Movies Recommendation

Supervised by

Dr: Hamada Nayel

Eng: Shimaa Esmail

Team Leader

Hazem Refai Mohamed Refai

Team Member

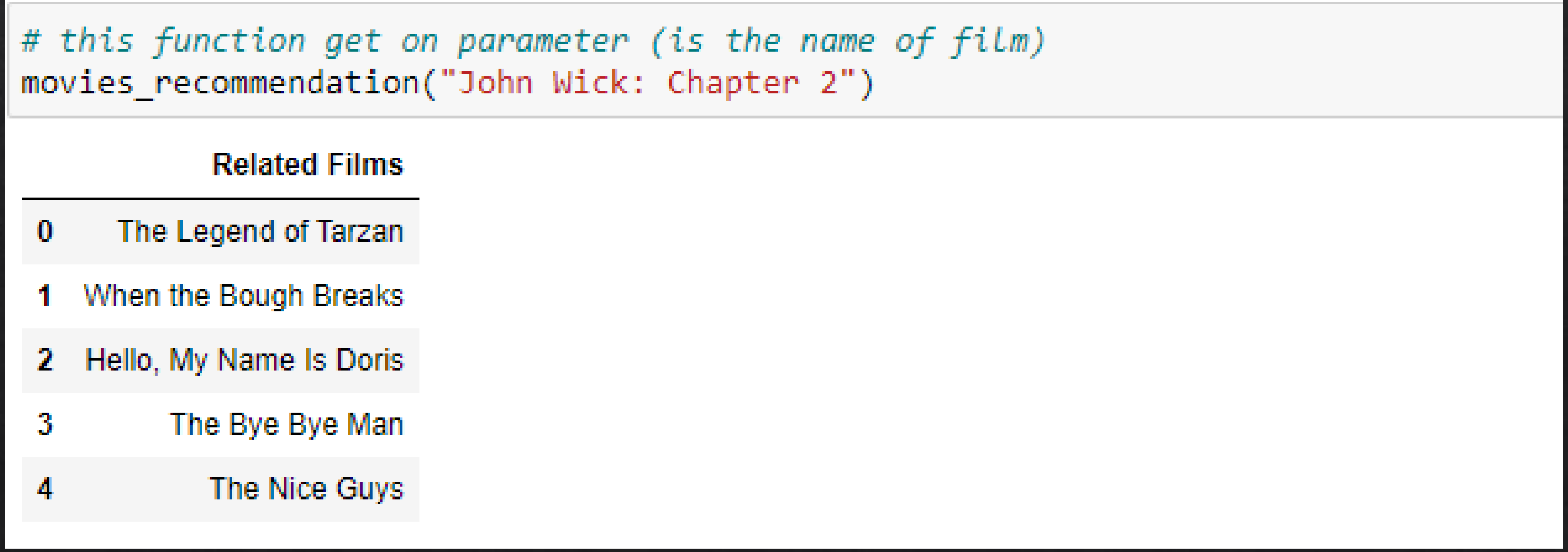
- Khaled Ahmed Slama Eldifrawy
- Eslam Khaled Soliman Gad
- Ibrahim Mousa El-Sayed Habib
- Ahmed Samer Eid Abdelhamid

Natural Language Processing (NLP)

- is a field in Artificial Intelligence (AI) devoted to creating computers that use natural language as input and or output.
- There are a common pipeline (Step-by-step Procedure) for build any model such as :
  1. Data Acquisition
  2. Text Cleaning
  3. Pre-Processing
  4. Feature Engineering
  5. Modeling

The Project Idea

- One of the application of NLP is similarity, it is the base of recommendation and movies recommendation.
- We apply similarity in project by TF-IDF and cosine similarity to calculate the similarity between films to get the movies in the same plot or same description.

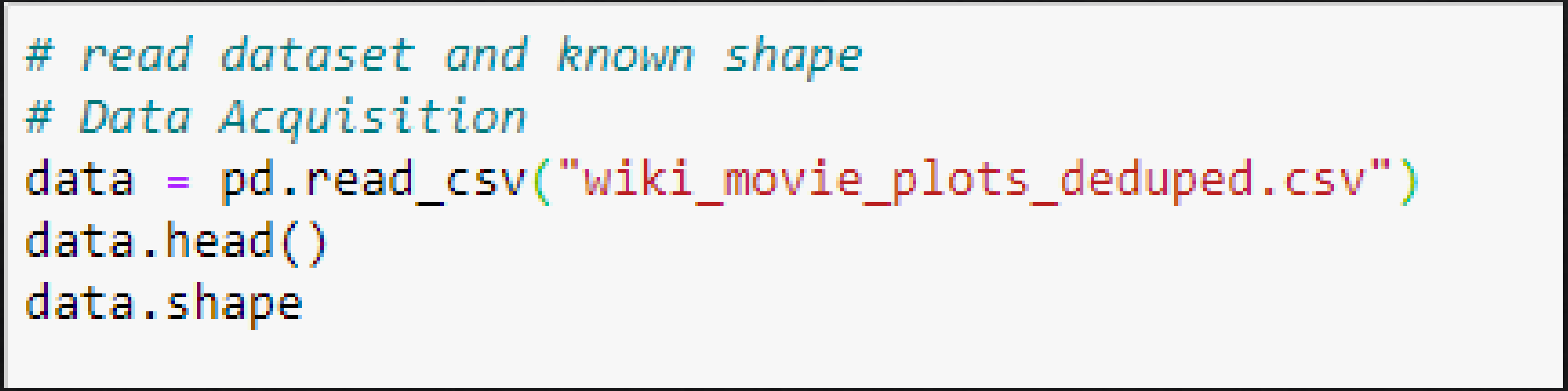


```
# this function get on parameter (is the name of film)
movies_recommendation("John Wick: Chapter 2")
```

	Related Films
0	The Legend of Tarzan
1	When the Bough Breaks
2	Hello, My Name Is Doris
3	The Bye Bye Man
4	The Nice Guys

Data Acquisition

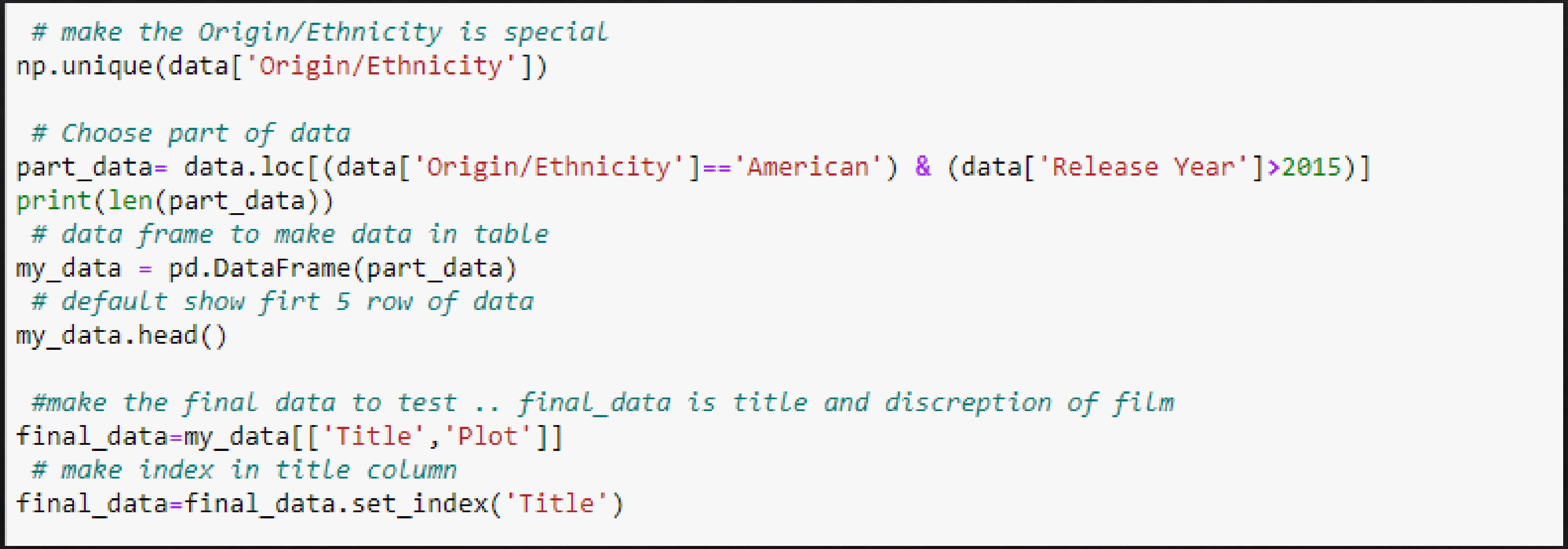
- We can get the data by :
  - Use public dataset using pandas library
    - Import pandas as pd



```
# read dataset and known shape
# Data Acquisition
data = pd.read_csv("wiki_movie_plots_deduped.csv")
data.head()
data.shape
```

Text Cleaning

- In this section we clean the data, text extraction and choose the part of data to test on it, split data and use the useful data for project.



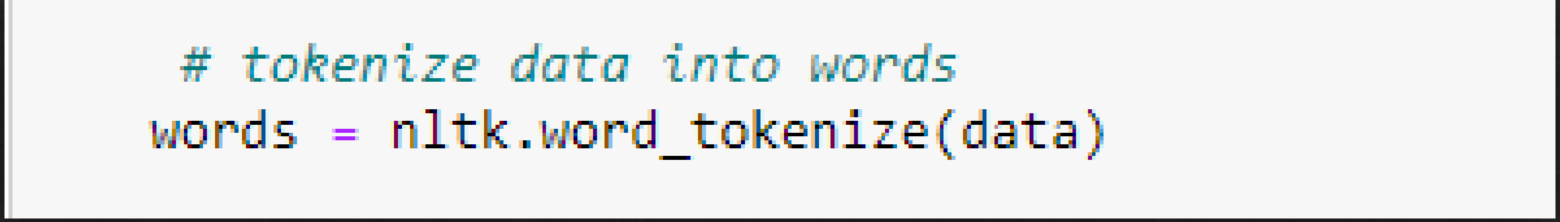
```
 # make the Origin/Ethnicity is special
np.unique(data['Origin/Ethnicity'])

 # Choose part of data
part_data= data.loc[(data['Origin/Ethnicity']=='American') & (data['Release Year']>2015)]
print(len(part_data))
 # data frame to make data in table
my_data = pd.DataFrame(part_data)
 # default show firt 5 row of data
my_data.head()

 #make the final data to test .. final_data is title and discreption of film
final_data=my_data[['Title','Plot']]
 # make index in title column
final_data=final_data.set_index('Title')
```

Pre-Processing

- Preliminaries

1. Sentence and word segmentation.

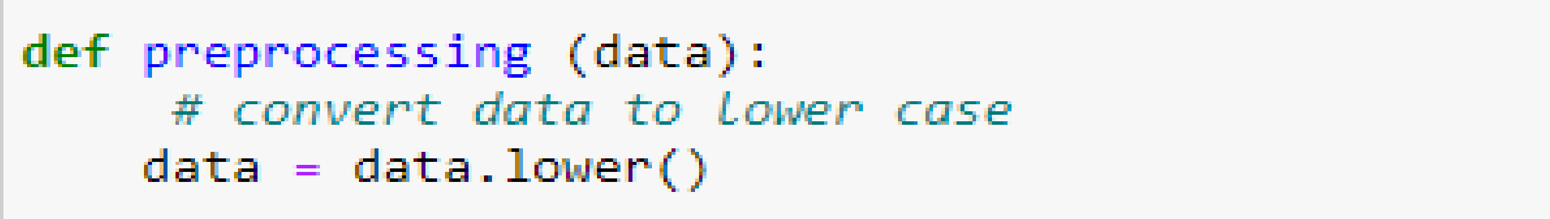
   - from nltk.tokenize import sent_tokenize, word_tokenize



```
 # tokenize data into words
words = nltk.word_tokenize(data)
```

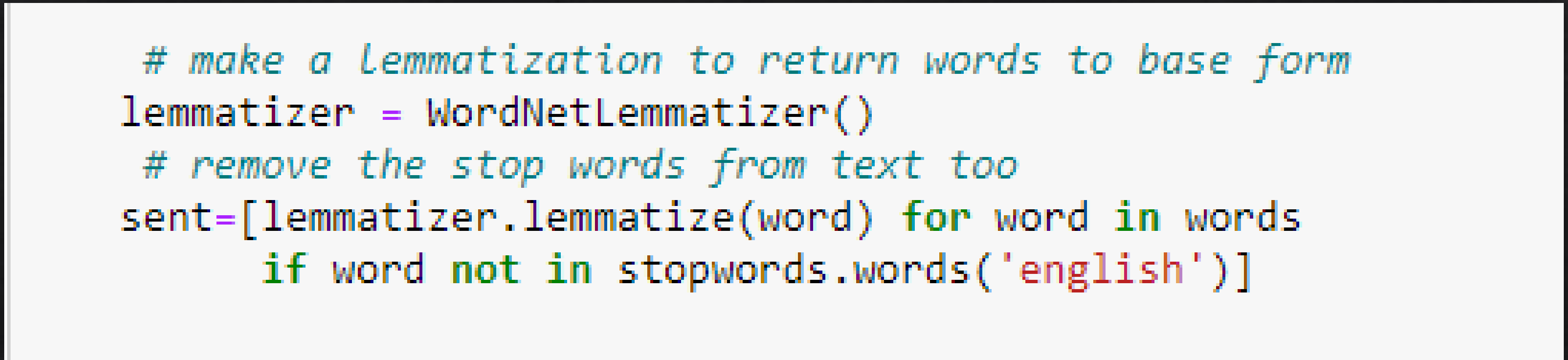
2. Frequent steps

   - Converts to lower case (English o english)



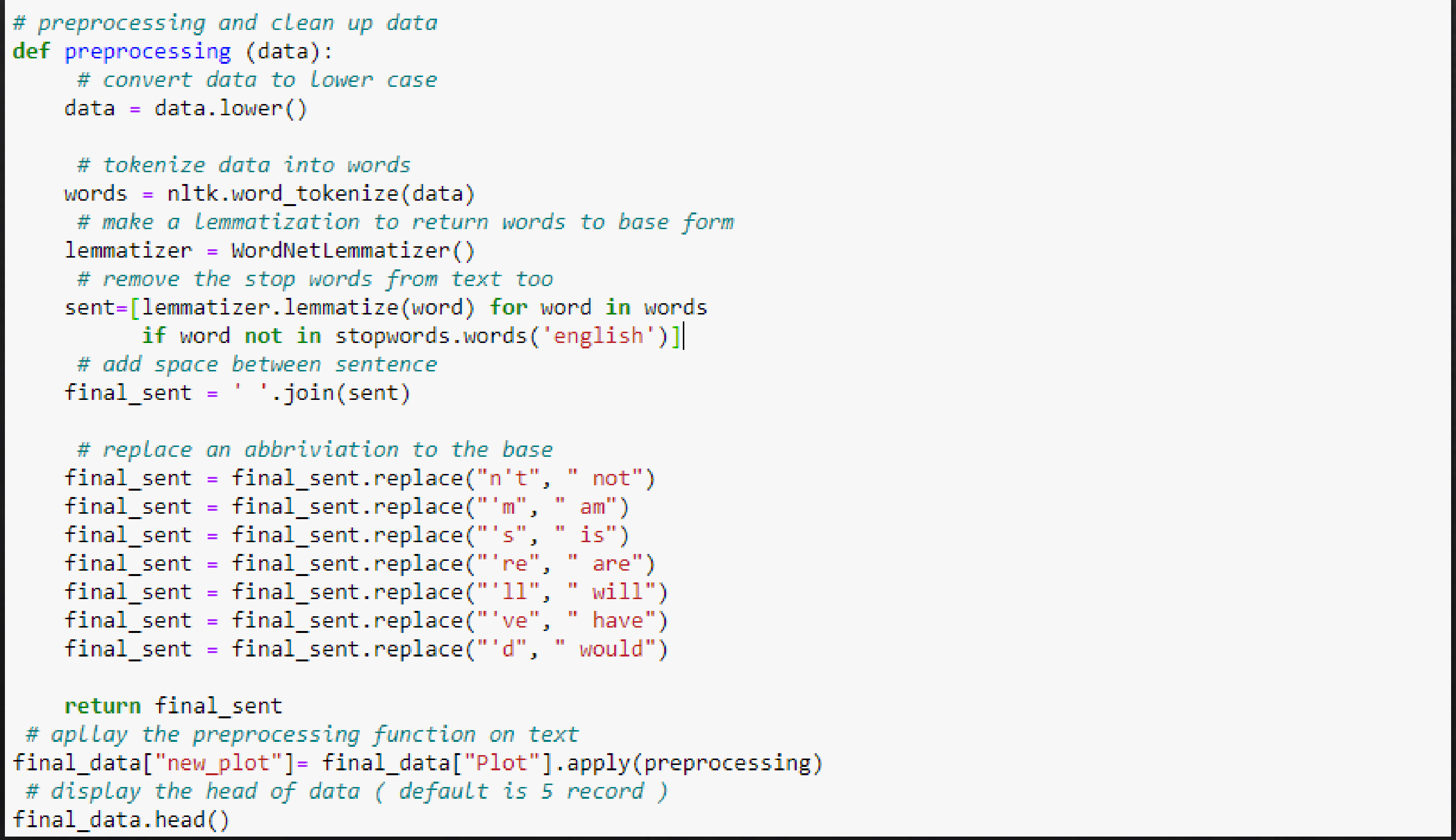
```
def preprocessing (data):
     # convert data to lower case
    data = data.lower()
```

- Lemmatization and Stop words removal
  - Lemmatization: Mapping all the different forms of a word to its base word or lemma
  - Stop word removal: remove word such as (at, in, or the, for,...)



```
 # make a lemmatization to return words to base form
lemmatizer = WordNetLemmatizer()
 # remove the stop words from text too
sent=[lemmatizer.lemmatize(word) for word in words
      if word not in stopwords.words('english')]
```

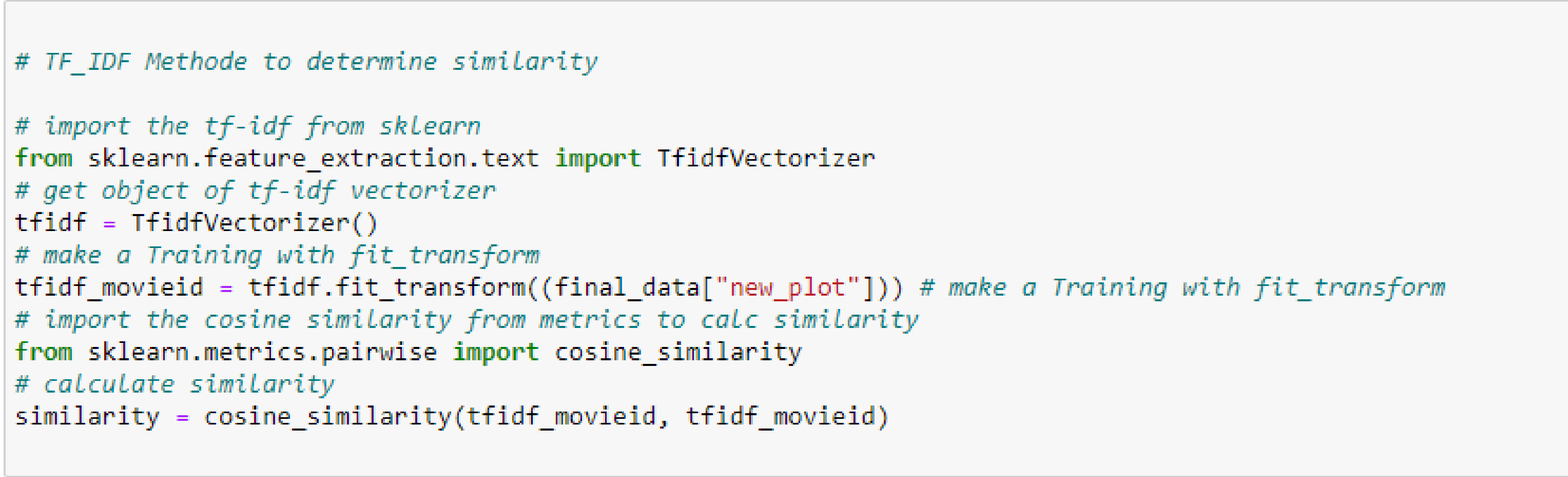
All Pre-Processing Function



```
# preprocessing and clean up data
def preprocessing (data):
     # convert data to lower case
    data = data.lower()

     # tokenize data into words
    words = nltk.word_tokenize(data)
     # make a lemmatization to return words to base form
    lemmatizer = WordNetLemmatizer()
     # remove the stop words from text too
    sent=[lemmatizer.lemmatize(word) for word in words
          if word not in stopwords.words('english')]
     # add space between sentence
    final_sent = ' '.join(sent)

     # replace an abbriviation to the base
    final_sent = final_sent.replace("n't", " not")
    final_sent = final_sent.replace("'m", " am")
    final_sent = final_sent.replace("'s", " is")
    final_sent = final_sent.replace("'re", " are")
    final_sent = final_sent.replace("'ll", " will")
    final_sent = final_sent.replace("'ve", " have")
    final_sent = final_sent.replace("'d", " would")

    return final_sent
 # apllay the preprocessing function on text
final_data["new_plot"]= final_data["Plot"].apply(preprocessing)
 # display the head of data ( default is 5 record )
final_data.head()
```

TF-IDF Technique

- Use TF-IDF (Term Frequency – Inverse Term Frequency) to determine the similarity between films by TfidfVectorizer().
- Calculate the similarity by import **cosine_similarty** module from **sklearn.metrics.pairwise**



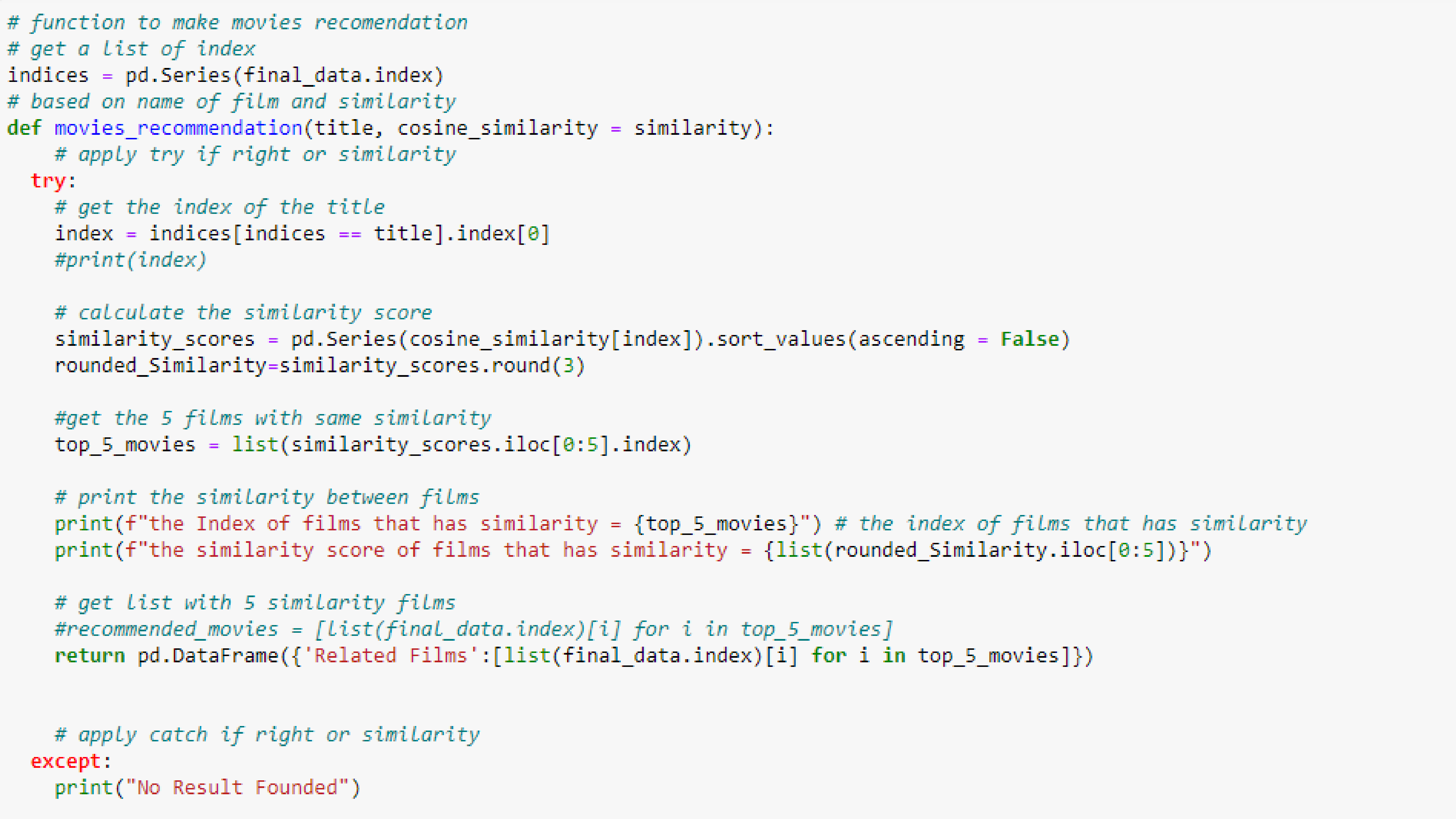
```
# TF_IDF Methode to determine similarity

# import the tf-idf from sklearn
from sklearn.feature_extraction.text import TfidfVectorizer
# get object of tf-idf vectorizer
tfidf = TfidfVectorizer()
# make a Training with fit_transform
tfidf_movieid = tfidf.fit_transform((final_data["new_plot"])) # make a Training with fit_transform
# import the cosine similarity from metrics to calc similarity
from sklearn.metrics.pairwise import cosine_similarity
# calculate similarity
similarity = cosine_similarity(tfidf_movieid, tfidf_movieid)
```

Movies Recommendation Function

- Set index to the record
- Give the function two parameters :

  1- film title

  2- similarity scores to determine similarity
- Calculate the similarity by the **cosine_similarity**
- Get the top five films with same plot or description
- Return DataFrame with top five films with high similarity in data set
- If films is not in dataset return **No Related Films**



```
# function to make movies recomendation
# get a list of index
indices = pd.Series(final_data.index)
# based on name of film and similarity
def movies_recommendation(title, cosine_similarity = similarity):
    # apply try if right or similarity
  try:
    # get the index of the title
    index = indices[indices == title].index[0]
    #print(index)

    # calculate the similarity score
    similarity_scores = pd.Series(cosine_similarity[index]).sort_values(ascending = False)
    rounded_Similarity=similarity_scores.round(3)

    #get the 5 films with same similarity
    top_5_movies = list(similarity_scores.iloc[0:5].index)

    # print the similarity between films
    print(f"the Index of films that has similarity = {top_5_movies}") # the index of films that has similarity
    print(f"the similarity score of films that has similarity = {list(rounded_Similarity.iloc[0:5])}")

    # get list with 5 similarity films
    #recommended_movies = [list(final_data.index)[i] for i in top_5_movies]
    return pd.DataFrame({'Related Films':[list(final_data.index)[i] for i in top_5_movies]})


    # apply catch if right or similarity
  except:
    print("No Result Founded")
```