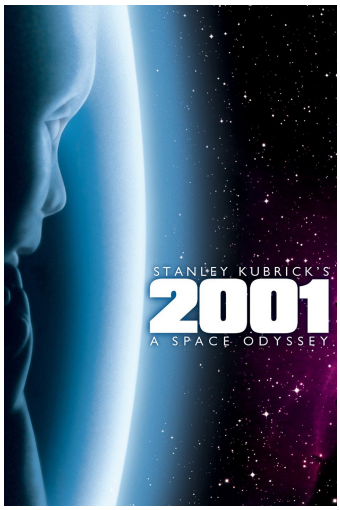
1. Import and observe dataset

We all love watching movies! There are some movies we like, some we don't. Most people have a preference for movies of a similar genre. Some of us love watching action movies, while some of us like watching horror. Some of us like watching movies that have ninjas in them, while some of us like watching superheroes.

Movies within a genre often share common base parameters. Consider the following two movies:



Both movies, *2001: A Space Odyssey* and *Close Encounters of the Third Kind*, are movies based on aliens coming to Earth. I've seen both, and they indeed share many similarities. We could conclude that both of these fall into the same genre of movies based on intuition, but that's no fun in a data science context. In this notebook, we will quantify the similarity of movies based on their plot summaries available on IMDb and Wikipedia, then separate them into groups, also known as clusters. We'll create a dendrogram to represent how closely the movies are related to each other.

Let's start by importing the dataset and observing the data provided.

In [28]:
```
# Import modules
import numpy as np
import pandas as pd
import nltk

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

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

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

# Display the data
movies_df
```

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

Out[28]:

	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...
5	5	One Flew Over the Cuckoo's Nest	[u' Drama']	In 1963 Oregon, Randle Patrick "Mac" McMurphy ...	In 1963 Oregon, Randle Patrick McMurphy (Nicho...
6	6	Gone with the Wind	[u' Drama', u' Romance', u' War']	\nPart 1\n \n Part 1 Part 1 \n \n On the...	The film opens in Tara, a cotton plantation ow...
7	7	Citizen Kane	[u' Drama', u' Mystery']	\n\n\n\nOrson Welles as Charles Foster Kane\n\...	It's 1941, and newspaper tycoon Charles Foster...
8	8	The Wizard of Oz	[u' Adventure', u' Family', u' Fantasy', u' Mu...	The film starts in sepia-tinted Kansas in the ...	Dorothy Gale (Judy Garland) is an orphaned tee...
9	9	Titanic	[u' Drama', u' Romance']	In 1996, treasure hunter Brock Lovett and his ...	In 1996, treasure hunter Brock Lovett and his ...
10	10	Lawrence of Arabia	[u' Adventure', u' Biography', u' Drama', u' H...	] \n The film is presented in two parts, s...	In 1935, T. E. Lawrence (Peter O'Toole) is kil...
11	11	The Godfather: Part II	[u' Crime', u' Drama']	\nIn 1901 Corleone, Sicily, nine-year-old Vito...	The Godfather Part II presents two parallel st...
12	12	Psycho	[u' Horror', u' Mystery', u' Thriller']	Patrick Bateman is a wealthy investment banker...	In a Phoenix hotel room on a Friday afternoon,...
13	13	Sunset Blvd.	[u' Drama', u' Film-Noir']	At a Sunset Boulevard mansion, the body of Joe...	The film opens with the camera tracking down S...
14	14	Vertigo	[u' Mystery', u' Romance', u' Thriller']	ridge, Fort Point\n\n \n \n\n\n"Madeleine" a...	A woman's face gives way to a kaleidoscope of ...
15	15	On the Waterfront	[u' Crime', u' Drama']	Mob-connected union boss Johnny Friendly (Lee ...	Terry Malloy (Marlon Brando) once dreamt of be...
16	16	Forrest Gump	[u' Drama', u' Romance']	While waiting at a bus stop in 1981, Forrest G...	The film begins with a feather falling to the ...

	rank	title	genre	wiki_plot	imdb_plot
17	17	The Sound of Music	[u' Biography', u' Drama', u' Family', u' Musi...	In 1938, while living as a young postulant at ...	The widowed, retired Austrian naval officer, C...
18	18	West Side Story	[u' Crime', u' Drama', u' Musical', u' Romance...	]] \n In the West Side's Lincoln Square ne...	A fight set to music between an American gang,...
19	19	Star Wars	[u' Action', u' Adventure', u' Fantasy', u' Sc...	The galaxy is in a civil war, and spies for th...	\nNote: Italicized paragraphs denote scenes ad...
20	20	E.T. the Extra-Terrestrial	[u' Adventure', u' Family', u' Sci-Fi']	In a California forest, a group of alien botan...	In a forested area overlooking a sprawling sub...
21	21	2001: A Space Odyssey	[u' Mystery', u' Sci-Fi']	The film consists of four major sections, all ...	To Richard Strauss' tone poem "Thus Spake Zara...
22	22	The Silence of the Lambs	[u' Crime', u' Drama', u' Thriller']	is pulled from her training at the FBI Academy...	Promising FBI Academy student Clarice Starling...
23	23	Chinatown	[u' Drama', u' Mystery', u' Thriller']	A woman identifying herself as Evelyn Mulwray ...	Set in 1937 Los Angeles, a private investigato...
24	24	The Bridge on the River Kwai	[u' Adventure', u' Drama', u' War']	In World War II, British prisoners arrive at a...	this synopsis is primarily from the wikipedia ...
25	25	Singin' in the Rain	[u' Comedy', u' Musical', u' Romance']	Don Lockwood is a popular silent film star wit...	Don Lockwood (Gene Kelly) is a popular silent ...
26	26	It's a Wonderful Life	[u' Drama', u' Family', u' Fantasy']	\n\n\n\nDonna Reed (as Mary Bailey) and James ...	This movie is about a divine intervention by a...
27	27	Some Like It Hot	[u' Comedy']	It is February 1929 in the city of Chicago. Jo...	Joe and Jerry, a saxophonist and bassist, resp...
28	28	12 Angry Men	[u' Drama']	The story begins in a New York City courthous...	A teenaged Hispanic boy has just been tried fo...
29	29	Dr. Strangelove or: How I Learned to Stop Worr...	[u' Comedy', u' War']	United States Air Force Brigadier General Jack...	At the Burpelson U.S. Air Force Base somewhere...
...	...	...	...	...	...
70	70	Rain Man	[u' Drama']	Charlie Babbitt is in the middle of importing ...	Charlie Babbitt (Tom Cruise), a Los Angeles ca...
71	71	Annie Hall	[u' Comedy', u' Drama', u' Romance']	The comedian Alvy Singer (Woody Allen) is tryi...	Annie Hall is a film about a comedian, Alvy Si...
72	72	Out of Africa	[u' Biography', u' Drama', u' Romance']	The story begins in 1913 in Denmark, when Kare...	[Out Of Africa]A well-heeled Danish lady goes ...
73	73	Good Will Hunting	[u' Drama']	Twenty-year-old Will Hunting (Damon) of South ...	Though Will Hunting (Matt Damon) has genius-le...

	rank	title	genre	wiki_plot	imdb_plot
74	74	Terms of Endearment	[u' Comedy', u' Drama']	Aurora Greenway (Shirley MacLaine) and her dau...	NaN
75	75	Tootsie	[u' Comedy', u' Drama', u' Romance']	Michael Dorsey (Dustin Hoffman) is a respected...	Michael Dorsey is an actor living and working ...
76	76	Fargo	[u' Crime', u' Drama', u' Thriller']	In the winter of 1987, Minneapolis car salesma...	The movie opens with a car towing a new tan Ol...
77	77	Giant	[u' Drama', u' Romance']	Jordan "Bick" Benedict (Rock Hudson), head of ...	In the early 1920s, Jordan "Bick" Benedict (Ro...
78	78	The Grapes of Wrath	[u' Drama']	The film opens with Tom Joad (Henry Fonda), re...	After serving four years in prison for killing...
79	79	Shane	[u' Drama', u' Romance', u' Western']	\n\n\n\nAlan Ladd and Jean Arthur\n\n \n \n\...	NaN
80	80	The Green Mile	[u' Crime', u' Drama', u' Fantasy', u' Mystery']	In a Louisiana nursing home in 1999, Paul Edge...	The movie begins with an old man named Paul Ed...
81	81	Close Encounters of the Third Kind	[u' Drama', u' Sci-Fi']	In the Sonoran Desert, French scientist Claude...	In what appers to be the Sonoran Desert; in or...
82	82	Network	[u' Drama']	Howard Beale, the longtime anchor of the Union...	NaN
83	83	Nashville	[u' Drama', u' Music']	The overarching plot takes place over five day...	The overarching plot takes place over five day...
84	84	The Graduate	[u' Comedy', u' Drama', u' Romance']	Benjamin Braddock, going on from twenty to twe...	The film explores the life of 21-year-old Ben ...
85	85	American Graffiti	[u' Comedy', u' Drama']	In late August 1962 recent high school graduat...	It's the last night of the summer in 1962, and...
86	86	Pulp Fiction	[u' Crime', u' Drama', u' Thriller']	The Diner" "Prologue—The Diner" \n "Pumpkin...	Late one morning in the Hawthorne Grill, a res...
87	87	The African Queen	[u' Adventure', u' Romance', u' War']	Robert Morley and Katharine Hepburn play Samue...	An English spinster, Rose (Katharine Hepburn),...
88	88	Stagecoach	[u' Adventure', u' Western']	In 1880, a motley group of strangers boards th...	NaN
89	89	Mutiny on the Bounty	[u' Adventure', u' Drama', u' History']	In the year 1787, the Bounty sets sail from En...	NaN
90	90	The Maltese Falcon	[u' Drama', u' Film-Noir', u' Mystery']	\n\n\nIn 1539 the Knight Templars of Malta, pa...	Private eye Sam Spade and his partner Miles Ar...

	rank	title	genre	wiki_plot	imdb_plot
91	91	A Clockwork Orange	[u' Crime', u' Drama', u' Sci-Fi']	In futuristic London, Alex DeLarge is the lead...	A bit of the old ultra-violence".London, Engla...
92	92	Taxi Driver	[u' Crime', u' Drama']	Travis Bickle, an honorably discharged U.S. Ma...	Travis Bickle (Robert De Niro) goes to a New Y...
93	93	Wuthering Heights	[u' Drama', u' Romance']	A traveller named Lockwood (Miles Mander) is c...	NaN
94	94	Double Indemnity	[u' Crime', u' Drama', u' Film-Noir', u' Thril...	\n\n\n\nNeff confesses into a Dictaphone.\n\n ...	Walter Neff (MacMurray) is a successful insura...
95	95	Rebel Without a Cause	[u' Drama']	\n\n\n\nJim Stark is in police custody.\n\n \...	Shortly after moving to Los Angeles with his p...
96	96	Rear Window	[u' Mystery', u' Thriller']	\n\n\n\nJames Stewart as L.B. Jefferies\n\n \...	L.B. "Jeff" Jeffries (James Stewart) recuperat...
97	97	The Third Man	[u' Film-Noir', u' Mystery', u' Thriller']	\n\n\n\nSocial network mapping all major chara...	Sights of Vienna, Austria, flash across the sc...
98	98	North by Northwest	[u' Mystery', u' Thriller']	Advertising executive Roger O. Thornhill is mi...	At the end of an ordinary work day, advertisin...
99	99	Yankee Doodle Dandy	[u' Biography', u' Drama', u' Musical']	\n In the early days of World War II, Cohan ...	NaN

100 rows × 5 columns

In [29]:
```
%%nose

def test_pandas_loaded():
    assert ('pd' in globals()), "Please check your pandas import statement."

def test_numpy_loaded():
    assert ('np' in globals()), "Please check your numpy import statement."

def test_nltk_loaded():
    assert ('nltk' in globals()), "Please check your nltk import statement."

# def test_movies_loaded():
#     assert (movies_df.shape == (100, 6)), "Make sure you have loaded correct
ly the dataset file `datasets/movies.csv`"

def test_movies_correctly_loaded():
    correct_movies_df = pd.read_csv('datasets/movies.csv')
    assert correct_movies_df.equals(movies_df), "The variable movies_df does n
ot contain the data in movies.csv."
```

Out[29]:
```
4/4 tests passed
```

2. Combine Wikipedia and IMDb plot summaries

The dataset we imported currently contains two columns titled `wiki_plot` and `imdb_plot`. They are the plot found for the movies on Wikipedia and IMDb, respectively. The text in the two columns is similar, however, they are often written in different tones and thus provide context on a movie in a different manner of linguistic expression. Further, sometimes the text in one column may mention a feature of the plot that is not present in the other column. For example, consider the following plot extracts from *The Godfather*:

- 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"

While the Wikipedia plot only mentions it is the day of the daughter's wedding, the IMDb plot also mentions the year of the scene and the name of the daughter.

Let's combine both the columns to avoid the overheads in computation associated with extra columns to process.

In [30]:
```
# 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)

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

Out[30]:

	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...
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...	In 1947, banker Andy Dufresne is convicted of ...
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...	In 1939, the Germans move Polish Jews into the...
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...	In a brief scene in 1964, an aging, overweight...
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...	It is early December 1941. American expatriate...

In [31]:
```
%%nose

def test_plot_correctly_added():
    correct_movies_df = pd.read_csv('datasets/movies.csv')
    correct_movies_df["plot"] = correct_movies_df["wiki_plot"].astype(str) +
 "\n" + correct_movies_df["imdb_plot"].astype(str)
    assert correct_movies_df.equals(movies_df), "The variable movies_df does n
ot contain the data in movies.csv."
```

Out[31]:
```
1/1 tests passed
```

3. Tokenization

Tokenization is the process by which we break down articles into individual sentences or words, as needed. Besides the tokenization method provided by NLTK, we might have to perform additional filtration to remove tokens which are entirely numeric values or punctuation.

While a program may fail to build context from "While waiting at a bus stop in 1981" (*Forrest Gump*), because this string would not match in any dictionary, it is possible to build context from the words "while", "waiting" or "bus" because they are present in the English dictionary.

Let us perform tokenization on a small extract from *The Godfather*.

In [32]:
```
# Tokenize a paragraph into sentences and store in sent_tokenized
sent_tokenized = [sent for sent in nltk.sent_tokenize("""
                        Today (May 19, 2016) is his only daughter's wedding.
                        Vito Corleone is the Godfather.
                        """)]

# Word Tokenize first sentence from sent_tokenized, save as words_tokenized
words_tokenized = [word for word in nltk.word_tokenize(sent_tokenized[0])]

# Remove tokens that do not contain any letters from words_tokenized
import re

filtered = [word for word in words_tokenized if re.search('[a-zA-Z]', word)]

# Display filtered words to observe words after tokenization
filtered
```

Out[32]:
```
['Today', 'May', 'is', 'his', 'only', 'daughter', "'s", 'wedding']
```

In [33]:
```
%%nose

correct_sent_tokenized = [sent for sent in nltk.sent_tokenize("""
                        Today (May 19, 2016) is his only daughter's wedding.
                        Vito Corleone is the Godfather.
                        """)]

correct_words_tokenized = [word for word in nltk.word_tokenize(correct_sent_to
kenized[0])]

correct_filtered = [word for word in correct_words_tokenized if re.search('[a-
zA-Z]', word)]

def test_nltk_sent_tokenize_present():
    assert nltk.sent_tokenize("abc def"), "Did you import the nltk module corr
ectly? nltk.sent_tokenize() isn't working as expected."

def test_sent_tokenized():
    assert sent_tokenized == correct_sent_tokenized, \
    "Please check you have tokenized the sentences correctly and stored them i
n sent_tokenized."

def test_words_tokenized():
    assert words_tokenized == correct_words_tokenized, \
    "Please check you have tokenized the words correctly and stored them in wo
rds_tokenized."

def test_re_import():
    assert "re" in globals(), "Please check that you have correctly imported t
he re module."

def test_filtered():
    assert filtered == correct_filtered, \
    "Please check you have filtered the tokens correctly and stored them in fi
ltered."
```

Out[33]:
```
5/5 tests passed
```

4. Stemming

Stemming is the process by which we bring down a word from its different forms to the root word. This helps us establish meaning to different forms of the same words without having to deal with each form separately. For example, the words 'fishing', 'fished', and 'fisher' all get stemmed to the word 'fish'.

Consider the following sentences:

- "Young William Wallace witnesses the treachery of Longshanks" ~ *Gladiator*
- "escapes to the city walls only to witness Cicero's death" ~ *Braveheart*

Instead of building separate dictionary entries for both witnesses and witness, which mean the same thing outside of quantity, stemming them reduces them to 'wit'.

There are different algorithms available for stemming such as the Porter Stemmer, Snowball Stemmer, etc. We shall use the Snowball Stemmer.

In [34]:
```
# Import the SnowballStemmer to perform stemming
from nltk.stem.snowball import SnowballStemmer

# Create an English language SnowballStemmer object
stemmer = SnowballStemmer("english")

# Print filtered to observe words without stemming
print("Without stemming: ", filtered)

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

# Print the stemmed_words to observe words after stemming
print("After stemming:   ", stemmed_words)
```

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

In [35]:
```
%%nose

def test_nltk_sent_tokenize_present():
    assert "SnowballStemmer" in globals(), "Did you import the SnowballStemmer
correctly?"

def test_english_stemmer():
    assert str(stemmer.__dict__["stemmer"]) == "<EnglishStemmer>", \
    "Please check you have initialized the stemmer with 'english' language."

def test_stemmed_words():
    assert stemmed_words == [stemmer.stem(t) for t in filtered] , \
    "Please make sure you have correctly stemmed the words and stored them in
 stemmed_words."
```

Out[35]:
```
3/3 tests passed
```

5. Club together Tokenize & Stem

We are now able to tokenize and stem sentences. But we may have to use the two functions repeatedly one after the other to handle a large amount of data, hence we can think of wrapping them in a function and passing the text to be tokenized and stemmed as the function argument. Then we can pass the new wrapping function, which shall perform both tokenizing and stemming instead of just tokenizing, as the tokenizer argument while creating the TF-IDF vector of the text.

What difference does it make though? Consider the sentence from the plot of *The Godfather*: "Today (May 19, 2016) is his only daughter's wedding." If we do a 'tokenize-only' for this sentence, we have the following result:

> 'today', 'may', 'is', 'his', 'only', 'daughter', "'s", 'wedding'

But when we do a 'tokenize-and-stem' operation we get:

> 'today', 'may', 'is', 'his', 'onli', 'daughter', "'s", 'wed'

All the words are in their root form, which will lead to a better establishment of meaning as some of the non-root forms may not be present in the NLTK training corpus.

In [36]:
```
# Define a function to perform both stemming and tokenization
def tokenize_and_stem(text):

    # Tokenize by sentence, then by word
    tokens = [word for sent in nltk.sent_tokenize(text) for word in nltk.word_
tokenize(sent)]

    # Filter out raw tokens to remove noise
    filtered_tokens = [token for token in tokens if re.search('[a-zA-Z]', toke
n)]

    # Stem the filtered_tokens
    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']
```

In [37]:
```
%%nose

def test_tokenize_and_stem():
    assert "tokenize_and_stem" in globals(), "Make sure you have declared the
 tokenize_and_stem() function correctly."

def test_correct_stems():

    def c_tokenize_and_stem(text):
        tokens = [word for sent in nltk.sent_tokenize(text) for word in nltk.w
ord_tokenize(sent)]
        filtered_tokens = [token for token in tokens if re.search('[a-zA-Z]',
 token)]
        stems = [stemmer.stem(t) for t in filtered_tokens]
        return stems

    stemsList = [0 for x in movies_df["plot"] if c_tokenize_and_stem(x) == tok
enize_and_stem(x)]

    assert sum(stemsList) == 0, \
    "Your definition for tokenize_and_stem is wrong. Some common errors are us
ing filtered instead of filtered_tokens during stemming or repeating variable
 names that lead to variable override."

def test_correct_tokenize_and_stem():
    c_words_stemmed = tokenize_and_stem("Today (May 19, 2016) is his only daug
hter's wedding.")

    assert words_stemmed == c_words_stemmed, \
    "Please check you have correctly created your tokenize_and_stem() function
and stored the result of its call in words_stemmed."
```

Out[37]: 2/2 tests passed

6. Create TfidfVectorizer

Computers do not *understand* text. These are machines only capable of understanding numbers and performing numerical computation. Hence, we must convert our textual plot summaries to numbers for the computer to be able to extract meaning from them. One simple method of doing this would be to count all the occurrences of each word in the entire vocabulary and return the counts in a vector. Enter `CountVectorizer`.

Consider the word 'the'. It appears quite frequently in almost all movie plots and will have a high count in each case. But obviously, it isn't the theme of all the movies! Term Frequency-Inverse Document Frequency (https://campus.datacamp.com/courses/natural-language-processing-fundamentals-in-python/simple-topic-identification?ex=11) (TF-IDF) is one method which overcomes the shortcomings of `CountVectorizer`. The Term Frequency of a word is the measure of how often it appears in a document, while the Inverse Document Frequency is the parameter which reduces the importance of a word if it frequently appears in several documents.

For example, when we apply the TF-IDF on the first 3 sentences from the plot of *The Wizard of Oz*, we are told that the most important word there is 'Toto', the pet dog of the lead character. This is because the movie begins with 'Toto' biting someone due to which the journey of Oz begins!

In simplest terms, TF-IDF recognizes words which are unique and important to any given document. Let's create one for our purposes.

In [38]:
```
# Import TfidfVectorizer to create TF-IDF vectors
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))
```

In [39]:
```
%%nose

def test_tfidf_vectorizer_import():
    assert "TfidfVectorizer" in globals(), "Please check if you have correctly
imported the TfidfVectorizer."

def test_tfidf_stop_words():
    if "TfidfVectorizer" in globals():
        assert tfidf_vectorizer.__getattribute__("stop_words") == "english", \
        "You have to use the english stop words in the tfidf_vectorizer."
    else:
        assert 1==2, "You have to use the english stop words in the tfidf_vect
orizer."

def test_tfidf_tokenizer():
    if "TfidfVectorizer" in globals():
        assert tfidf_vectorizer.__getattribute__("tokenizer") == tokenize_and_
stem , \
        "You have to use the tokenize_and_stem function as the tokenizer in th
e tfidf_vectorizer."
    else:
        assert 1==2, "You have to use the tokenize_and_stem function as the to
kenizer in the tfidf_vectorizer."
```

Out[39]:
```
3/3 tests passed
```

7. Fit transform TfidfVectorizer

Once we create a TF-IDF Vectorizer, we must fit the text to it and then transform the text to produce the corresponding numeric form of the data which the computer will be able to understand and derive meaning from. To do this, we use the `fit_transform()` method of the `TfidfVectorizer` object.

If we observe the `TfidfVectorizer` object we created, we come across a parameter stopwords. 'stopwords' are those words in a given text which do not contribute considerably towards the meaning of the sentence and are generally grammatical filler words. For example, in the sentence 'Dorothy Gale lives with her dog Toto on the farm of her Aunt Em and Uncle Henry', we could drop the words 'her' and 'the', and still have a similar overall meaning to the sentence. Thus, 'her' and 'the' are stopwords and can be conveniently dropped from the sentence.

On setting the stopwords to 'english', we direct the vectorizer to drop all stopwords from a pre-defined list of English language stopwords present in the nltk module. Another parameter, `ngram_range` , defines the length of the ngrams to be formed while vectorizing the text.

In [40]:
```
# 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)
```

```
(100, 564)
```

In [41]:
```
%%nose

def test_tfidf_matrix_shape():
    assert tfidf_matrix.shape == (100, 564), \
    "Please make sure you have fit and transformed the training data in correc
t dimensions to the vectorizer and stored in tfidf_matrix."
```

Out[41]:
```
1/1 tests passed
```

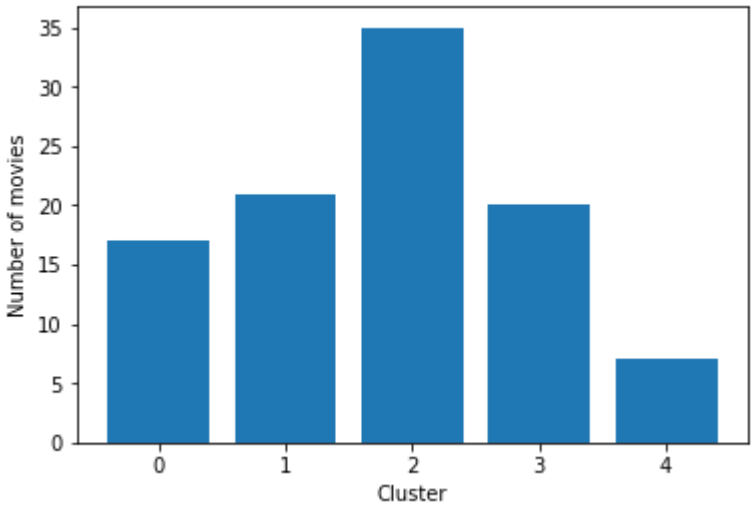
8. Import KMeans and create clusters

To determine how closely one movie is related to the other by the help of unsupervised learning, we can use clustering techniques. Clustering is the method of grouping together a number of items such that they exhibit similar properties. According to the measure of similarity desired, a given sample of items can have one or more clusters.

A good basis of clustering in our dataset could be the genre of the movies. Say we could have a cluster '0' which holds movies of the 'Drama' genre. We would expect movies like *Chinatown* or *Psycho* to belong to this cluster. Similarly, the cluster '1' in this project holds movies which belong to the 'Adventure' genre (*Lawrence of Arabia* and the *Raiders of the Lost Ark*, for example).

K-means is an algorithm which helps us to implement clustering in Python. The name derives from its method of implementation: the given sample is divided into ***K*** clusters where each cluster is denoted by the ***mean*** of all the items lying in that cluster.

We get the following distribution for the clusters:



In [42]:
```
# Import k-means to perform clustering
from sklearn.cluster import KMeans

# Create a KMeans object with 5 clusters and save as km
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[42]:
```
2    35
1    21
3    20
0    17
4     7
Name: cluster, dtype: int64
```

In [43]:
```
%%nose

def test_kmeans_imported():
    assert "KMeans" in globals(), "Please check if you have imported KMeans co
rrectly."

def test_num_clusters():
    assert km.n_clusters == 5, "Make sure you have set n_clusters to 5 in the
 k-means object km."

def test_labels():
    assert len(km.labels_.tolist()) == 100, \
    "Make sure you have correctly used the fit method of the km object to trai
n the k-means model."

def test_clusters_series():
    assert len(movies_df["cluster"]) == 100 and movies_df["cluster"].equals(p
d.Series(clusters)), \
    "Make sure you have correctly created a new column in the movies_df DataFr
ame with the name of 'cluster' which holds the predicted clusters from the k-m
eans object."
```

Out[43]:
```
4/4 tests passed
```

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 (https://en.wikipedia.org/wiki/Cosine_similarity). 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 [44]:
```
# 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)
```

In [45]:
```
%%nose

def test_cosine_similarity_import():
    assert "cosine_similarity" in globals(), \
    "Please check if you have correctly imported the cosine_similarity method
 from sklearn.metrics.pairwise."

def test_similary_distance():
    assert ("similarity_distance" in globals()) and (similarity_distance  ==
 (1 - cosine_similarity(tfidf_matrix))).all(), \
    "Check that you have correctly calculated and stored the similarity_distan
ce."
```

Out[45]:
```
2/2 tests passed
```

10. Import Matplotlib, Linkage, and Dendrograms

We shall now create a tree-like diagram (called a dendrogram) of the movie titles to help us understand the level of similarity between them visually. Dendrograms help visualize the results of hierarchical clustering, which is an alternative to k-means clustering. Two pairs of movies at the same level of hierarchical clustering are expected to have similar strength of similarity between the corresponding pairs of movies. For example, the movie *Fargo* would be as similar to *North By Northwest* as the movie *Platoon* is to *Saving Private Ryan*, given both the pairs exhibit the same level of the hierarchy.

Let's import the modules we'll need to create our dendrogram.

In [46]:
```
# Import matplotlib.pyplot for plotting graphs
import matplotlib.pyplot as plt

# Configure matplotlib to display the output inline
%matplotlib inline

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

```
DEBUG:matplotlib.pyplot:Loaded backend module://ipykernel.pylab.backend_inlin
e version unknown.
```

In [47]:
```
%%nose

def test_plt():
    assert "plt" in globals(), "Please check you have correctly imported matpl
otlib.pyplot as plt."

def test_linkage():
    assert "linkage" in globals(), "Please check you have correctly imported l
inkage from scipy.cluster.hierarchy."

def test_dendrogram():
    assert "dendrogram" in globals(), "Please check you have correctly importe
d dendrogram from scipy.cluster.hierarchy."
```

Out[47]:
```
3/3 tests passed
```

11. Create merging and plot dendrogram

We shall plot a dendrogram of the movies whose similarity measure will be given by the similarity distance we previously calculated. The lower the similarity distance between any two movies, the lower their linkage will make an intercept on the y-axis. For instance, the lowest dendrogram linkage we shall discover will be between the movies, *It's a Wonderful Life* and *A Place in the Sun*. This indicates that the movies are very similar to each other in their plots.

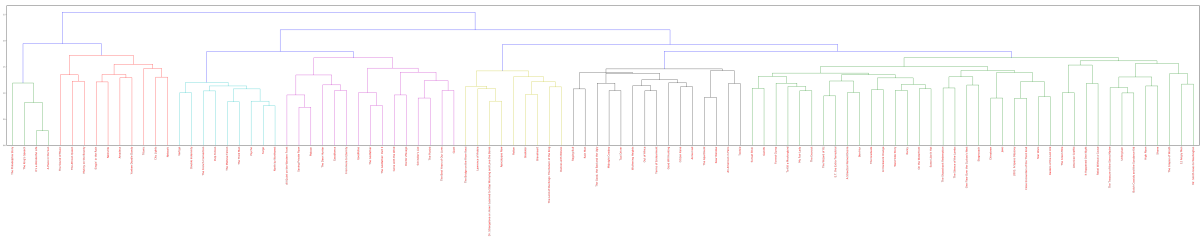
In [48]:
```
# Create mergings matrix
mergings = linkage(similarity_distance, method='complete')

# 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()
```

```
DEBUG:matplotlib.axes._base:update_title_pos
DEBUG:matplotlib.axes._base:update_title_pos
DEBUG:matplotlib.axes._base:update_title_pos
DEBUG:matplotlib.axes._base:update_title_pos
DEBUG:matplotlib.axes._base:update_title_pos
```



In [49]:
```
%%nose

def test_mergings():
    c_mergings = linkage(similarity_distance, method='complete')
    assert (mergings == c_mergings).all(), "Make sure you have correctly used
 the linkage method over similarity_distance and stored the result in merging
s."

def test_dendrogram():
    c_dendrogram = {'icoord': [[25.0, 25.0, 35.0, 35.0], [15.0, 15.0, 30.0, 3
0.0], [5.0, 5.0, 22.5, 22.5], [55.0, 55.0, 65.0, 65.0], [45.0, 45.0, 60.0, 60.
0], [75.0, 75.0, 85.0, 85.0], [95.0, 95.0, 105.0, 105.0], [80.0, 80.0, 100.0,
 100.0], [125.0, 125.0, 135.0, 135.0], [115.0, 115.0, 130.0, 130.0], [90.0, 9
0.0, 122.5, 122.5], [52.5, 52.5, 106.25, 106.25], [13.75, 13.75, 79.375, 79.37
5], [145.0, 145.0, 155.0, 155.0], [165.0, 165.0, 175.0, 175.0], [185.0, 185.0,
195.0, 195.0], [215.0, 215.0, 225.0, 225.0], [205.0, 205.0, 220.0, 220.0], [19
0.0, 190.0, 212.5, 212.5], [170.0, 170.0, 201.25, 201.25], [150.0, 150.0, 185.
625, 185.625], [245.0, 245.0, 255.0, 255.0], [235.0, 235.0, 250.0, 250.0], [27
5.0, 275.0, 285.0, 285.0], [265.0, 265.0, 280.0, 280.0], [242.5, 242.5, 272.5,
272.5], [305.0, 305.0, 315.0, 315.0], [295.0, 295.0, 310.0, 310.0], [325.0, 32
5.0, 335.0, 335.0], [345.0, 345.0, 355.0, 355.0], [365.0, 365.0, 375.0, 375.
0], [350.0, 350.0, 370.0, 370.0], [330.0, 330.0, 360.0, 360.0], [302.5, 302.5,
345.0, 345.0], [257.5, 257.5, 323.75, 323.75], [167.8125, 167.8125, 290.625, 2
90.625], [405.0, 405.0, 415.0, 415.0], [395.0, 395.0, 410.0, 410.0], [385.0, 3
85.0, 402.5, 402.5], [435.0, 435.0, 445.0, 445.0], [455.0, 455.0, 465.0, 465.
0], [440.0, 440.0, 460.0, 460.0], [425.0, 425.0, 450.0, 450.0], [393.75, 393.7
5, 437.5, 437.5], [475.0, 475.0, 485.0, 485.0], [505.0, 505.0, 515.0, 515.0],
 [495.0, 495.0, 510.0, 510.0], [535.0, 535.0, 545.0, 545.0], [525.0, 525.0, 54
0.0, 540.0], [565.0, 565.0, 575.0, 575.0], [555.0, 555.0, 570.0, 570.0], [532.
5, 532.5, 562.5, 562.5], [502.5, 502.5, 547.5, 547.5], [480.0, 480.0, 525.0, 5
25.0], [585.0, 585.0, 595.0, 595.0], [605.0, 605.0, 615.0, 615.0], [590.0, 59
0.0, 610.0, 610.0], [502.5, 502.5, 600.0, 600.0], [625.0, 625.0, 635.0, 635.
0], [665.0, 665.0, 675.0, 675.0], [655.0, 655.0, 670.0, 670.0], [645.0, 645.0,
662.5, 662.5], [630.0, 630.0, 653.75, 653.75], [685.0, 685.0, 695.0, 695.0],
 [705.0, 705.0, 715.0, 715.0], [690.0, 690.0, 710.0, 710.0], [725.0, 725.0, 73
5.0, 735.0], [745.0, 745.0, 755.0, 755.0], [765.0, 765.0, 775.0, 775.0], [750.
0, 750.0, 770.0, 770.0], [730.0, 730.0, 760.0, 760.0], [700.0, 700.0, 745.0, 7
45.0], [641.875, 641.875, 722.5, 722.5], [785.0, 785.0, 795.0, 795.0], [805.0,
805.0, 815.0, 815.0], [790.0, 790.0, 810.0, 810.0], [825.0, 825.0, 835.0, 835.
0], [845.0, 845.0, 855.0, 855.0], [865.0, 865.0, 875.0, 875.0], [850.0, 850.0,
870.0, 870.0], [830.0, 830.0, 860.0, 860.0], [800.0, 800.0, 845.0, 845.0], [68
2.1875, 682.1875, 822.5, 822.5], [885.0, 885.0, 895.0, 895.0], [905.0, 905.0,
 915.0, 915.0], [890.0, 890.0, 910.0, 910.0], [935.0, 935.0, 945.0, 945.0], [9
25.0, 925.0, 940.0, 940.0], [955.0, 955.0, 965.0, 965.0], [932.5, 932.5, 960.
0, 960.0], [985.0, 985.0, 995.0, 995.0], [975.0, 975.0, 990.0, 990.0], [946.2
5, 946.25, 982.5, 982.5], [900.0, 900.0, 964.375, 964.375], [752.34375, 752.34
375, 932.1875, 932.1875], [551.25, 551.25, 842.265625, 842.265625], [415.625,
 415.625, 696.7578125, 696.7578125], [229.21875, 229.21875, 556.19140625, 556.
19140625], [46.5625, 46.5625, 392.705078125, 392.705078125]], 'dcoord': [[0.0,
0.2787094302645489, 0.2787094302645489, 0.0], [0.0, 0.8172836789210608, 0.8172
836789210608, 0.2787094302645489], [0.0, 1.1894660657146754, 1.189466065714675
4, 0.8172836789210608], [0.0, 1.2259173682111164, 1.2259173682111164, 0.0],
 [0.0, 1.3472949604309772, 1.3472949604309772, 1.2259173682111164], [0.0, 1.21
00386272658268, 1.2100386272658268, 0.0], [0.0, 1.2910078363550863, 1.29100783
63550863, 0.0], [1.2100386272658268, 1.352735693873075, 1.352735693873075, 1.2
910078363550863], [0.0, 1.3015772300698019, 1.3015772300698019, 0.0], [0.0, 1.
```

```
4677321321279562, 1.4677321321279562, 1.3015772300698019], [1.352735693873075,
1.5473996472844698, 1.5473996472844698, 1.4677321321279562], [1.34729496043097
72, 1.719965964755748, 1.719965964755748, 1.5473996472844698], [1.189466065714
6754, 1.9432740781675348, 1.9432740781675348, 1.719965964755748], [0.0, 1.0028
170615552476, 1.0028170615552476, 0.0], [0.0, 1.0381649276863196, 1.0381649276
863196, 0.0], [0.0, 0.8314072653177778, 0.8314072653177778, 0.0], [0.0, 0.7553
538076145848, 0.7553538076145848, 0.0], [0.0, 0.8456374462267305, 0.8456374462
267305, 0.7553538076145848], [0.8314072653177778, 1.078117803037354, 1.0781178
03037354, 0.8456374462267305], [1.0381649276863196, 1.1330390511933564, 1.1330
390511933564, 1.078117803037354], [1.0028170615552476, 1.2022707135579105, 1.2
022707135579105, 1.1330390511933564], [0.0, 0.7303008576285346, 0.730300857628
5346, 0.0], [0.0, 0.9614233166122937, 0.9614233166122937, 0.7303008576285346],
[0.0, 1.049156445273212, 1.049156445273212, 0.0], [0.0, 1.1537880797876277, 1.
1537880797876277, 1.049156445273212], [0.9614233166122937, 1.344841441831146,
 1.344841441831146, 1.1537880797876277], [0.0, 0.7620024400941099, 0.762002440
0941099, 0.0], [0.0, 1.0061975163637775, 1.0061975163637775, 0.762002440094109
9], [0.0, 0.9635372238285962, 0.9635372238285962, 0.0], [0.0, 0.90453969444772
67, 0.9045396944477267, 0.0], [0.0, 1.0497803115093862, 1.0497803115093862, 0.
0], [0.9045396944477267, 1.2350006085207865, 1.2350006085207865, 1.04978031150
93862], [0.9635372238285962, 1.3980433268189865, 1.3980433268189865, 1.2350006
085207865], [1.0061975163637775, 1.475202709415688, 1.475202709415688, 1.39804
33268189865], [1.344841441831146, 1.6729103655212556, 1.6729103655212556, 1.47
5202709415688], [1.2022707135579105, 1.794890720368648, 1.794890720368648, 1.6
729103655212556], [0.0, 0.8430542664780709, 0.8430542664780709, 0.0], [0.0, 1.
0801505370386209, 1.0801505370386209, 0.8430542664780709], [0.0, 1.12892320812
02225, 1.1289232081202225, 1.0801505370386209], [0.0, 0.9756695249805762, 0.97
56695249805762, 0.0], [0.0, 1.117746492134165, 1.117746492134165, 0.0], [0.975
6695249805762, 1.2148735439153626, 1.2148735439153626, 1.117746492134165], [0.
0, 1.3118286525278535, 1.3118286525278535, 1.2148735439153626], [1.12892320812
02225, 1.4221979530778903, 1.4221979530778903, 1.3118286525278535], [0.0, 1.07
7729878786937, 1.077729878786937, 0.0], [0.0, 1.0467673241334103, 1.0467673241
334103, 0.0], [0.0, 1.1862349890360238, 1.1862349890360238, 1.046767324133410
3], [0.0, 1.043561733298193, 1.043561733298193, 0.0], [0.0, 1.138867958563079
7, 1.1388679585630797, 1.043561733298193], [0.0, 1.118901552905375, 1.11890155
2905375, 0.0], [0.0, 1.194119898248351, 1.194119898248351, 1.118901552905375],
[1.1388679585630797, 1.251497933066662, 1.251497933066662, 1.194119898248351],
[1.1862349890360238, 1.377234356406368, 1.377234356406368, 1.251497933066662],
[1.077729878786937, 1.387180497549022, 1.387180497549022, 1.377234356406368],
 [0.0, 0.9181769196839356, 0.9181769196839356, 0.0], [0.0, 1.1820658130840584,
1.1820658130840584, 0.0], [0.9181769196839356, 1.4269796609809122, 1.426979660
9809122, 1.1820658130840584], [1.387180497549022, 1.4600847124514604, 1.460084
7124514604, 1.4269796609809122], [0.0, 1.0885850135995854, 1.0885850135995854,
0.0], [0.0, 1.041713743061957, 1.041713743061957, 0.0], [0.0, 1.12444894947313
42, 1.1244489494731342, 1.041713743061957], [0.0, 1.187179607479755, 1.1871796
07479755, 1.1244489494731342], [1.0885850135995854, 1.3165764676063083, 1.3165
764676063083, 1.187179607479755], [0.0, 0.9453468874865079, 0.945346887486507
9, 0.0], [0.0, 1.0114306791873557, 1.0114306791873557, 0.0], [0.94534688748650
79, 1.22802959348862, 1.22802959348862, 1.0114306791873557], [0.0, 1.064386502
1895123, 1.0643865021895123, 0.0], [0.0, 1.0519126879398584, 1.051912687939858
4, 0.0], [0.0, 1.0798707957051075, 1.0798707957051075, 0.0], [1.05191268793985
84, 1.165312969004922, 1.165312969004922, 1.0798707957051075], [1.064386502189
5123, 1.287982124878887, 1.287982124878887, 1.165312969004922], [1.22802959348
862, 1.3511897654991216, 1.3511897654991216, 1.287982124878887], [1.3165764676
063083, 1.3800933906241102, 1.3800933906241102, 1.3511897654991216], [0.0, 1.0
951017490033617, 1.0951017490033617, 0.0], [0.0, 1.151957621128982, 1.15195762
1128982, 0.0], [1.0951017490033617, 1.3080681614123404, 1.3080681614123404, 1.
151957621128982], [0.0, 0.9021860525378849, 0.9021860525378849, 0.0], [0.0, 0.
```

9003093102219428, 0.9003093102219428, 0.0], [0.0, 0.9288388939307061, 0.9288388939307061, 0.0], [0.9003093102219428, 1.2177991442080265, 1.2177991442080265, 0.9288388939307061], [0.9021860525378849, 1.3919714080728927, 1.3919714080728927, 1.2177991442080265], [1.3080681614123404, 1.429667174344469, 1.429667174344469, 1.3919714080728927], [1.3800933906241102, 1.5050163486809163, 1.5050163486809163, 1.429667174344469], [0.0, 1.0052721386770818, 1.0052721386770818, 0.0], [0.0, 1.182747896800404, 1.182747896800404, 0.0], [1.0052721386770818, 1.5332815422756605, 1.5332815422756605, 1.182747896800404], [0.0, 1.0003206609986515, 1.0003206609986515, 0.0], [0.0, 1.1028431356087476, 1.1028431356087476, 1.0003206609986515], [0.0, 1.1351966178962787, 1.1351966178962787, 0.0], [1.1028431356087476, 1.3062337324254805, 1.3062337324254805, 1.1351966178962787], [0.0, 1.1712000424509748, 1.1712000424509748, 0.0], [0.0, 1.3710020543238846, 1.3710020543238846, 1.1712000424509748], [1.3062337324254805, 1.5646940153196518, 1.5646940153196518, 1.3710020543238846], [1.5332815422756605, 1.6123492188894006, 1.6123492188894006, 1.5646940153196518], [1.5050163486809163, 1.6716074509018797, 1.6716074509018797, 1.6123492188894006], [1.4600847124514604, 1.7983154751040344, 1.7983154751040344, 1.6716074509018797], [1.4221979530778903, 1.9282793755056098, 1.9282793755056098, 1.7983154751040344], [1.7948907203686848, 2.2056276774833523, 2.2056276774833523, 1.9282793755056098], [1.9432740781675348, 2.5432342313306777, 2.5432342313306777, 2.2056276774833523]], 'ivl': ['The Philadelphia Story', "The King's Speech", "It's a Wonderful Life", 'A Place in the Sun', 'The Sound of Music', 'The African Queen', 'Mutiny on the Bounty', "Singin' in the Rain", 'Nashville', 'Amadeus', 'Yankee Doodle Dandy', 'Titanic', 'City Lights', 'Network', 'Vertigo', 'Double Indemnity', 'The French Connection', 'Pulp Fiction', 'The Maltese Falcon', 'The Third Man', 'Psycho', 'Fargo', 'North by Northwest', 'All Quiet on the Western Front', 'Saving Private Ryan', 'Platoon', 'The Deer Hunter', 'Casablanca', 'From Here to Eternity', 'Goodfellas', 'The Godfather', 'The Godfather: Part II', 'Gone with the Wind', 'Doctor Zhivago', "Schindler's List", 'The Pianist', 'The Best Years of Our Lives', 'Giant', 'The Bridge on the River Kwai', 'Lawrence of Arabia', 'Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb', 'Apocalypse Now', 'Patton', 'Gladiator', 'Braveheart', 'The Lord of the Rings: The Return of the King', 'Dances with Wolves', 'Raging Bull', 'Rain Man', 'The Good, the Bad and the Ugly', 'Midnight Cowboy', 'Taxi Driver', 'Wuthering Heights', 'Out of Africa', 'Terms of Endearment', 'Good Will Hunting', 'Citizen Kane', 'Annie Hall', 'The Apartment', 'Rear Window', 'An American in Paris', 'Tootsie', 'Sunset Blvd.', 'Gandhi', 'Forrest Gump', 'To Kill a Mockingbird', 'My Fair Lady', 'The Exorcist', 'The Wizard of Oz', 'E.T. the Extra-Terrestrial', 'A Streetcar Named Desire', 'Ben-Hur', 'The Graduate', 'A Clockwork Orange', 'West Side Story', 'Rocky', 'On the Waterfront', 'Some Like It Hot', 'The Shawshank Redemption', 'The Silence of the Lambs', "One Flew Over the Cuckoo's Nest", 'Stagecoach', 'Chinatown', 'Jaws', '2001: A Space Odyssey', 'Close Encounters of the Third Kind', 'Star Wars', 'Raiders of the Lost Ark', 'The Green Mile', 'American Graffiti', 'It Happened One Night', 'Rebel Without a Cause', 'The Treasure of the Sierra Madre', 'Unforgiven', 'Butch Cassidy and the Sundance Kid', 'High Noon', 'Shane', 'The Grapes of Wrath', '12 Angry Men', 'Mr. Smith Goes to Washington'], 'leaves': [41, 65, 26, 67, 17, 87, 89, 25, 83, 30, 99, 9, 64, 82, 14, 94, 63, 86, 90, 97, 12, 76, 98, 62, 36, 55, 61, 4, 35, 59, 0, 11, 6, 47, 2, 58, 44, 77, 24, 10, 29, 31, 48, 34, 50, 33, 57, 3, 70, 51, 68, 92, 93, 72, 74, 73, 7, 71, 54, 96, 43, 75, 13, 32, 16, 42, 45, 60, 8, 20, 40, 46, 84, 91, 18, 39, 15, 27, 1, 22, 5, 88, 23, 49, 21, 81, 19, 38, 80, 85, 66, 95, 53, 37, 52, 56, 79, 78, 28, 69], 'color_list': ['g', 'g', 'g', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'b', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'm', 'm', 'm', 'm', 'm', 'm', 'm', 'm', 'm', 'm', 'm', 'm', 'm', 'm', 'b', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'k', 'k', 'k', 'k', 'k', 'k', 'k', 'k', 'k', 'k', 'k', 'k', 'k', 'k', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g',

```
 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'b',
 'b', 'b', 'b']}

    assert dendrogram_ == c_dendrogram, "Please make sure you have correctly c
reated the dendrogram."
```

Out[49]: 2/2 tests passed

12. Which movies are most similar?

We can now determine the similarity between movies based on their plots! To wrap up, let's answer one final question: which movie is most similar to the movie *Braveheart*?

In [50]:
```
# Answer the question
ans = "Gladiator"
print(ans)
```

```
Gladiator
```

In [51]:
```
%%nose

def test_ans():
    assert ans == "Gladiator", "Oops, your answer is incorrect! Please check t
he plot and your answer again."
```

Out[51]: 1/1 tests passed