Uma Introdução para NLP:

O Processamento de Linguagem Natural (PLN - ou NLP, em inglês) é uma área da computação que tem como objetivo extrair representações e significados mais completos de textos livres escritos em linguagem natural.

http://www.facom.ufu.br/~wendelmelo/terceiros/tutorial_nltk.pdf

In [1]:

```
import numpy as np
import pandas as pd
import nltk
nltk.download('punkt')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Rodolfo\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Rodolfo\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

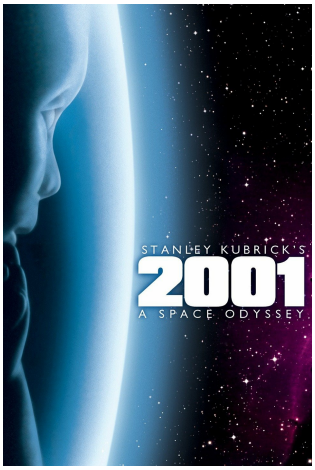
Out[1]:

```
True
```

1. Importar e observar o dataset

Todos nós gostamos de assistir filmes! Gostamos de alguns filmes, de outros não. A maioria das pessoas prefere filmes de gênero semelhante. Alguns de nós adoram assistir filmes de ação, enquanto outros gostam de assistir filmes de terror. Alguns de nós gostam de assistir a filmes com ninjas, enquanto outros gostam de assistir a super-heróis..

Filmes dentro de um gênero geralmente compartilham parâmetros básicos comuns. Considere os dois filmes a seguir:



Ambos os filmes, *2001: Uma Odisséia no Espaço* e *Contatos Imediatos do Terceiro Grau* , são filmes baseados em alienígenas vindo para a Terra. Eu vi os dois e eles realmente compartilham muitas semelhanças. Podemos concluir que ambos se enquadram no mesmo gênero de filmes com base na intuição, mas isso não é divertido em um contexto de ciência de dados.

Neste notebook, quantificaremos a similaridade dos filmes com base em seus resumos de enredo disponíveis na IMDb e Wikipedia, em seguida, separá-los em grupos, também conhecidos como clusters. Criaremos um dendrograma para representar a proximidade dos filmes entre si.

Vamos começar importando o conjunto de dados e observando os dados fornecidos.

In [2]:

```
# Set seed for reproducibility
np.random.seed(5)

# Read in IMDb and Wikipedia movie data (both in same file)
movies_df = pd.read_csv("dataset/movies.csv")

print("Number of movies loaded: %s " % (len(movies_df)))

# Display the data
movies_df.head()
```

```
Number of movies loaded: 100
```

Out[2]:

	rank	title	genre	wiki_plot	imdb_plot
0	0	The Godfather	[u' Crime', u' Drama']	On the day of his only daughter's wedding, Vit...	In late summer 1945, guests are gathered for t...
1	1	The Shawshank Redemption	[u' Crime', u' Drama']	In 1947, banker Andy Dufresne is convicted of ...	In 1947, Andy Dufresne (Tim Robbins), a banker...
2	2	Schindler's List	[u' Biography', u' Drama', u' History']	In 1939, the Germans move Polish Jews into the...	The relocation of Polish Jews from surrounding...
3	3	Raging Bull	[u' Biography', u' Drama', u' Sport']	In a brief scene in 1964, an aging, overweight...	The film opens in 1964, where an older and fat...
4	4	Casablanca	[u' Drama', u' Romance', u' War']	It is early December 1941. American expatriate...	In the early years of World War II, December 1...

2. Combinar resumo do filmes da Wikipedia e IMDb

O conjunto de dados que importamos atualmente contém duas colunas intituladas `wiki_plot` e `imdb_plot`. Eles são o enredo encontrado para os filmes na Wikipedia e IMDb, respectivamente. O texto nas duas colunas é semelhante, no entanto, muitas vezes são escritos em tons diferentes e, portanto, fornecem contexto em um filme de uma maneira diferente de expressão linguística. Além disso, às vezes o texto em uma coluna pode mencionar uma característica que não está presente na outra coluna. Por exemplo, considere os seguintes trechos de trama de *O Poderoso Chefão*:

- **Wikipedia: "On the day of his only daughter's wedding, Vito Corleone"**
- **IMDb: "In late summer 1945, guests are gathered for the wedding reception of Don Vito Corleone's daughter Connie"**

Enquanto o enredo da Wikipedia apenas menciona que é o dia do casamento da filha, o enredo do IMDb também menciona o ano da cena e o nome da filha.

Vamos combinar as duas colunas para evitar sobrecargas na computação associadas a colunas extras para processar.

In [3]:

```
# Combine wiki_plot and imdb_plot into a single column
movies_df['plot'] = movies_df['wiki_plot'].astype(str) + "\n" + \
                    movies_df['imdb_plot'].astype(str)

#pd.set_option('display.max_colwidth', -1) #option to show all text on collumns
#pd.set_option('display.max_colwidth', 40)

# Inspect the new DataFrame
movies_df.head(1)
```

Out[3]:

	rank	title	genre	wiki_plot	imdb_plot	plot
0	0	The Godfather	[u' Crime', u' Drama']	On the day of his only daughter's wedding, Vit...	In late summer 1945, guests are gathered for t...	On the day of his only daughter's wedding, Vit...

3. Tokenização

A tokenização, também conhecida como segmentação de palavras, quebra a sequência de caracteres em um texto localizando o limite de cada palavra, ou seja, os pontos onde uma palavra termina e outra começa. Para fins de linguística computacional, as palavras assim identificadas são frequentemente chamadas de tokens.

Além do método de tokenização fornecido pelo NLTK, pode ser necessário realizar uma filtragem adicional para remover os tokens que são valores inteiramente numéricos ou pontuação.

Embora um programa possa falhar ao construir o contexto de "While waiting at a bus stop in 1981" (Forrest Gump), devido a essa string não corresponder a nenhuma palavra no dicionário, é possível construir o contexto a partir das palavras "while", "waiting" ou "bus" porque estão presentes no dicionário de inglês.

Vamos realizar a tokenização em um pequeno extrato de "O Poderoso Chefão":

In [4]:

```
# Tokeniza um paragrafo em sentenças / frases
sent_tokenized = [sent for sent in nltk.sent_tokenize("""
                        Today (May 19, 2016) is his only daughter's wedding.
                        Vito Corleone is the Godfather.
                        """)]

# Tokeniza a primeira sentença da variavel acima em palavras:
words_tokenized  = [word for word in nltk.word_tokenize(sent_tokenized[0])]

#Enquanto o "word_tokenize" vê tudo como um texto só e separa eles em tokens (palavras),
o sent-tokenize separa as frases...

# Remove tokens que não contém letras
import re
filtered = [word for word in words_tokenized if re.search('[a-zA-Z]', word)]

# Palavras filtradas após tokenização
filtered
```

Out[4]:

```
['Today', 'May', 'is', 'his', 'only', 'daughter', "'s", 'wedding']
```

4. Stemming (Lematização)

A subseção anterior tratava do problema da tokenização, ou seja, quebrar uma entrada de texto em palavras e sentenças que serão tópicos para um processamento subsequente. O processo subsequente em questão é a análise a nível de palavras, a análise léxica.

Uma palavra pode ser pensada de duas maneiras: como uma sequência de caracteres no texto em execução, como por exemplo, o verbo ENTREGAR, ou como um objeto mais abstrato que é o termo principal para um conjunto de sequência de caracteres (palavras), sendo o verbo ENTREGAR o objeto abstrato, ou lemma, do conjunto *{entrega, entregador, entregando, entregue}*.

A tarefa básica da análise léxica é relacionar variantes morfológicas aos seus lemmas, que nada mais são do que as formas canônicas das palavras, ou a forma em que as palavras se encontram no dicionário.

Este propósito é alcançado na prática durante o *stemming*, uma operação de pré processamento de textos onde palavras mais complexas morfologicamente são identificadas, decompondo em seu *stem* invariante, ou melhor, na forma canônica do *lemma* e seus afixos, e no final os afixos são deletados. O *stem*, portanto, é o chamado radical da palavra. Por exemplo, ainda sobre o verbo ENTREGAR, o *lemma* é ENTREGAR e o *stem* é ENTREG, pois a partir do *stem* podem ser criadas outras palavras.

Existem diferentes algoritmos disponíveis para o stemming, como Porter Stemmer, Snowball Stemmer, etc. Para o exemplo, iremos usar o Snowball Stemmer

In [5]:

```
# Importe o SnowballStemmer para realizar o stemming
from nltk.stem.snowball import SnowballStemmer

# Crie um objeto SnowballStemmer em inglês
stemmer = SnowballStemmer("english")

# Para observar palavras sem lematização:
print("Without stemming: ", filtered)

# Stem the words from filtered
stemmed_words = [stemmer.stem(word) for word in filtered]

# Imprima as stemmed_words para observar as palavras após a lematização
print("After stemming:   ", stemmed_words)
```

```
Without stemming:  ['Today', 'May', 'is', 'his', 'only', 'daughter', "'s", 'wedding']
After stemming:    ['today', 'may', 'is', 'his', 'onli', 'daughter', "'s", 'wed']
```

5. Tokenização e Lematização juntas

Agora podemos tokenizar e eliminar frases. Mas podemos ter que usar as duas funções repetidamente, uma após a outra, para lidar com uma grande quantidade de dados, portanto, podemos pensar em envolvê-las em uma função e passar o texto a ser tokenizado e lematizado como argumento da função.

In [6]:

```
# Define a function to perform both stemming (lematização)  and tokenization
def tokenize_and_stem(text):

    # Faça a tokenização de todas as frases e coloque tudo em um vetor só:
    tokens = [word for sent in nltk.sent_tokenize(text) for word in nltk.word_tokenize(s
ent)]

    # Tira numeros e pontuação do vetor acima
    filtered_tokens = [token for token in tokens if re.search('[a-zA-Z]', token)]

    #lematiza as sentenças
    stems = [stemmer.stem(t) for t in filtered_tokens]

    return stems

words_stemmed = tokenize_and_stem("Today (May 19, 2016) is his only daughter's wedding.")
print(words_stemmed)
```

```
['today', 'may', 'is', 'his', 'onli', 'daughter', "'s", 'wed']
```

6. Criar TfidfVectorizer

Os computadores não *entendem* texto. Essas são máquinas capazes apenas de compreender números e realizar cálculos numéricos. Portanto, devemos converter nossos resumos de enredo textual em números para que o computador seja capaz de extrair significado deles. Um método simples de fazer isso seria contar todas as ocorrências de cada palavra em todo o vocabulário e retornar as contagens em um vetor. Para isso temos o `CountVectorizer` .

Considere a palavra 'o'. Aparece com bastante frequência em quase todos os enredos de filme e terá uma contagem alta em cada caso. Mas obviamente, não é o tema de todos os filmes! O valor tf–idf (que significa frequência do termo–inverso da frequência nos documentos) é um método que supera as deficiências do `CountVectorizer` . A Frequência do Termo de uma palavra é a medida de quantas vezes ela aparece em um documento, enquanto a Frequência Inversa do Documento é o parâmetro que reduz a importância de uma palavra se ela aparecer com frequência em vários documentos.

Por exemplo, quando aplicamos o TF-IDF nas primeiras 3 frases do enredo de *O Mágico de Oz* , somos informados de que a palavra mais importante que existe é 'Toto', o animal de estimação cão do personagem principal. Isso ocorre porque o filme começa com 'Toto' mordendo alguém devido ao qual a jornada de Oz começa!

Em termos mais simples, o TF-IDF reconhece palavras que são únicas e importantes para qualquer documento. Vamos criar um para nossos propósitos.

In [7]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
# Instantiate TfidfVectorizer object with stopwords and tokenizer
# parameters for efficient processing of text
tfidf_vectorizer = TfidfVectorizer(max_df=0.8, max_features=200000,
                                   min_df=0.2, stop_words='english',
                                   use_idf=True, tokenizer=tokenize_and_stem,
                                   ngram_range=(1,3))
```

7. Treinar TfidfVectorizer

Uma vez que criamos um TF-IDF Vectorizer, devemos ajustar o texto a ele e então transformar o texto para produzir a forma numérica correspondente dos dados que o computador será capaz de entender e derivar significado. Para fazer isso, usamos o método `fit_transform()` do objeto `TfidfVectorizer`.

Se nós observarmos o objeto `TfidfVectorizer` que nós criamos, nos deparamos com um parâmetro chamado stopwords.

Stopwords são palavras que podem ser consideradas irrelevantes para o entedimento do sentido de um texto, ou seja, palavras semanticamente irrelavantes. Exemplos: as, e, os, de, para, com, sem, foi. Essas palavras são geralmente removidas de um texto durante a fase de pré-processamento.

In [8]:

```
stopwords = nltk.corpus.stopwords.words('portuguese')
stopwords[:10]
```

Out[8]:

```
['de', 'a', 'o', 'que', 'e', 'é', 'do', 'da', 'em', 'um']
```

In [9]:

```
# Fit and transform the tfidf_vectorizer with the "plot" of each movie
# to create a vector representation of the plot summaries
tfidf_matrix = tfidf_vectorizer.fit_transform([x for x in movies_df["plot"]])

print(tfidf_matrix.shape)
```

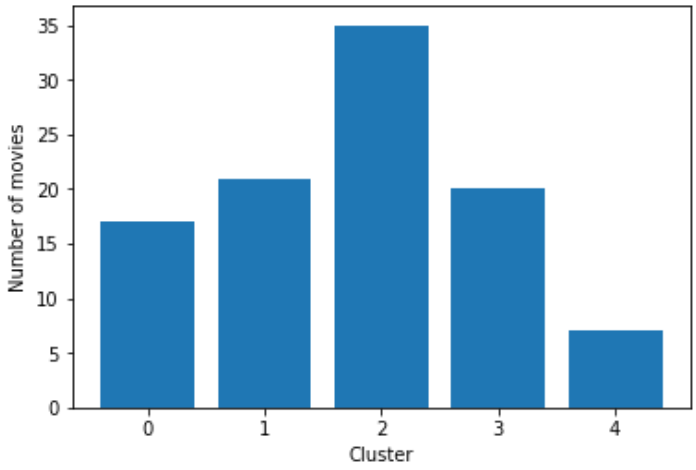
```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\feature_extraction\text.py:300: UserWa
rning: Your stop_words may be inconsistent with your preprocessing. Tokenizing the stop w
ords generated tokens ['abov', 'afterward', 'alon', 'alreadi', 'alway', 'ani', 'anoth', '
anyon', 'anyth', 'anywher', 'becam', 'becaus', 'becom', 'befor', 'besid', 'cri', 'describ
', 'dure', 'els', 'elsewher', 'empti', 'everi', 'everyon', 'everyth', 'everywher', 'fifti
', 'forti', 'henc', 'hereaft', 'herebi', 'howev', 'hundr', 'inde', 'mani', 'meanwhil', 'm
oreov', 'nobodi', 'noon', 'noth', 'nowher', 'onc', 'onli', 'otherwis', 'ourselv', 'perhap
', 'pleas', 'sever', 'sinc', 'sincer', 'sixti', 'someon', 'someth', 'sometim', 'somewher'
, 'themselv', 'thenc', 'thereaft', 'therebi', 'therefor', 'togeth', 'twelv', 'twenti', 'v
eri', 'whatev', 'whenc', 'whenev', 'wherea', 'whereaft', 'wherebi', 'wherev', 'whi', 'you
rselv'] not in stop_words.
  'stop_words.' % sorted(inconsistent))
```

```
(100, 564)
```

8. Importar KMeans e criar os clusters

Para determinar a proximidade de um filme com o outro com a ajuda do aprendizado não supervisionado, podemos usar técnicas de agrupamento. Clustering é o método de agrupar vários itens de forma que exibam propriedades semelhantes. De acordo com a medida de similaridade desejada, uma dada amostra de itens pode ter um ou mais clusters.
K-means é um algoritmo que nos ajuda a implementar clustering em Python. O nome deriva de seu método de implementação: a amostra dada é dividida em K clusters onde cada cluster é denotado pela média de todos os

itens que estão naquele cluster. Obtemos a seguinte distribuição para os clusters:



In [10]:

```
from sklearn.cluster import KMeans

# Create a KMeans object with 5 clusters
km = KMeans(n_clusters=5)

# Fit the k-means object with tfidf_matrix
km.fit(tfidf_matrix)

clusters = km.labels_.tolist()

# Create a column cluster to denote the generated cluster for each movie
movies_df["cluster"] = clusters

# Display number of films per cluster (clusters from 0 to 4)
movies_df['cluster'].value_counts()
```

Out[10]:

```
2    35
1    21
3    20
0    17
4     7
Name: cluster, dtype: int64
```

9. Calculate similarity distance

Consider the following two sentences from the movie *The Wizard of Oz*:

> **"they find in the Emerald City"**
>
> **"they finally reach the Emerald City"**

If we put the above sentences in a `CountVectorizer`, the vocabulary produced would be "they, find, in, the, Emerald, City, finally, reach" and the vectors for each sentence would be as follows:

> **1, 1, 1, 1, 1, 1, 0, 0**
>
> **1, 0, 0, 1, 1, 1, 1, 1**

When we calculate the cosine angle formed between the vectors represented by the above, we get a score of 0.667. This means the above sentences are very closely related. *Similarity distance* is 1 - cosine similarity angle. This follows from that if the vectors are similar, the cosine of their angle would be 1 and hence, the distance between then would be 1 - 1 = 0.

Let's calculate the similarity distance for all of our movies.

In [11]:

```
# Import cosine_similarity to calculate similarity of movie plots
from sklearn.metrics.pairwise import cosine_similarity

# Calculate the similarity distance
similarity_distance = 1 - cosine_similarity(tfidf_matrix)
```

10. Matplotlib, Linkage, e Dendrogramas

Vamos agora criar um diagrama em forma de árvore (chamado dendrograma) dos títulos dos filmes para nos ajudar a entender o nível de semelhança entre eles visualmente. Os dendrogramas ajudam a visualizar os resultados do agrupamento hierárquico, que é uma alternativa ao agrupamento k-means. Espera-se que dois pares de filmes no mesmo nível de agrupamento hierárquico tenham força de similaridade semelhante entre os pares de filmes correspondentes. Por exemplo, o filme *Fargo* seria tão semelhante a *North By Northwest* quanto o filme *Platoon* é para *Saving Private Ryan* , dado que ambos os pares exibem o mesmo nível de hierarquia. *Vamos importar os módulos de que precisaremos para criar nosso dendrograma.*

In [12]:

```
# Import matplotlib.pyplot for plotting graphs
import matplotlib.pyplot as plt
# ... YOUR CODE FOR TASK 10 ...

# Configure matplotlib to display the output inline
%matplotlib inline

# Import modules necessary to plot dendrogram
from scipy.cluster.hierarchy import linkage, dendrogram
```

11. Criar dendrograma de mesclagem e plotagem

Faremos um dendrograma dos filmes cuja medida de similaridade será dada pela distância de similaridade que calculamos anteriormente. Quanto menor for a distância de similaridade entre dois filmes, menor será a interceptação no eixo y. Por exemplo, o vínculo de dendrograma mais baixo que descobriremos será entre os filmes, It's a Wonderful Life e A Place in the Sun. Isso indica que os filmes são muito semelhantes entre si em seus enredos.

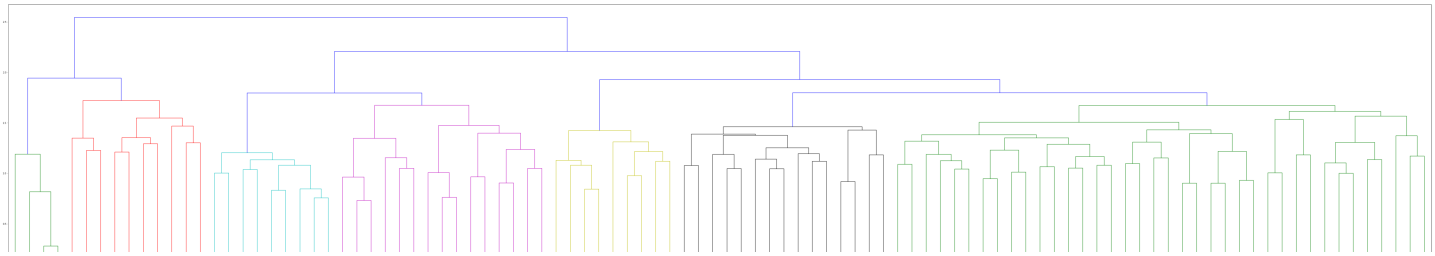
In [13]:

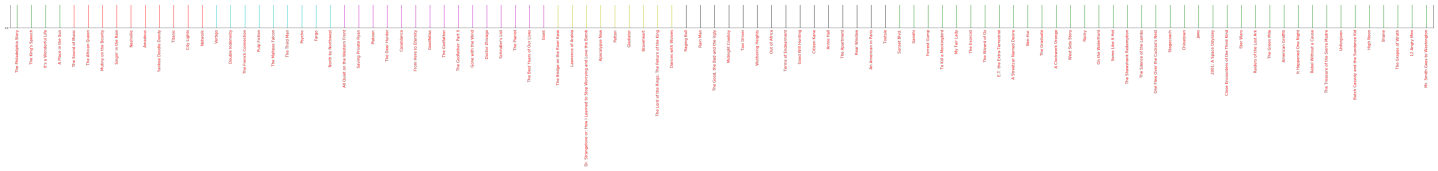
```
# Create mergings matrix
mergings = linkage(similarity_distance, method='complete')

# Plot the dendrogram, using title as label column
dendrogram_ = dendrogram(mergings,
               labels=[x for x in movies_df["title"]],
               leaf_rotation=90,
               leaf_font_size=16,
)

# Adjust the plot
fig = plt.gcf()
_ = [lbl.set_color('r') for lbl in plt.gca().get_xmajorticklabels()]
fig.set_size_inches(108, 21)

# Show the plotted dendrogram
plt.show()
```





In [ ]:

In [ ]: