

## ✓ Laboratório 2

Este laboratório tem como objetivo colocar em prática principalmente os tópicos vistos na aula 5.

No livro da disciplina, você poderá encontrar mais informações sobre os assuntos nos seguintes capítulos:

- Alguns comentários sobre Bag of words: [Cap 4 - SLP Book](#)
- Semântica vetorial, embeddings, similaridade com cosseno e TF-IDF: [Cap 6 - SLP Book](#)

### Tópicos:

0. [Bibliotecas](#)
1. [Representação Bag-of-words](#)
2. [Term Frequency - Inverse Document Frequency \(TF-IDF\)](#)
3. [Similaridade com cosseno](#)

## ✓ Bibliotecas

[Voltar ao topo](#)

```
# Bibliotecas utilizadas em visualização de dados
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Bibliotecas para NLP
import nltk # https://www.nltk.org/
import re
import math
```

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

nltk.download('punkt_tab')
```

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

[texto do link](#)

## Representação Bag-of-words

[Voltar ao topo](#)

### ✓ TAREFA 1:

#### BoW

1. Pre-processar o texto abaixo (apenas tokenização, não precisa remover stopwords).
2. Criar uma lista de vocabulário (conjunto de todas as palavras únicas do texto).
3. Representar cada frase como um vetor de tamanho N (sendo N o tamanho do vocabulário) com cada elemento sendo a contagem de quantas palavras existem daquele mesmo índice no vocabulário.

Perceba que, agora, uma das intuições sobre como podemos comparar a similaridade entre frases é pela distância vetorial entre elas.

> TEXTO:

(...)

And as I sat there brooding on the old, unknown world, I thought of Gatsby's wonder when he first picked out the green light at the end of Daisy's dock. He had come a long way to this blue lawn, and his dream must have seemed so close that he could hardly fail to grasp it. He did not know that it was already behind him, somewhere back in that vast obscurity beyond the city, where the dark fields of the republic rolled on under the night.

Gatsby believed in the green light, the orgastic future that year by year recedes before us. It eluded us then, but that's no matter—tomorrow we will run faster, stretch out our arms further... And one fine morning

So we beat on, boats against the current, borne back ceaselessly into the past.

[The Great Gatsby\\_por F. Scott Fitzgerald](#)

[ ] ↪ 1 cell hidden

## ✓ [Teste] Corpus com gatinhos e cachorrinhos

Frases menores primeiro para validar o método

```
Documento_1 = "0 gato correu rápido."
Documento_2 = "0 cachorro correu devagar."
Documento_3 = "0 gato e o cachorro correram."
```

possui o vocabulário de tamanho 8

```
Vocabulário = ["0", "gato", "correu", "rápido", "cachorro", "devagar", "e", "correram"]
```

cujos vetores são:

```
d1 (Documento 1) = [1, 1, 1, 1, 0, 0, 0, 0]
d2 (Documento 2) = [1, 0, 1, 0, 1, 1, 0, 0]
d3 (Documento 3) = [2, 1, 0, 0, 1, 0, 1, 1]
```

## Texto retirado dos slides

```
texto = "0 gato correu rápido." + "0 cachorro correu devagar." + "0 gato e o cachorro"
```

# Corpus

```
corpus = [frase for frase in texto.split('.')]
corpus = [frase for frase in corpus if frase != '']
corpus = [frase.lower() for frase in corpus]
print(f"Corpus      : {corpus}")
```

```
vocabulario = set(token for frase in corpus for token in nltk.word_tokenize(frase))
print(f"vocabulário : {vocabulario}")
```

```
⇒ Corpus      : ['o gato correu rápido', 'o cachorro correu devagar', 'o gato e o cachorro']
vocabulário : {'correu', 'gato', 'o', 'rápido', 'cachorro', 'correram', 'devagar'}
```

```
## Documentos:
# Tipo: dicionário
# {
#     "doc_i": [lista de frases splitadas do tokenizadas doc_i]
# }
# Exemplo:
# documentos['doc_1'] = ['o', 'gato', 'correu', 'rápido']
documentos = {f"doc_{i+1}": nltk.word_tokenize(frase) for i, frase in enumerate(docs)}
print(documentos)
```

```
⇒ {'doc_1': ['o', 'gato', 'correu', 'rápido'], 'doc_2': ['o', 'cachorro', 'correu']}
```

```
## Bow_all_docs
# : contém um dicionario em que chaves sao nomes de documentos e
#     valor é um dicionario de {'token': qtd_ocorrencias}
# Tipo: dicionário onde as chaves são strings e os valores são dicionários,
#     cujas chaves são strings e os valores são inteiros.
# Exemplo:
# documentos['doc_1']: {'gato': 1, 'correu': 1, 'o': 1, 'rápido': 1}
bow_all_docs_dict = {
    f"doc_{i+1}": {token:0 for token in vocabulario} for i, frase in enumerate(docs)
}
print(bow_all_docs_dict)

for doc_name, doc_tokens in documentos.items():
    for token in doc_tokens:
        bow_all_docs_dict[doc_name][token] += 1

print(bow_all_docs_dict)
print(f"\nbow_all_docs['doc_1']: {bow_all_docs_dict['doc_1']}")
```

```
⇒ {'doc_1': {'correu': 0, 'gato': 0, 'o': 0, 'rápido': 0, 'cachorro': 0, 'correu': 0},
    'doc_2': {'correu': 1, 'gato': 1, 'o': 1, 'rápido': 1, 'cachorro': 0, 'correu': 0}}

bow_all_docs['doc_1']: {'correu': 1, 'gato': 1, 'o': 1, 'rápido': 1, 'cachorro': 0}
```

## Alternativa

```
# Alternativa
from collections import Counter

bow_all_docs_dict = {
    f"doc_{i+1}": Counter(documentos[f"doc_{i+1}"])
    for i in range(len(documentos))
}
print(bow_all_docs_dict)
```

## ▼ visualizando

```
# quick check
for doc in bow_all_docs_dict:
    print(f"{doc}: {bow_all_docs_dict[doc]}")
```

⇒ doc\_1: {'correu': 1, 'gato': 1, 'o': 1, 'rápido': 1, 'cachorro': 0, 'correram  
doc\_2: {'correu': 1, 'gato': 0, 'o': 1, 'rápido': 0, 'cachorro': 1, 'correram  
doc\_3: {'correu': 0, 'gato': 1, 'o': 2, 'rápido': 0, 'cachorro': 1, 'correram

## ▼ Validacao

```

## Referencia
ref_Vocabulario = ["0", "gato", "correu", "rápido", "cachorro", "devagar", "e", "c
ref_Vocabulario = list(map(lambda x: x.lower(), ref_Vocabulario))
ref_d1 = [1, 1, 1, 1, 0, 0, 0, 0]
ref_d2 = [1, 0, 1, 0, 1, 1, 0, 0]
ref_d3 = [2, 1, 0, 0, 1, 0, 1, 1]

gabarito = {
    "doc_1": dict(zip(ref_Vocabulario, ref_d1)),
    "doc_2": dict(zip(ref_Vocabulario, ref_d2)),
    "doc_3": dict(zip(ref_Vocabulario, ref_d3))
}

# quick check
for doc_name, doc_bow in gabarito.items():
    print(f"{doc_name}: {doc_bow}")

⇒ doc_1: {'o': 1, 'gato': 1, 'correu': 1, 'rápido': 1, 'cachorro': 0, 'devagar':
doc_2: {'o': 1, 'gato': 0, 'correu': 1, 'rápido': 0, 'cachorro': 1, 'devagar':
doc_3: {'o': 2, 'gato': 1, 'correu': 0, 'rápido': 0, 'cachorro': 1, 'devagar':

print("Validacao")
valido = True
for doc_name, doc_bow in gabarito.items():
    # print(f"{doc_name}: {doc_name["doc_bow"] == bow_all_docs[doc_name]}")
    for token in doc_bow:
        try:
            if doc_bow[token] != bow_all_docs_dict[doc_name][token]:
                valido = False
                print(f"[False]: {token}: {doc_bow[token] == bow_all_docs_dict[doc_name]
        except:
            valido = False
            print(f"[Falso] Palavra nao encontrada: {token} ")
if valido:
    print("[True]Todos os tokens estão corretos")

⇒ Validacao
[True]Todos os tokens estão corretos


```



## ▼ Dataframe

```
# Dataframe
columns = list(vocabulario)
print(f"columns: {columns}")

df = pd.DataFrame(columns=columns)
print(df)

for doc in bow_all_docs_dict:
    for token in bow_all_docs_dict[doc]:
        df.at[doc, token] = bow_all_docs_dict[doc][token]
df.head()
```

 columns: ['correu', 'gato', 'o', 'rápido', 'cachorro', 'correram', 'devagar',  
 Empty DataFrame  
 Columns: [correu, gato, o, rápido, cachorro, correram, devagar, e]  
 Index: []

	correu	gato	o	rápido	cachorro	correram	devagar	e	
doc_1	1	1	1	1	0	0	0	0	
doc_2	1	0	1	0	1	0	1	0	
doc_3	0	1	2	0	1	1	0	1	

Next steps:

[Generate code with df](#)[View recommended plots](#)[New interactive sheet](#)

## ✓ Texto da tarefa

1. Definir texto
2. Get Corpus
3. Def Documentos
4. Bag of Words dictionary



```

texto = """
And as I sat there brooding on the old, unknown world, I thought of Gatsby's wonder
Gatsby believed in the green light, the orgastic future that year by year recedes
So we beat on, boats against the current, borne back ceaselessly into the past.
"""
print(texto)

```

↗

```

And as I sat there brooding on the old, unknown world, I thought of Gatsby's wonder
Gatsby believed in the green light, the orgastic future that year by year recedes
So we beat on, boats against the current, borne back ceaselessly into the past

```

```

corpus = [frase for frase in texto.split('.')]
print(corpus)

```

↗ ['\nAnd as I sat there brooding on the old, unknown world, I thought of Gatsby's wonder\nGatsby believed in the green light, the orgastic future that year by year recedes\nSo we beat on, boats against the current, borne back ceaselessly into the past\n']

```

vocabulario = set()
for frase in corpus:
    frase = frase.lower()
    frase = nltk.word_tokenize(frase)
    frase = [token for token in frase if token.isalpha()]
    frase = [token for token in frase if token != ""]
    # frase = [token for token in frase if token not in nltk.corpus.stopwords.words('english')]
    vocabulario.update(frase)
print(vocabulario)

```

↗ {'world', 'wonder', 'then', 'recedes', 'before', 'into', 'arms', 'first', 'have', 'beats', 'against', 'current', 'borne', 'back', 'ceaselessly', 'past', 'thought', 'Gatsby's', 'wonder', 'Gatsby', 'believed', 'in', 'the', 'green', 'light', 'the', 'orgastic', 'future', 'that', 'year', 'by', 'year', 'recedes', 'So', 'we', 'beat', 'on', 'boats', 'against', 'the', 'current', 'borne', 'back', 'ceaselessly', 'into', 'the', 'past'}

```

for word in vocabulario:
    if word == '' or word == "''" or word == ",":
        print(word)
        print("dimp")

```

```
## Regex:
## remover tudo que nao for alfanumerico => \W , com excecao dos '.' (pra poder
## (estava incluindo "\n" e apostrofe "'")
texto = texto.replace('\W[^\.]|(\n)', '')
# print(texto.split()[0])
print(texto)
```

```
## check
for palavra in texto.split():
    if palavra == '\nand':
        print(palavra)
```

⇒ And as I sat there brooding on the old, unknown world, I thought of Gatsby's v  
Gatsby believed in the green light, the orgastic future that year by year reced  
So we beat on, boats against the current, borne back ceaselessly into the past

```
print(vocabulario)
custom_stopwords= ["'",",",""]
```

⇒ {'world', 'wonder', 'then', 'recedes', 'before', 'into', 'arms', 'first', 'hav

```
# Corpus
corpus = [frase for frase in texto.split('.') if frase != '']
corpus = [frase.lower() for frase in corpus]
print(f"Corpus      : {corpus}")
```

⇒ Corpus : ['\nand as i sat there brooding on the old, unknown world, i th

```
## Documentos:
# Tipo: dicionário
# {
#     "doc_i": [lista de frases splitadas do tokenizadas doc_i]
# }
# Exemplo:
# documentos['doc_1'] = ['o', 'gato', 'correu', 'rápido']
documentos = {f"doc_{i+1}": nltk.word_tokenize(frase) for i, frase in enumerate(c
print(documentos)
```

```
⇒ {'doc_1': ['and', 'as', 'i', 'sat', 'there', 'brooding', 'on', 'the', 'old',
```

```
## Bow_all_docs
# : contém um dicionario em que chaves sao nomes de documentos e
# valor é um dicionario de {'token': qtd_ocorrencias}
# Tipo: dicionário onde as chaves são strings e os valores são dicionários,
# cujas chaves são strings e os valores são inteiros.
# Exemplo:
# documentos['doc_1']: {'gato': 1, 'correu': 1, 'o': 1, 'rápido': 1}
bow_all_docs_dict = {
    f"doc_{i+1}": {token:0 for token in vocabulario} for i, frase in enumerate(co
}
print(bow_all_docs_dict)

for doc_name, doc_tokens in documentos.items():
    for token in doc_tokens:
        if token in vocabulario:
            bow_all_docs_dict[doc_name][token] += 1

print(bow_all_docs_dict)
print(f"\nbow_all_docs['doc_1']: {bow_all_docs_dict['doc_1']}")
```

```
⇒ {'doc_1': {'world': 0, 'wonder': 0, 'then': 0, 'recedes': 0, 'before': 0, 'int
{'doc_1': {'world': 1, 'wonder': 1, 'then': 0, 'recedes': 0, 'before': 0, 'int

bow_all_docs['doc_1']: {'world': 1, 'wonder': 1, 'then': 0, 'recedes': 0, 'be
```

```
# Visualizar
for doc in bow_all_docs_dict:
    print(f"{doc}: {bow_all_docs_dict[doc]}")
```

```
⇒ doc_1: {'world': 1, 'wonder': 1, 'then': 0, 'recedes': 0, 'before': 0, 'into': 0, 'arms': 0, 'first': 0, 'have': 0, 'eluded': 0}
doc_2: {'world': 0, 'wonder': 0, 'then': 0, 'recedes': 0, 'before': 0, 'into': 0, 'arms': 0, 'first': 0, 'have': 1, 'eluded': 0}
doc_3: {'world': 0, 'wonder': 0, 'then': 0, 'recedes': 0, 'before': 0, 'into': 0, 'arms': 0, 'first': 0, 'have': 0, 'eluded': 0}
doc_4: {'world': 0, 'wonder': 0, 'then': 0, 'recedes': 1, 'before': 1, 'into': 0, 'arms': 0, 'first': 0, 'have': 0, 'eluded': 0}
doc_5: {'world': 0, 'wonder': 0, 'then': 1, 'recedes': 0, 'before': 0, 'into': 1, 'arms': 1, 'first': 0, 'have': 0, 'eluded': 1}
doc_6: {'world': 0, 'wonder': 0, 'then': 0, 'recedes': 0, 'before': 0, 'into': 0, 'arms': 0, 'first': 0, 'have': 0, 'eluded': 0}
```

```
columns = list(vocabulario)
print(columns)
```

```
df = pd.DataFrame(columns=columns)
print(df)
```

```
for doc in bow_all_docs_dict:
    for token in bow_all_docs_dict[doc]:
        df.at[doc, token] = bow_all_docs_dict[doc][token]
df.head()
```

```
⇒ ['world', 'wonder', 'then', 'recedes', 'before', 'into', 'arms', 'first', 'have', 'eluded']
Empty DataFrame
Columns: [world, wonder, then, recedes, before, into, arms, first, have, eluded]
Index: []
```

```
[0 rows x 100 columns]
```

	world	wonder	then	recedes	before	into	arms	first	have	eluded	...
<b>doc_1</b>	1	1	0	0	0	0	0	1	0	0	..
<b>doc_2</b>	0	0	0	0	0	0	0	0	1	0	..
<b>doc_3</b>	0	0	0	0	0	0	0	0	0	0	..
<b>doc_4</b>	0	0	0	1	1	0	0	0	0	0	..
<b>doc_5</b>	0	0	1	0	0	1	1	0	0	1	..

```
5 rows x 100 columns
```

## > Validation

Checar se não há tokens inválidos. Ex: ' , , Ainda há o s que provavelmente da quebra de ` 's porém não há remoção de stopwords neste momento.

[ ] ↪ 1 cell hidden

## Term Frequency - Inverse Document Frequency

[Voltar ao topo](#)

### ✓ TAREFA 2:

#### TF-IDF

1. Preencha as funções do TF-IDF abaixo.
2. Utilize-as para gerar vetores dos documentos abaixo.
3. Descomente os trechos de chamada da função `checaTermo()` para teste.
4. Se você remover as stopwords, haverá impacto na matriz TF-IDF? Qual? Explique de forma breve e compare a matriz com e sem as stop-words.

Obs.: Na prática você pode utilizar a função `TfidfVectorizer` do `scikit-learn` para calcular vetores TF-IDF: [https://scikit-learn.org/1.5/modules/generated/sklearn.feature\\_extraction.text.TfidfVectorizer.html](https://scikit-learn.org/1.5/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html)

- Term Frequency - Inverse Document Frequency (TF-IDF)

The diagram illustrates the components of the TF-IDF formula. On the left, the Term Frequency formula is shown:  $TF(t, d) = \frac{f_{t,d}}{N_d}$ . Annotations point to  $t$  (termo),  $d$  (documento),  $f_{t,d}$  (número de vezes que a palavra  $t$  aparece no documento  $d$ ), and  $N_d$  (número total de palavras no documento  $d$ ). On the right, the Inverse Document Frequency formula is shown:  $IDF(t) = \log \left( \frac{N}{1 + df(t)} \right)$ . Annotations point to the base 10 logarithm,  $N$  (número total de documentos do corpus), and  $df(t)$  (número de documentos que contêm o termo  $t$ ).

$$TF-IDF(t, d) = TF(t, d) \times IDF(t)$$

> def memorias\_postumas\_bras\_cubas\_tokens

[ ] ↳ 1 cell hidden

▼ continuação

foi só pra fechar a declaração do corpus


Start coding or [generate](#) with AI.

```
# [ COMPLETAR FUNÇÃO ABAIXO ]
# Função para calcular a frequência de um termo no documento
"""
    term: o termo para calcular a frequência
    document: lista de palavras no documento
"""
def tf(term, document):

    # [ Código aqui ]
    freq_term = document.count(term) / len(document)

    # Frequência do termo no documento dividido pelo número total de palavras no documento
    return freq_term


## Teste para tf
teste_tf = ["foo", "bar", "foo", "bar"]
print("Passou no teste") if tf("foo", teste_tf) == 0.5 else print("Erro")
```

 Passou no teste

```
# [ COMPLETAR FUNÇÃO ABAIXO ]
# Função para calcular a frequência inversa de documentos que contêm o termo
"""
    term: o termo para calcular a frequência inversa
    corpus: lista de documentos, onde cada documento é uma lista de palavras
"""
def idf(term, corpus):

    # [ Código aqui ]
    num_docs_containing_term = sum(1 for doc in corpus if term in doc)
    return np.log(len(corpus) / (1 + num_docs_containing_term))

## Teste para idf
## teste_idf("foo", teste_idf) == log( 2 / 1 + (2))
teste_idf = [["foo", "bar", "foo", "bar"], ["foo", "bar", "foo", "foo"]]
print("Passou no teste") if idf("foo", teste_idf) == np.log(2 / (1 + 2 )) else print("Erro")
```

 Passou no teste

```
# Função que combina tf e idf para gerar a matriz de pesos
def tf_idf(corpus):
    """
    corpus: lista de documentos, onde cada documento é uma lista de palavras

    Retorna uma matriz de TF-IDF
    """
    # Extrair todos os termos únicos no corpus
    unique_terms = set(term for document in corpus for term in document)

    # Inicializar a matriz de tf-idf
    tf_idf_matrix = np.zeros((len(unique_terms), len(corpus)))

    # Mapear termos para índices
    term_index = {term: i for i, term in enumerate(unique_terms)}

    # Calcular tf-idf para cada termo em cada documento
    for j, document in enumerate(corpus):
        for term in document:
            i = term_index[term]
            tf_value = tf(term, document)
            idf_value = idf(term, corpus)
            tf_idf_matrix[i, j] = tf_value * idf_value

    return tf_idf_matrix, list(unique_terms)

# Gerar a matriz de TF-IDF
matrix, terms = tf_idf(corpus)

# Checar termos interessantes
def checaTermo(string):
    print(f"{string}:\n")
    for i in range(len(terms)):
        if terms[i] == string:
            print(f"Cap1: \t{matrix[i][0]}")
            print(f"Cap2: \t{matrix[i][1]}")
            print(f"Cap3: \t{matrix[i][2]}")
```



```
def calcula_tf_idf(corpus, palavra):  
    corpus_str = [' '.join(doc) for doc in corpus]  
  
    tfidf = TfidfVectorizer()  
    tfidf_matrix = tfidf.fit_transform(corpus_str)  
  
    palavra_idx = tfidf.get_feature_names_out().tolist().index(palavra)  
  
    tfidf_values = tfidf_matrix[:, palavra_idx].toarray()  
  
    return tfidf_values
```

```
# checaTermo("afrodite")  
checaTermo("afrodite")  
  
# Check  
calcula_tf_idf(corpus, "afrodite")
```

⇒ afrodite:

```
Cap1: 0.0  
Cap2: 0.0007189097661492276  
Cap3: 0.0  
array([[0.          ],  
       [0.0360397],  
       [0.          ]])
```

```
# checaTermo("senhora")
checaTermo("senhora")
```

```
# Check
calcula_tf_idf(corpus, "senhora")
```

```
⇒ senhora:
```

```
Cap1:  -0.00022086915351384332
Cap2:  -0.0005100745965457108
Cap3:  -0.0007719554716952976
array([[0.01093923],
       [0.02128563],
       [0.03615315]])
```

```
# checaTermo("vaidade")
checaTermo("vaidade")
```

```
# Check
calcula_tf_idf(corpus, "vaidade")
```

```
⇒ vaidade:
```

```
Cap1:  0.0004669463816984619
Cap2:  0.0
Cap3:  0.0
array([[0.02778258],
       [0.         ],
       [0.         ]])
```

```
# checaTermo("verme")
checaTermo("verme")
```

```
# Check
calcula_tf_idf(corpus, "verme")
```

```
⇒ verme:
```

```
array([[0.         ],
       [0.         ],
       [0.0204042]])
```

[ Respostas aqui ]

## Similaridade do cosseno

Como estamos representando termos como vetores, uma das formas de calcular a similaridade entre eles é calcular o cosseno do ângulo entre os dois, que resultará em um valor entre [0,1], sendo 0 quando os termos são ortogonais (completamente diferentes), e 1 quando são paralelos (completamente iguais).

[Voltar ao topo](#)

### ✓ TAREFA 3:

#### Similaridade do cosseno

1. Crie uma função para calcular a similaridade entre vetores, utilizando o cosseno entre eles.
2. Teste sua função com os vetores de TF-IDF gerados anteriormente. Não é necessário testar as variações sem stop-words.
3. Compare a similaridade total entre eles. Quem é mais similar a quem?

```
from sklearn.metrics.pairwise import cosine_similarity

def calcula_cosseno_check(v1, v2):
    # Calcula a similaridade do cosseno entre os dois vetores
    return cosine_similarity([v1], [v2])[0][0]

def similaridade_cosseno(v1,v2):
    produto_escalar = np.dot(v1, v2)
    norma_v1 = np.linalg.norm(v1)
    norma_v2 = np.linalg.norm(v2)
    return produto_escalar / (norma_v1 * norma_v2)

def dist_cosseno(v1,v2):
    return 1 - similaridade_cosseno(v1,v2)
```

```

transposta = np.transpose(matrix)

d1 = transposta[0]
d2 = transposta[1]
d3 = transposta[2]

# Comparacao
print(f"Similaridade entre Cap1 e Cap2: {similaridade_cosseno(d1, d2)}")
print(f"Similaridade entre Cap1 e Cap3: {similaridade_cosseno(d1, d3)}")
print(f"Similaridade entre Cap2 e Cap3: {similaridade_cosseno(d2, d3)}")

print("=====")
# Comparacao
print(f"Similaridade entre Cap1 e Cap2: {dist_cosseno(d1, d2)}")
print(f"Similaridade entre Cap1 e Cap3: {dist_cosseno(d1, d3)}")
print(f"Similaridade entre Cap2 e Cap3: {dist_cosseno(d2, d3)}")

⇒ Similaridade entre Cap1 e Cap2: 0.8412109535303997
   Similaridade entre Cap1 e Cap3: 0.9204575255543084
   Similaridade entre Cap2 e Cap3: 0.8484731285678712
   =====
   Similaridade entre Cap1 e Cap2: 0.15878904646960035
   Similaridade entre Cap1 e Cap3: 0.07954247444569162
   Similaridade entre Cap2 e Cap3: 0.15152687143212884

```

[ Respostas aqui ]

[Voltar ao topo](#)

