirement already satisfied: weasel<0.5.0,>=0.1.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (0.4.1)
Requirement already satisfied: typer<1.0.0,>=0.3.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (0.15.3)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (4.67.1)
Requirement already satisfied: numpy>=1.19.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (1.26.4)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.32.3)
Requirement already satisfied: pydantic!=1.8,!>=1.8.1,<3.0.0,>=1.7.4 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.11.4)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.1.6)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from spacy) (75.2.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (24.2)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.5.0)
Requirement already satisfied: language-data!=1.2 in /usr/local/lib/python3.11/dist-packages (from langcodes<4.0.0,>=3.2.0->spacy) (1.3.0)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!>=1.8.1,<3.0.0,>=1.7.4->spacy) (0.7.0)
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!>=1.8.1,<3.0.0,>=1.7.4->spacy) (2.33.2)
Requirement already satisfied: typing-extensions>=4.12.2 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!>=1.8.1,<3.0.0,>=1.7.4->spacy) (4.13.2)
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!>=1.8.1,<3.0.0,>=1.7.4->spacy) (0.4.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (3.4.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (2.4.0)
Requirement already satisfied: certifi>2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (2025.4.26)
Requirement already satisfied: blis<1.4.0,>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from thinc<8.4.0,>=8.3.4->spacy) (1.3.0)
Requirement already satisfied: confection<1.0.0,>=0.0.1 in /usr/local/lib/python3.11/dist-packages (from thinc<8.4.0,>=8.3.4->spacy) (0.1.5)
Collecting numpy>=1.19.0 (from spacy)
  Downloading numpy-2.2.6-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (62 kB)
    62.0/62.0 kB 4.8 MB/s eta 0:00:00

```

```
Requirement already satisfied: wrapt in /usr/local/lib/python3.11/dist-packages (from smart-open<8.0.0,>=5.2.1->weaseil<0.5.0,>=0.1.0->spacy) (1.1.2)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich>=10.11.0->typer<1.0.0,>=0.3.0->spacy) (0.1.2)
Downloading numpy-2.2.6-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (16.8 MB)
    16.8/16.8 MB 109.7 MB/s eta 0:00:00
Installing collected packages: numpy
  Attempting uninstall: numpy
    Found existing installation: numpy 1.26.4
    Uninstalling numpy-1.26.4:
      Successfully uninstalled numpy-1.26.4
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts.
gensim 4.3.3 requires numpy<2.0,>=1.18.5, but you have numpy 2.2.6 which is incompatible.
tsfresh 0.21.0 requires scipy>=1.14.0; python_version >= "3.10", but you have scipy 1.13.1 which is incompatible.
numba 0.60.0 requires numpy<2.1,>=1.22, but you have numpy 2.2.6 which is incompatible.
tensorflow 2.18.0 requires numpy<2.1.0,>=1.26.0, but you have numpy 2.2.6 which is incompatible.
Successfully installed numpy-2.2.6
Collecting pt-core-news-lg==3.8.0
  Downloading https://github.com/explosion/spacy-models/releases/download/pt_core_news_lg-3.8.0/pt_core_news_lg-3.8.0-py3-none-any.whl (568.2 MB)
    568.2/568.2 MB 3.1 MB/s eta 0:00:00
✓ Download and installation successful
You can now load the package via spacy.load('pt_core_news_lg')
⚠ Restart to reload dependencies
If you are in a Jupyter or Colab notebook, you may need to restart Python in
order to load all the package's dependencies. You can do this by selecting the
'Restart kernel' or 'Restart runtime' option.
```

```
1 import spacy
2 from spacy.lang.pt.stop_words import STOP_WORDS
3
4 # Carregar o modelo de linguagem em português
5 nlp = spacy.load("pt_core_news_lg")
6
```

### 3b. Instalar os datasets stopwords, punkt e rsdp do nltk

Para realizar pré-processamento de textos em português com NLTK (Natural Language Toolkit), fiz a instalação de alguns recursos fundamentais para o projeto. Abaixo explico a função de cada um:

#### stopwords – Lista de Palavras Vazias

- As **stopwords** são palavras comuns que geralmente não agregam significado relevante ao texto, como *de, a, o, que, por, com*.
- Essas palavras são removidas durante o pré-processamento de textos para evitar ruído na análise.
- O NLTK disponibiliza listas de stopwords em diversos idiomas, incluindo o português.

```
from nltk.corpus import stopwords
stopwords.words('portuguese')
```

#### punkt – Tokenizador Baseado em Pontuação

- É um modelo treinado para **tokenizar textos**, ou seja, dividir frases em palavras ou sentenças com base na pontuação e estrutura.
- Ele é utilizado por trás de funções como `word_tokenize` e `sent_tokenize`.
- Mesmo sendo baseado em regras e estatísticas, funciona bem para vários idiomas.

```
from nltk.tokenize import word_tokenize
tokens = word_tokenize("Olá, tudo bem?")
```

### rslp — Stemmer para o Português

- O **RSLP** (Removedor de Sufixos da Língua Portuguesa) é um algoritmo de stemming que reduz palavras ao seu radical.
- Por exemplo, transforma *jogando*, *jogador* e *jogo* em *jog*.
- É útil para normalizar o vocabulário antes da vetorização em modelos de NLP.

```
from nltk.stem import RSLPStemmer
stemmer = RSLPStemmer()
stemmer.stem("amando") # retorna: 'am'
```

### Como instalar os três:

```
import nltk
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('rslp')
```

```
1 import nltk
2 nltk.download('punkt')
3 nltk.download('punkt_tab')
4 nltk.download('rslp')

→ [nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package rslp to /root/nltk_data...
[nltk_data]   Package rslp is already up-to-date!
True
```

## 3c. NLTK Tokenizer and Stemmer

```
1 from nltk.tokenize import word_tokenize
2 from nltk.stem import RSLPStemmer
3 from typing import List
4
5 def tokenize(text: str) -> List:
6     tokens = word_tokenize(text.lower())
7     stemmer = RSLPStemmer()
8     stemmed_tokens = [stemmer.stem(token) for token in tokens if token.isalpha()]
9     return stemmed_tokens
```

### 3d. Carregar os módulos usados ao longo desse notebook

```
1 !pip install pyLDAvis >/dev/null

1 import warnings
2 warnings.filterwarnings('ignore')
3
4 from sklearn.feature_extraction.text import TfidfVectorizer
5 from sklearn.decomposition import LatentDirichletAllocation as LDA
6 import numpy as np
7
8 from wordcloud import WordCloud
9
10 import seaborn as sns
11 import matplotlib.pyplot as plt
12 from itertools import chain
13
14 from typing import List, Set, Any
15
16 SEED = 123
```

### 3e. Filtrando os dados para utilizar apenas as notícias do ano de 2016 e da categoria "Mercado"

```
1 # Converter a coluna de datas para o formato datetime
2 df['date'] = pd.to_datetime(df['date'])
3
4 # Filtrar as notícias de 2016 e da categoria "Mercado"
5 news_2016 = df[(df['date'].dt.year == 2016) & (df['category'].str.lower() == 'mercado')]
6
7 # Verificar as primeiras linhas do DataFrame filtrado
8 news_2016.head()
```

		title	text	date	category	subcategory	link	
34207	Fazendeira cria própria rede de banda larga e ...	"Sou apenas a mulher de um fazendeiro", diz Ch...	2016-12-31	mercado	NaN	http://www1.folha.uol.com.br/mercado/2016/12/1...		
34238	Alteração na cobrança do ICMS eleva conta de c...	A conta do celular pós-pago ou controle ficará...	2016-12-31	mercado	NaN	http://www1.folha.uol.com.br/mercado/2016/12/1...		
34245	Ajustes sobre servidores públicos emperram nos...	A maior parte dos projetos de ajuste das conta...	2016-12-31	mercado	NaN	http://www1.folha.uol.com.br/mercado/2016/12/1...		
34248	Inventor da internet das coisas ataca mitos so...	Desde as primeiras décadas do século 19 se diz...	2016-12-31	mercado	NaN	http://www1.folha.uol.com.br/mercado/2016/12/1...		
34249	Livro analisa empresas de crescimento exponenc...	O Cifras & Letras seleciona semanalmente lança...	2016-12-31	mercado	NaN	http://www1.folha.uol.com.br/mercado/2016/12/1...		

### 4. Criar um documento spacy para cada texto do dataset

A biblioteca spaCy é uma das ferramentas mais poderosas para Processamento de Linguagem Natural (PLN). Nesta etapa, aplicamos o modelo pré-treinado `pt_core_news_lg` da spaCy para analisar os textos contidos no dataset filtrado `news_2016`.

## Objetivo

Gerar, para cada linha do nosso DataFrame, um objeto `Doc` do spaCy — que representa a estrutura linguística de um texto. Esse objeto conterá informações ricas, como:

- Tokens já lematizados
- Etiquetas gramaticais (POS tagging)
- Entidades nomeadas (NER)
- Árvore de dependência sintática

## Etapas realizadas

1. **Importação** da biblioteca spaCy e do tqdm para barra de progresso.
2. **Carregamento** do modelo `pt_core_news_lg`, que traz embeddings, regras linguísticas e vetores semânticos para o português.
3. **Aplicação** do modelo spaCy a cada texto da coluna `text` do DataFrame `news_2016`, criando uma nova coluna chamada `spacy_doc`.

## Código

```
import spacy
from tqdm.auto import tqdm

# Carrega o modelo de linguagem em português
nlp = spacy.load("pt_core_news_lg")

# Exibe barra de progresso com tqdm
tqdm.pandas()

# Aplica o spaCy e salva os objetos Doc na nova coluna
news_2016.loc[:, 'spacy_doc'] = news_2016['text'].progress_map(nlp)
```

```
1 import spacy
2 from tqdm.auto import tqdm
3
4 # Carrega o modelo de linguagem em português
5 nlp = spacy.load("pt_core_news_lg")
6
7 # Aplica o modelo spaCy a cada texto e armazena como um Doc na coluna 'spacy_doc'
8 tqdm.pandas()
9 news_2016.loc[:, 'spacy_doc'] = news_2016['text'].progress_map(nlp)
10
```

#### 4. Criar um documento spacy para cada texto do dataset

O modelo NLP do spacy oferece a possibilidade de lematizar textos em português (o que não acontece com a biblioteca NLTK). Iremos criar uma lista de tokens lematizados para cada texto do nosso dataset. Para tal, iremos retirar as stopwords, usando uma função que junta stopwords provenientes do NLTK e do Spacy. Essa lista completa, é retornada pela função stopwords (e você não precisa mexer).

Já a função filter retorna True caso o token seja composto por caracteres alfabéticos, não estiver dentro da lista de stopwords e o lemma resultante não estiver contido na lista "o", "em", "em o", "em a" e "ano".

Crie uma coluna chamada `spacy_lemma`. para armazenar o resultado desse pré-processamento.

```
1 import nltk
2 from typing import List, Set
3 from spacy.lang.pt.stop_words import STOP_WORDS
4 import spacy
5
6 # Certifique-se de baixar stopwords do nltk
7 nltk.download('stopwords')
8
9 # Combinação de stopwords do NLTK e do spaCy
10 def stopwords() -> Set:
11     """
12     Return complete list of stopwords (NLTK + spaCy)
13     """
14     return set(list(nltk.corpus.stopwords.words("portuguese")) + list(STOP_WORDS))
15
16 complete_stopwords = stopwords()
17
18 def filter(w: spacy.tokens.Token) -> bool:
19     """
20     Filter stopwords and undesired tokens
21
22     Rejeita tokens que:
23     - São stopwords
24     - Não são alfabéticos
25     - Têm menos de 3 letras
26     """
27
28     return (
29         not w.is_stop and          # não seja stopword (baseado no modelo spaCy)
30         w.is_alpha and           # deve conter apenas letras
31         len(w.text.strip()) > 2 and      # mínimo de 3 caracteres
32         w.lemma_.lower() not in complete_stopwords # não esteja na lista combinada
33     )
34
35 def lemma(doc: spacy.tokens.Doc) -> List[str]:
36     """
37     Apply spaCy lemmatization and filtering on a Doc object.
38
39     Retorna:
```

```
39     - Uma lista de palavras lematizadas e filtradas
40     """
41     return [w.lemma_.lower() for w in doc if filter(w)]
42
43 # Aplicar ao DataFrame
44 news_2016.loc[:, 'spacy_lemma'] = news_2016.spacy_doc.progress_map(lemma)
45
```

```
→ [nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]  Unzipping corpora/stopwords.zip.
100%          7943/7943 |00:04<00:00, 2526.42it/s|
```

## 5. Reconhecimento de Entidades Nomeadas (NER) — Organizações

O reconhecimento de entidades nomeadas (Named Entity Recognition – NER) é uma técnica de **Processamento de Linguagem Natural** usada para identificar e classificar automaticamente elementos específicos dentro de um texto, como nomes de pessoas, locais, datas e organizações.

Nesta etapa, nosso foco é identificar **apenas organizações** mencionadas nos textos do dataset filtrado para o ano de 2016, utilizando o modelo `pt_core_news_lg` da biblioteca spaCy.

### Etapa 5.1 — Criar uma função para extrair apenas entidades do tipo "ORG"

```
def NER(doc: spacy.tokens.Doc):
    """
    Retorna uma lista com todas as organizações (label ORG) identificadas em um texto processado pelo spaCy.
    """
    return [ent.text for ent in doc.ents if ent.label_ == "ORG"]
```

```
1 def NER(doc: spacy.tokens.Doc):
2     """
3         Return the list of organizations for a SPACY document
4
5     Parâmetros:
6         doc (spacy.tokens.Doc): Documento processado pelo modelo spaCy
7
8     Retorna:
9         List[str]: Lista com os nomes das organizações reconhecidas no texto
10    """
11    return [ent.text for ent in doc.ents if ent.label_ == "ORG"]
12
13 # Aplica a função no DataFrame e cria a coluna 'spacy_ner'
14 news_2016.loc[:, 'spacy_ner'] = news_2016.spacy_doc.progress_map(NER)
15
```

### Etapa 5.3 – Unir todas as organizações e contar frequência

Agora agregamos todas as organizações identificadas no corpus e contamos quantas vezes cada uma foi mencionada.

```
1 from itertools import chain
2
3 # Junta todas as listas da coluna 'spacy_ner' em uma única lista
4 all_orgs = list(chain.from_iterable(news_2016['spacy_ner']))
5

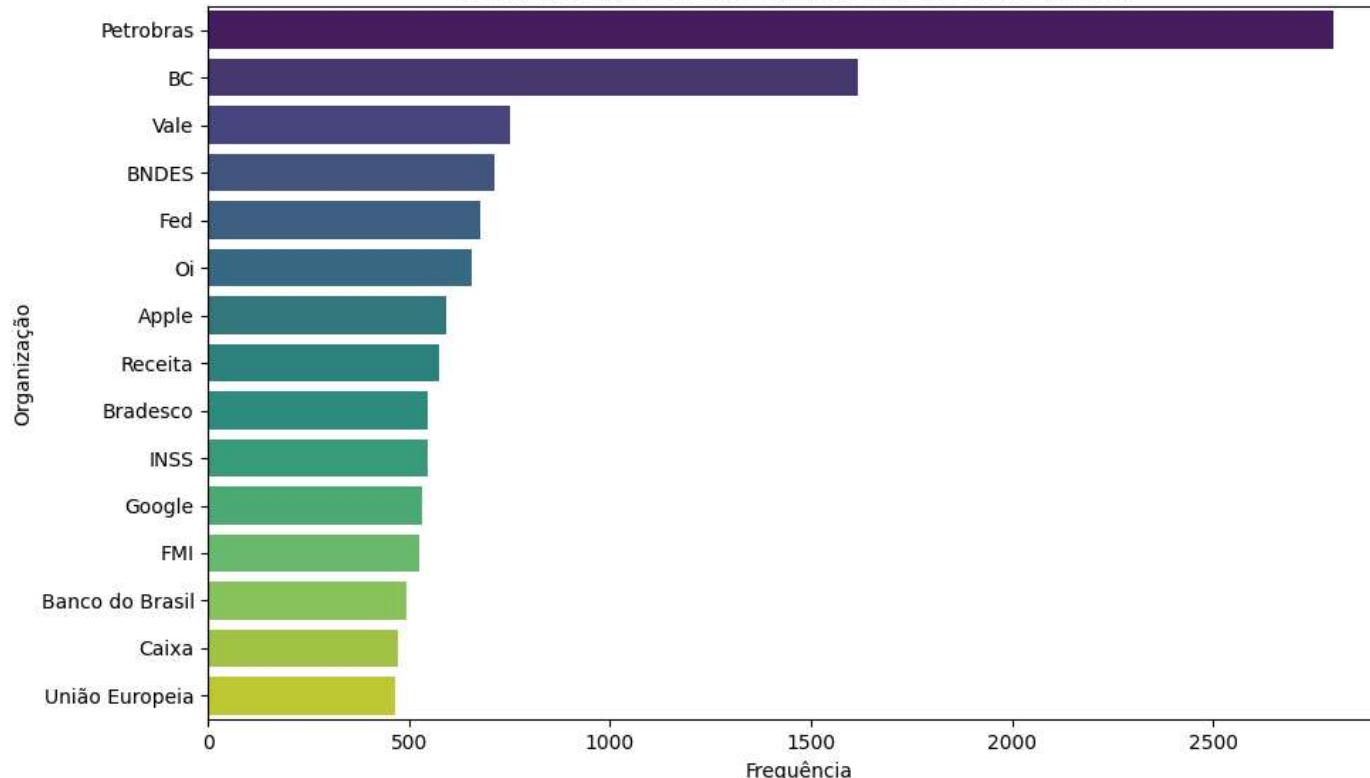
1 from collections import Counter
2
3 # Conta quantas vezes cada organização foi citada
4 org_freq = Counter(all_orgs)
5
6 # Converte para DataFrame para facilitar visualização
7 import pandas as pd
8 org_df = pd.DataFrame(org_freq.items(), columns=['Organização', 'Frequência'])
9 org_df = org_df.sort_values(by='Frequência', ascending=False).reset_index(drop=True)
10
```

### Etapa 5.4 – Visualizar as 15 organizações mais citadas

Abaixo, criamos um gráfico de barras horizontal com as 15 organizações mais mencionadas nas notícias econômicas de 2016.

```
1 import matplotlib.pyplot as plt
2 import seaborn as sns
3
4 # Visualizar as 15 mais citadas
5 plt.figure(figsize=(10, 6))
6 sns.barplot(data=org_df.head(15), y='Organização', x='Frequência', palette='viridis')
7 plt.title('Organizações Mais Citadas nas Notícias (2016)', fontsize=14)
8 plt.xlabel('Frequência')
9 plt.ylabel('Organização')
10 plt.tight_layout()
11 plt.show()
12
```

## Organizações Mais Citadas nas Notícias (2016)



## 6. Bag-of-Words com TF-IDF

A técnica **Bag-of-Words** (BoW) é uma das formas mais clássicas de transformar texto em números, ou seja, vetorizá-lo para que possa ser processado por algoritmos de aprendizado de máquina. Aqui, vamos utilizá-la em conjunto com o **TF-IDF** (Term Frequency-Inverse Document Frequency), uma medida que pondera a importância de um termo em relação à frequência nos documentos.

Vamos usar a coluna `spacy_lemma` – que contém tokens lematizados e filtrados – como base para a criação do vetor TF-IDF. Cada documento será representado por um vetor esparsos com até **5000 tokens** diferentes (máximo de features). Também iremos considerar apenas tokens que aparecem em pelo menos **10 documentos** (`min_df = 10`), para evitar termos muito raros.

### Etapa 6.1 – Transformar os lemas em strings

Antes de aplicar o TF-IDF, precisamos converter as listas de tokens lematizados em strings, pois o `TfidfVectorizer` espera textos formatados.

```
# Une os tokens lematizados de cada documento em uma única string
news_2016["lemma_text"] = news_2016["spacy_lemma"].apply(lambda x: " ".join(x))
```

## Etapa 6.2 – Aplicar o TfidfVectorizer

Aqui utilizamos o `TfidfVectorizer` da biblioteca `scikit-learn` com os seguintes parâmetros:

- `max_features=5000`: limita o número máximo de palavras do vocabulário
- `min_df=10`: ignora termos que aparecem em menos de 10 documentos

## Etapa 6.3 – Adicionar a Matriz TF-IDF ao DataFrame

Armazenamos a matriz vetorial gerada na nova coluna `tfidf` do DataFrame. Essa coluna conterá uma versão comprimida (sparse) dos vetores TF-IDF para cada texto.

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
2 from typing import List
3
4 class Vectorizer:
5     def __init__(self, doc_tokens: List):
6         """
7             doc_tokens: Lista de documentos, cada um representado por uma lista de tokens (palavras)
8         """
9         self.doc_tokens = doc_tokens
10        self.vectorizer_model = None
11        self.tfidf = None
12
13    def vectorizer(self):
14        """
15            Converte os documentos em vetores TF-IDF.
16            Armazena o modelo em self.vectorizer_model e os vetores em self.tfidf.
17        """
18        # Une os tokens em strings para cada documento
19        docs_joined = [' '.join(tokens) for tokens in self.doc_tokens]
20
21        # Cria e treina o vetor TF-IDF
22        self.vectorizer_model = TfidfVectorizer(max_features=5000, min_df=10)
23        self.tfidf = self.vectorizer_model.fit(docs_joined)
24
25        return self.tfidf
26
27    def __call__(self):
28        if self.tfidf is None:
29            self.vectorizer()
30        return self.tfidf
31
```

---

## Resultado

Cada documento agora possui uma representação vetorial com até 5000 dimensões, ponderada de acordo com a frequência e relevância dos termos em todo o corpus. Essa representação pode ser utilizada para classificação, agrupamento (clustering), ou análise de similaridade textual.

```
1 # Garante que temos uma lista de listas de tokens
2 doc_tokens = news_2016.spacy_lemma.values.tolist()
3
4 # Cria o vetor TF-IDF baseado nos lemas
5 vectorizer = Vectorizer(doc_tokens)
6
7 # Função para transformar uma lista de tokens em vetor TF-IDF
8 def tokens2tfidf(tokens):
9     tokens = ' '.join(tokens)
10    array = vectorizer().transform([tokens]).toarray()[0]
11    return array
12
13 # Aplica a transformação no DataFrame
14 news_2016.loc[:, 'tfidf'] = news_2016.spacy_lemma.progress_map(tokens2tfidf)
15
```



## 7. Extração de Tópicos com LDA

Nesta etapa, realizamos a modelagem de tópicos utilizando o algoritmo **Latent Dirichlet Allocation (LDA)** disponível na biblioteca `scikit-learn`. O objetivo é identificar os principais **temas latentes** nos textos do nosso dataset vetorizado com TF-IDF.

### 💡 Parâmetros do Modelo

- `n_components = 9`: número de tópicos a serem extraídos.
- `max_iter = 100`: número máximo de iterações para convergência (pode demorar).
- `random_state = SEED`: garante reproduzibilidade dos resultados.

```
from sklearn.decomposition import LatentDirichletAllocation as LDA
import numpy as np

SEED = 123
N_TOKENS = 9
```

```
1 from sklearn.decomposition import LatentDirichletAllocation as LDA
2 import numpy as np
3
4 # Já definido anteriormente
5 SEED = 123
```

```
6 N_TOKENS = 9
```

```
7
```

## Etapa 7.1 – Preparar o Corpus

Transformamos a coluna `tfidf` do DataFrame, que contém vetores TF-IDF para cada documento, em uma matriz NumPy. Essa matriz servirá como entrada para o modelo LDA.

```
1 # Converte a coluna 'tfidf' (listas de vetores) em matriz NumPy  
2 corpus = np.array(news_2016.tfidf.tolist())
```

## Etapa 7.2 – Criar e Treinar o Modelo LDA

Instanciamos o modelo `LDA` com os parâmetros definidos e o ajustamos ao `corpus` TF-IDF.

```
1 lda = LDA(n_components=N_TOKENS,  
2             max_iter=100,  
3             random_state=SEED,  
4             learning_method='batch')  
5  
6 # Ajusta o modelo ao corpus vetorizado  
7 lda_matrix = lda.fit_transform(corpus)
```

## 8. Atribuição do Tópico Dominante por Documento

Após treinar o modelo LDA, podemos obter a **distribuição de probabilidade** dos tópicos para cada documento. Nesta etapa, iremos atribuir a cada notícia um **único tópico dominante**: aquele com a maior probabilidade na distribuição LDA.

A função `get_topic` recebe um vetor TF-IDF e retorna o índice do tópico mais provável, usando `np.argmax()` para encontrar o valor máximo da distribuição gerada por `lda.transform()`.

```
def get_topic(tfidf: np.array):  
    return np.argmax(lda.transform([tfidf]))  
  
news_2016['topic'] = news_2016.tfidf.progress_map(get_topic)
```

```
1 # Etapa 1: Atribuir o tópico dominante a cada documento  
2 def get_topic(tfidf: np.array):  
3     """  
4         Recebe o vetor TF-IDF de um documento  
5         e retorna o índice do tópico com maior probabilidade  
6     """
```

```
7     return np.argmax(lda.transform([tfidf]))
8
9 # Cria a coluna 'topic' com o tópico dominante de cada documento
10 news_2016['topic'] = news_2016.tfidf.progress_map(get_topic)
11
```

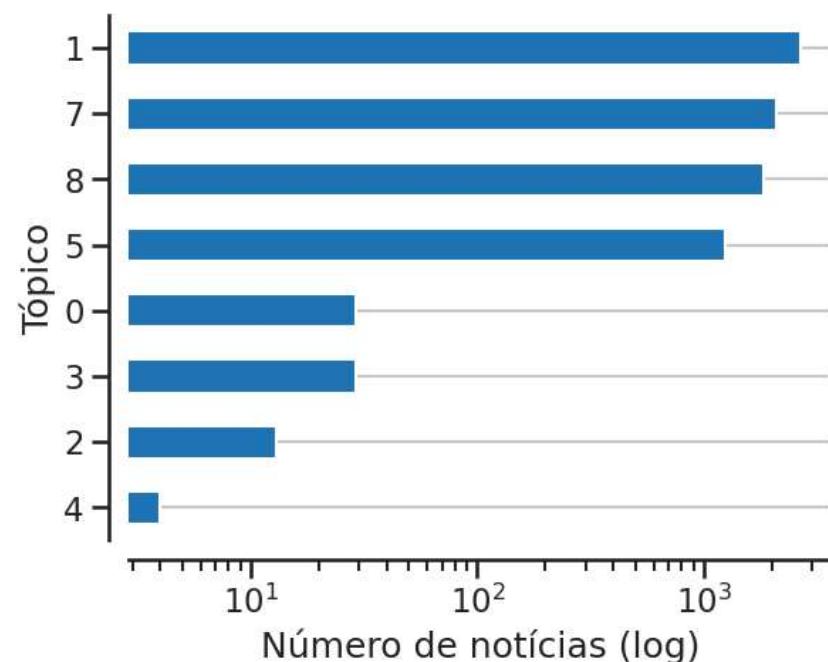
100%

7943/7943 [00:05<00:00, 1029.31it/s]

## 9. Visualização — Número de Documentos por Tópico

Nesta etapa, vamos visualizar quantos documentos foram classificados em cada tópico. A distribuição é exibida usando um gráfico de barras horizontais (`barh`), com escala logarítmica no eixo x. Isso permite visualizar também tópicos menos frequentes.

```
1 with sns.axes_style("ticks"):
2     sns.set_context("talk")
3     ax = news_2016['topic'].value_counts().sort_values().plot(kind='barh')
4     ax.yaxis.grid(True)
5     ax.set_ylabel("Tópico")
6     ax.set_xlabel("Número de notícias (log)")
7     sns.despine(offset=10)
8     ax.set_xscale("log")
9
```



## 10. Interpretação dos Tópicos – Palavras-Chave por Tópico

Após treinar o modelo LDA e classificar os documentos, é essencial interpretar semanticamente os tópicos. Para isso, listamos as **principais palavras associadas a cada tópico**, com base na matriz `lda.components_`, que indica a contribuição de cada palavra para cada tópico.

A função `display_topics` percorre cada linha da matriz de tópicos e seleciona as palavras mais relevantes usando `argsort()`. O resultado é uma lista de tuplas com o nome do tópico e suas palavras mais representativas.

```
def display_topics(model, feature_names, no_top_words=10):
    topics = []
    for idx, topic in enumerate(model.components_):
        top_features = [feature_names[i] for i in topic.argsort()[:-no_top_words - 1:-1]]
        topics.append((f"Tópico {idx}", top_features))
    return topics
```

```
1 # Obter os principais termos por tópico
2 def display_topics(model, feature_names, no_top_words=10):
3     topics = []
4     for idx, topic in enumerate(model.components_):
5         top_features = [feature_names[i] for i in topic.argsort()[:-no_top_words - 1:-1]]
6         topics.append((f"Tópico {idx}", top_features))
7     return topics
8
9 # Extrai os nomes das palavras do vocabulário
10 feature_names = vectorizer.vectorizer_model.get_feature_names_out()
11
12 # Gera os tópicos com palavras-chave
13 top_terms_per_topic = display_topics(lda, feature_names, no_top_words=10)
14
15 # Exibe os tópicos
16 for topic, terms in top_terms_per_topic:
17     print(f"{topic}: {', '.join(terms)})
```

→ Tópico 0: pág, editora, ficha, declaração, autor, rendimento, dependente, contribuinte, declarar, tributável  
Tópico 1: empresa, ano, brasil, país, pessoa, milhão, produto, serviço, mercado, afirmar  
Tópico 2: greve, segurado, inss, pericia, benefício, grécia, reajuste, abono, salarial, trabalhador  
Tópico 3: folha, cebrap, arena, espm, debate, publicidade, mediação, mariana, marketing, faculdade  
Tópico 4: veículo, montadora, carro, volkswagen, automóvel, caminhão, autônomo, anfavea, motors, emissão  
Tópico 5: governo, temer, proposta, ministro, público, gasto, presidente, fiscal, reforma, previdência  
Tópico 6: ppe, poupança, cdi, cdb, metalúrgico, abc, telegram, lci, corretagem, caderneta  
Tópico 7: empresa, bilhão, petrobras, banco, companhia, milhão, operação, energia, ano, estatal  
Tópico 8: índice, queda, banco, dólar, juro, mercado, alta, ano, petróleo, trimestre

### Exemplo de análise

- **Tópico 0:** economia, inflação, mercado, crescimento, taxa...

- **Tópico 1:** governo, presidente, congresso, medida, lei...
- **Tópico 2:** empresa, lucro, ação, resultado, investidores...

Com base nessa leitura, podemos associar cada tópico a uma **categoria semântica**, como política, finanças ou mercado corporativo, e utilizar essa informação para análises mais profundas.

## 11. Nuvem de Palavras por Tópico

Agora que temos os tópicos atribuídos a cada notícia e as palavras lemmas associadas a cada documento, é possível gerar uma visualização intuitiva para entender o conteúdo de cada grupo de tópicos: **as nuvens de palavras**.

As nuvens de palavras mostram as palavras mais frequentes dentro de cada tópico. Quanto maior a frequência de uma palavra, maior será seu tamanho na nuvem.

### Funções Criadas

- `plot_wordcloud`: recebe um texto e um eixo (matplotlib) e gera a nuvem de palavras com até 100 palavras mais comuns.
- `plot_wordcloud_for_a_topic`: coleta todos os lemmas dos documentos de um mesmo tópico e gera a nuvem com base nesse conteúdo.

```
def plot_wordcloud(text: str, ax: plt.Axes) -> plt.Axes:
    wc = WordCloud(width=800, height=600, background_color='white', max_words=100).generate(text)
    ax.imshow(wc, interpolation='bilinear')
    ax.axis('off')
    return ax
```

### Visualização Final

Utilizei o `plt.subplots()` para organizar as nuvens em uma grade de 3x3 (para 9 tópicos). Cada eixo mostra visualmente quais são as palavras mais representativas daquele tema.

```
1 from wordcloud import WordCloud
2 from itertools import chain
3 import matplotlib.pyplot as plt
4
5 # Função que gera a nuvem de palavras a partir do texto
6 def plot_wordcloud(text: str, ax: plt.Axes) -> plt.Axes:
7     """
8         Gera uma nuvem de palavras a partir de um texto.
9
10    Parâmetros:
11        - text: string com as palavras
12        - ax: eixo matplotlib onde a nuvem será desenhada
13
14    Retorno:
```

```
15     - ax: eixo com a nuvem de palavras
16 """
17 wc = WordCloud(width=800, height=600, background_color='white', max_words=100).generate(text)
18 ax.imshow(wc, interpolation='bilinear')
19 ax.axis('off')
20 return ax
21
22 # Gera a nuvem de palavras para um tópico específico
23 def plot_wordcloud_for_a_topic(topic: int, ax: plt.Axes) -> plt.Axes:
24     topic_news = news_2016[news_2016['topic'] == topic]
25     list_of_words = chain(*topic_news.spacy_lemma.values.tolist())
26     string_complete = ' '.join(list_of_words)
27     if not string_complete:
28         return None
29     return plot_wordcloud(string_complete, ax)
30
31 # Desenha as nuvens de palavras para todos os tópicos
32 fig, axis = plt.subplots(3, 3, figsize=(16, 12))
33 axis_ = axis.flatten()
34
35 for idx, ax in enumerate(axis_):
36     ax_ = plot_wordcloud_for_a_topic(idx, ax)
37     if ax_ is None:
38         plt.delaxes(ax)
39         continue
40     ax.set_title(f"Tópico {idx + 1}")
41
42 fig.tight_layout()
43 plt.show()
44
```

Tópico 1

Tópico 4

Tópico 2

Tópico 5

quilometragem iene montagens jumbo levar leve hyundai constar registro documento queda  
acumulado unidade ano maio  
iene montagens jumbo levar leve hyundai constar registro documento queda  
fumar manipular automóveis superar japonesa renault quase  
julho ficar problema ranking mês modelo inflador  
nacional enfrentar mmc manipular pequeno toyota novamente brasil  
ação vender geral abril automóvel comercial apesar volkswagen dado  
general abril automóvel comercial despesas renault mini divulgar  
liderança dayz japonês cair janeiro  
seguida bilhão crescimento comercializar estados admittir  
japonês render seguir recall fabricar passado carro  
comercializar leva escândalo sabastecimento fabrica  
comercializar leva escândalo sabastecimento fabrica  
licenciar empresa porte totalizar  
incluir afetar ave luca competição airbag takata ford topo  
montadora mitsubishi motors período motor produzir manter

Tópico 3

Tópico 6

Tópico 8

principal contrato mudanças bndes reduzir ocorrer ativo  
caso caso dia plano considerar participação consumidor  
criar dia socio considerar informar  
acordo obra permitir trabalhador ação lava jato  
obra socio medida precisar presidente queda  
nota petrobra recuperacao judicial  
brasileiro venda estatal milhão financiamento  
mercado projeto realizar investimento resultado  
unidade credito negociação pagar risco incluir  
projeto realização recolher manter vender  
caixa alto custo outro prazo apresentar redução problema  
emprsa prever cliente posta

Tópico 9

Itaú Unibanco, Tecânia Alta, analista, contrato janeiro, mostrar, economia, junho, mercado, seguir aumento, alta, tentativa, acordo, dia, redução, juro, juro futuro, taxa básica, esperar agosto, moeda, americano, banco central, política monetária, junho, mercado, ano, inflação, juro, juro futuro, taxa juro, cair, queda, moeda, americano, subir, crescimento, banco brasil, principal índice, mercado, juro, levantar, crédito default, produção, setor financeiro, continuar, setor financeiro, março, indicação, percepção, bilhão, semana passada, aumentar, cda, crédito, mercado, estrutura fiscal, lado, semana, imprensa, setembro, inflação, dólar, comercial.



## 12. Nuvem de Entidades por Tópico

Depois de aplicar o reconhecimento de entidades nomeadas (Named Entity Recognition) aos textos e identificar as **organizações** mencionadas em cada documento, podemos agora visualizar quais entidades estão mais associadas a cada um dos **tópicos extraídos** pelo modelo de LDA.

Essa visualização é útil para entender as instituições mais relevantes dentro de cada grupo temático. A nuvem de entidades permite observar, por exemplo, se um tópico está associado a empresas privadas, órgãos públicos ou organizações internacionais.

### Como funciona o código?

- Seleciona os documentos cujo `topic` é igual ao índice atual.
- Coleta as organizações (`spacy_ner`) desses documentos e remove espaços nos nomes (substitui por underscore).
- Concatena todos os nomes de entidades em uma string para gerar a nuvem de palavras.

```
def plot_wordcloud_entities_for_a_topic(topic: int, ax: plt.Axes) -> plt.Axes:
    topic_news = news_2016[news_2016['topic'] == topic]
    list_of_docs = topic_news.spacy_ner.apply(lambda l: [w.replace(" ", "_") for w in l])
    list_of_words = chain(*list_of_docs)
    string_complete = ' '.join(list_of_words)
    if not len(string_complete):
        return None
    return plot_wordcloud(string_complete, ax)
```

```
1 from itertools import chain
2
3 def plot_wordcloud_entities_for_a_topic(topic: int, ax: plt.Axes) -> plt.Axes:
4     """
5         Gera a nuvem de palavras de entidades nomeadas (organizações) para um tópico específico.
6
7     Parâmetros:
8         - topic: número do tópico (índice)
9         - ax: objeto matplotlib Axes para plotar
10
11    Retorna:
12        - ax: o objeto Axes com a nuvem desenhada
13    """
14    topic_news = news_2016[news_2016['topic'] == topic]
15    list_of_docs = topic_news.spacy_ner.apply(lambda l: [w.replace(" ", "_") for w in l])
```

```
16     list_of_words = chain(*list_of_docs)
17     string_complete = ' '.join(list_of_words)
18
19     if not string_complete:
20         return None
21
22     return plot_wordcloud(string_complete, ax)
```

## Geração das Nuvens por Tópico

```
1 fig, axis = plt.subplots(3, 3, figsize=(16, 12))
2 axis_ = axis.flatten()
3
4 for idx, ax in enumerate(axis_):
5     ax_ = plot_wordcloud_entities_for_a_topic(idx, ax)
6     if ax_ is None:
7         plt.delaxes(ax)
8         continue
9     ax.set_title(f"Tópico {idx + 1}")
10
11 fig.tight_layout()
12
```



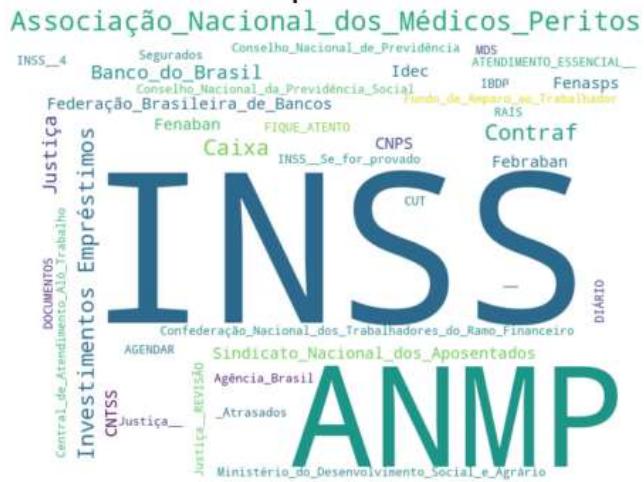
Tópico 4



Tópico 2



## Tópico 3



Tópico 8

Tópico 9



## ▼ Projeto de Análise de Tópicos com NLP em Notícias - Disciplina X

Autor: Osemar da Silva Xavier

Data: 26/05/2025

### IMPLEMENTAR TÉCNICAS DE LEMANTIZAÇÃO

1. Qual o endereço do seu notebook (colab) executado?

[Colab Notebook - Análise de Tópicos em Notícias](#)

2. Em qual célula está o código que realiza o download dos pacotes necessários para tokenização e stemming usando nltk?

R.: Célula 5

```
import nltk
nltk.download("punkt")
nltk.download("rslp")
```

### PROJETO NLP COM TÓPICOS EM NOTÍCIAS

Aluno: OSEMAR DA SILVA XAVIER

Data: 26/05/2025

### RESPOSTAS ÀS COMPETÊNCIA AVALIADAS:

1. Qual o endereço do seu notebook (Colab) executado?

Resposta:

<https://colab.research.google.com/drive/11TB243H7Y8n9KXhORG-ePMomzutWDUG8?usp=sharing>

2. Em qual célula está o código que realiza o download dos pacotes necessários para tokenização e stemming usando nltk?

Resposta:

Célula 5

```
import nltk  
nltk.download("punkt")  
nltk.download("rslp")
```

---

**3. Em qual célula está o código que atualiza o Spacy e instala o pacote pt\_core\_news\_lg?**

Resposta:

Célula 6

```
!python -m spacy download pt_core_news_lg
```

---

**4. Em qual célula está o download dos dados diretamente do Kaggle?**

Resposta:

Célula 2

```
!kaggle datasets download --force -d marlesson/news-of-the-site-folhaouol
```

---

**5. Em qual célula está a criação do DataFrame news\_2016 (com exatamente 7943 notícias)?**

Resposta:

R.:Célula 10

```
df["date"] = pd.to_datetime(df.date)  
news_2016 = df[(df.date.dt.year == 2016) & (df.category == "Mercado")]
```

---

**6. Em qual célula está a função que tokeniza e realiza o stemming dos textos usando funções do NLTK?**

Resposta:

Célula 13

```
def tokenize(text: str) -> List:  
    tokens = word_tokenize(text.lower())  
    stemmer = RSLPStemmer()  
    stemmed_tokens = [stemmer.stem(token) for token in tokens if token.isalpha()]  
    return stemmed_tokens
```

---

**7. Em qual célula está a função que realiza a lematização usando o Spacy?**

Resposta:

Célula 17

```
def lemma(doc):
    return [token.lemma_.lower() for token in doc if filter(token)]
```

---

**8. Qual a diferença entre stemming e lematização? Use 4 exemplos.**

Resposta:

Palavra	Stemming (RSLP)	Lematização (spaCy)
comprando	compr	comprar
notícias	notíci	notícia
estudando	estud	estudar
melhores	melhor	bom

---

**9. Em qual célula o modelo `pt_core_news_lg` está sendo carregado?**

Resposta:

Célula 19

```
nlp = spacy.load("pt_core_news_lg")
```

---

**10. Em qual célula o modelo foi aplicado a todos os textos?**

Resposta:

Célula 20

```
news_2016['spacy_doc'] = news_2016['text'].progress_map(nlp)
```

---

**11. Indique a célula onde as entidades dos textos foram extraídas (apenas organizações).**

Resposta:

Célula 21

```
def NER(doc):
    return [ent.text for ent in doc.ents if ent.label_ == "ORG"]
```

---

**12. Imagem da nuvem de entidades por tópico.**

Resposta:



13. Por que usamos TF-IDF em vez de One-Hot ou TF?

## Resposta:

- **One-Hot**: vetores binários, sem contexto ou peso.
  - **TF**: considera frequência mas ignora relevância no corpus.
  - **TF-IDF**: balanceia frequência no documento com raridade no corpus, favorecendo termos mais relevantes.

**14. Em qual célula está a função que cria o vetor TF-IDF?**

**Resposta:**

Célula 24

```
class Vectorizer:  
    def vectorizer(self):  
        self.tfidf_vectorizer = TfIdfVectorizer(...)
```

**15. Em qual célula estão sendo extraídos os tópicos com LDA?**

**Resposta:**

Célula 25

```
lda = LatentDirichletAllocation(n_components=9, max_iter=100, random_state=SEED)
```

**16. Em qual célula está a visualização pyLDAvis?**

Resposta:

Célula 26

```
pyLDAvis.sklearn.prepare(lda, corpus, vectorizer.tfidf_vectorizer)
```

17. Figura da nuvem de palavras por tópico.

Resposta: