**Movie Recommendation System**

`What is movie recommendation system ?

**A movie recommendation system** is a fancy way to describe a process that tries to predict your preferred items based on your or people similar to you.

In layman’s terms, we can say that a **Recommendation System** is a tool designed to predict/filter the items as per the user’s behavior.

**Types of Recommendation Systems:**

\* content Based: depends on content similarity.

\* collaborative: filtering based(depends on user interests).

\* hybrid based: both content and collaborative based.

We used content based similarity for this recommendation system.

The process followed in the movie recommendation system

1. Preparing data set - that contains required fields need with less no.of null values
2. Preprocessing data -removing null values
3. Model development and implementation.

### How to build a Movie Recommendation System using Machine Learning

The approach to build the movie recommendation engine consists of the following steps.

1. Perform Exploratory Data Analysis (EDA) on the data
2. Build the recommendation system
3. Get recommendations

#### Step 1: Perform Exploratory Data Analysis (EDA) on the data

The dataset contains two CSV files, credits, and movies. The credits file contains all the meta data information about the movie and the movie file contains the information like name and id of the movie, budget, languages in the movie that has been released, etc.

#### Step 2: Build the Movie Recommender System

The accuracy of predictions made by the recommendation system can be personalized using the “plot/description” of the movie.

Then the predictions should also include movies directed by the director of the film. It should also include movies with the cast of the given query movie.

#### Step 3: Get recommendations for the movies

The get\_recommendations() function takes the title of the movie and the similarity function as input. It follows the below steps to make recommendations.

* Get the index of the movie using the title.
* Get the list of similarity scores of the movies concerning all the movies.
* Enumerate them (create tuples) with the first element being the index and the second element is the cosine similarity score.
* Sort the list of tuples in descending order based on the similarity score.

***INPUT:***

Title of the movie and the similarity function as input.

(movies title,keywords,geners,cast)

***OUTPUT:***

Recommend the top similar movies.

***Analysis of the movie recommendations system***

*We import numpy and pandas libraries to work with linear algebra and dataset.*

**import numpy as np**

**import pandas as pd**

* Read the movie data set file into Jupyter notebook.

**movies = pd.read\_csv('tmdb\_5000\_movies.csv')**

**credits = pd.read\_csv('tmdb\_5000\_credits.csv')**

Since there are so many columns that are present in the dataset only needed columns are taken

**Movie Dataset:**

* **title**: Movie Title.
* **Overview**: Abstract of the Movie.
* **Popularity**: Movie popularity rating as per TMDB.
* **Vote\_average**: Votes average out of 10.
* **Vote\_count**: Number of votes from the users.
* **Release\_date**: Date of release of the movie.
* **Keywords**: Keywords for the movie by TMDB in the list.
* **Genres**: Movie Genres in the list.
* **Cast**: Cast of the movie on the list.
* **Crew**: Crew of the movie in the list.

CODE:

#importing the libraries

import numpy as np

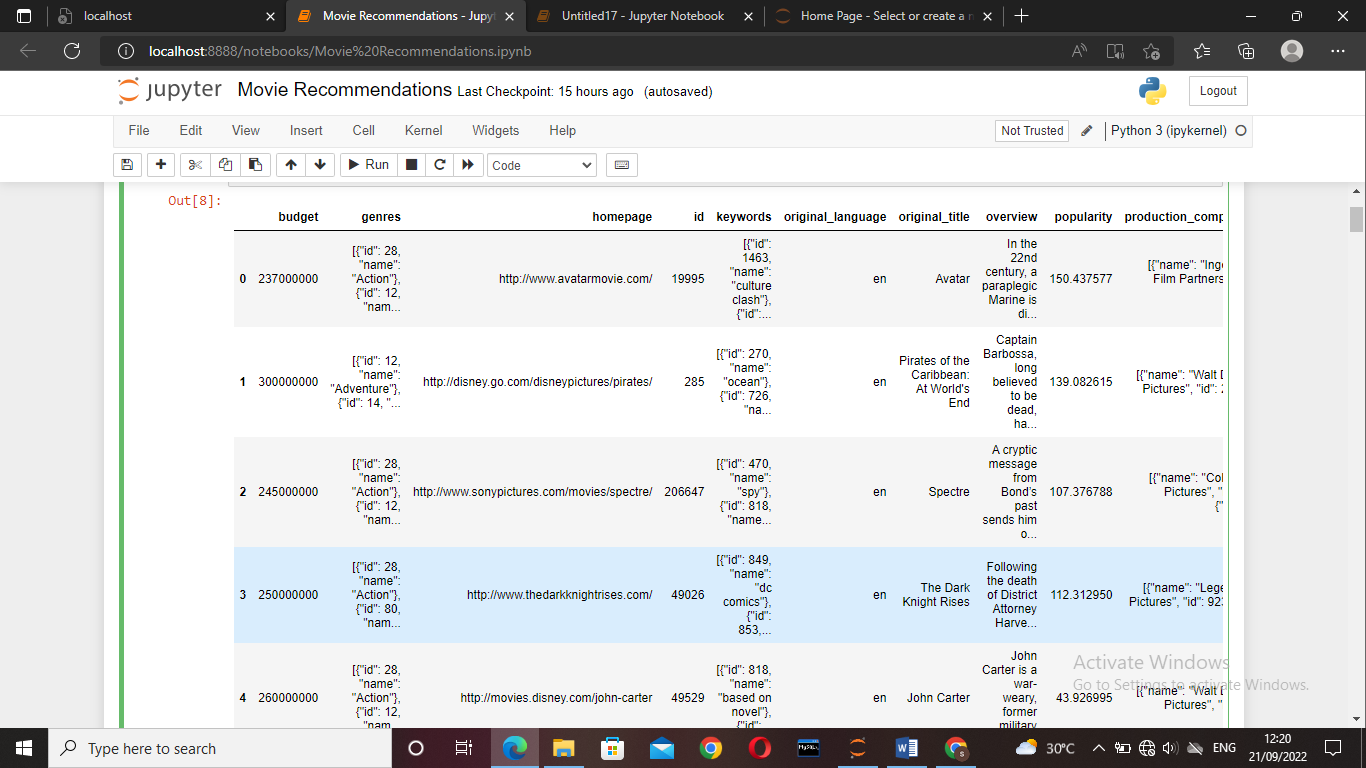
import pandas as pd

#importing the dataset

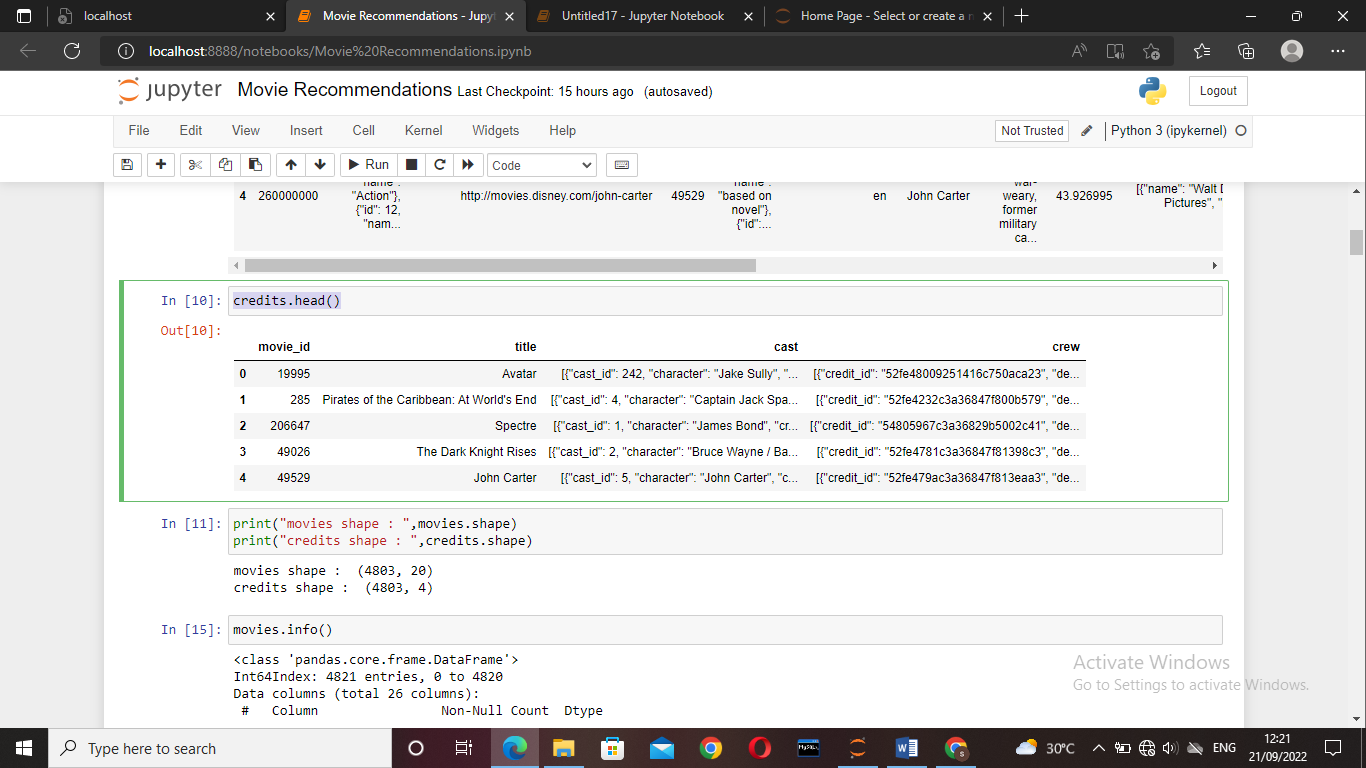
movies = pd.read\_csv("tmdb\_5000\_movies.csv")

credits = pd.read\_csv("tmdb\_5000\_credits.csv")

movies.head()

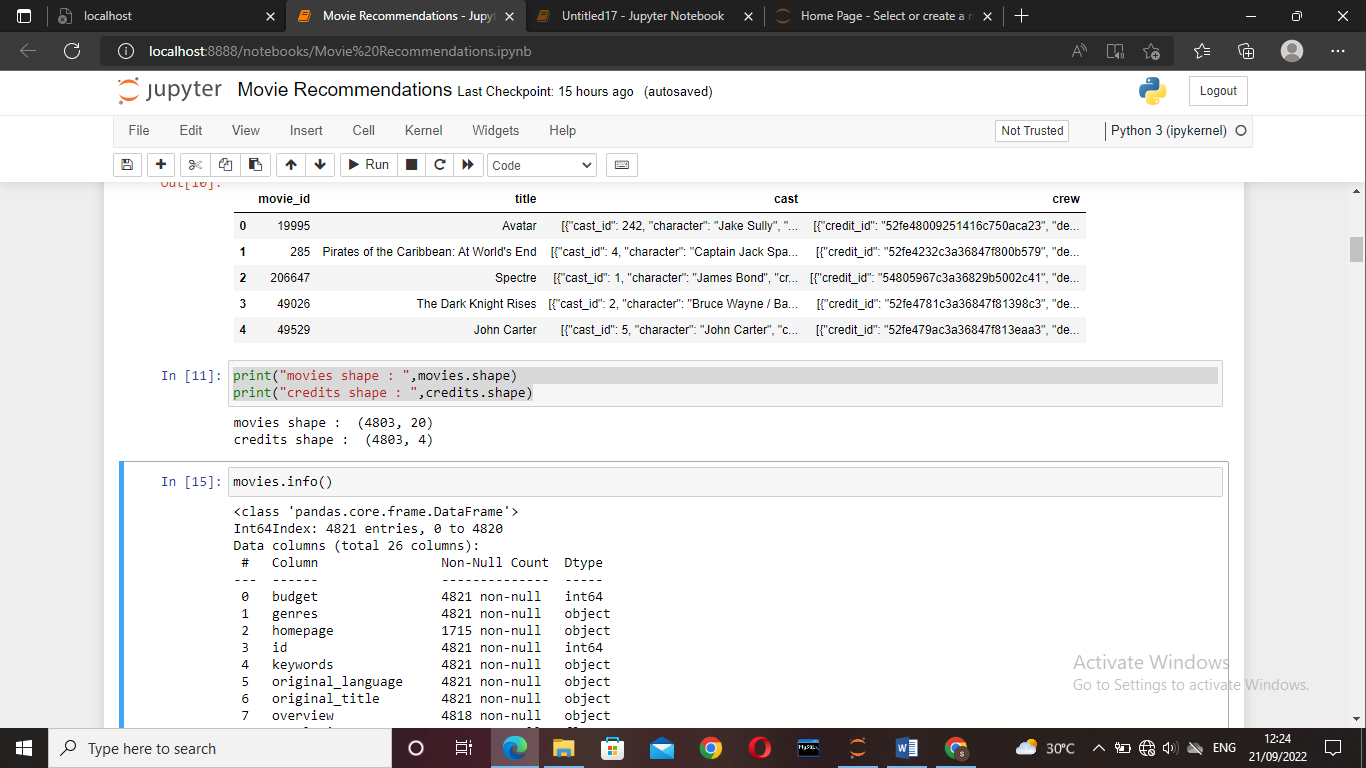


credits.head()

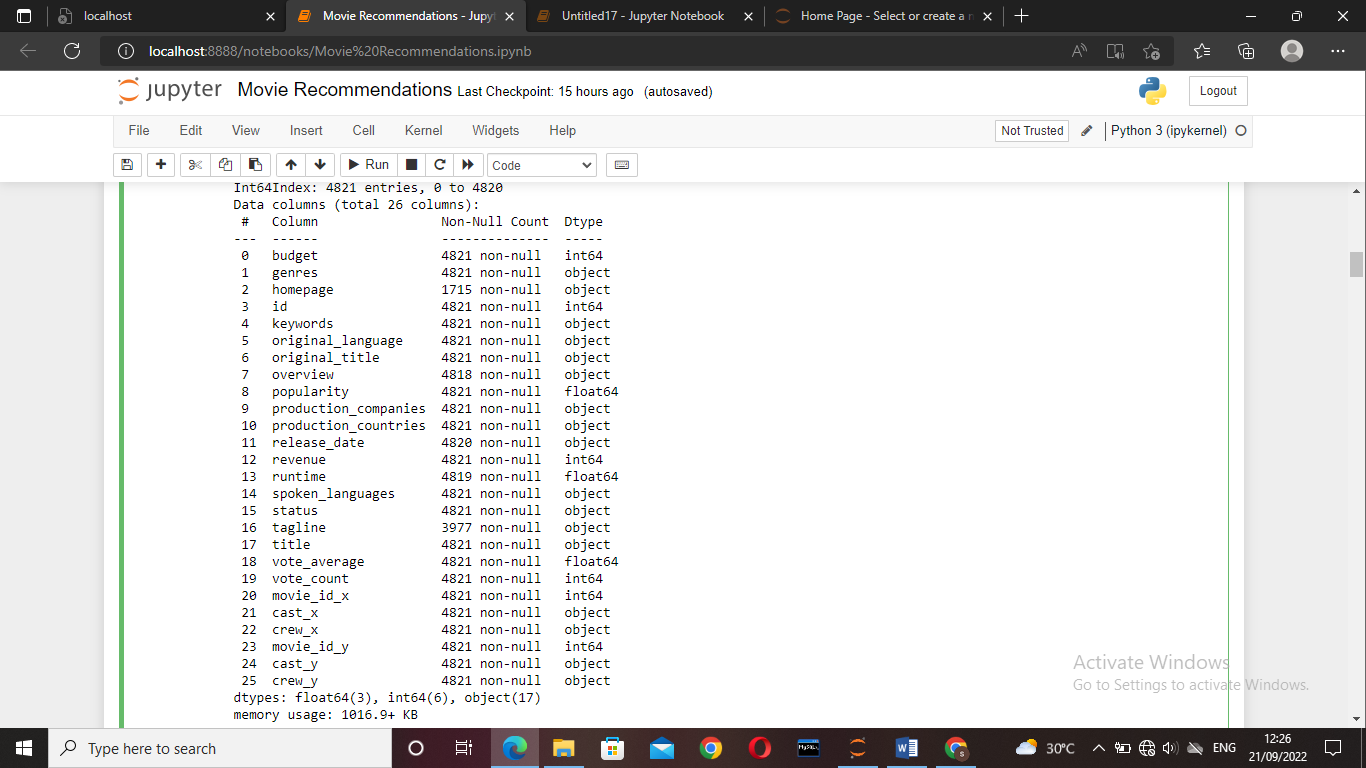


print("movies shape : ",movies.shape)

print("credits shape : ",credits.shape)



movies.info()



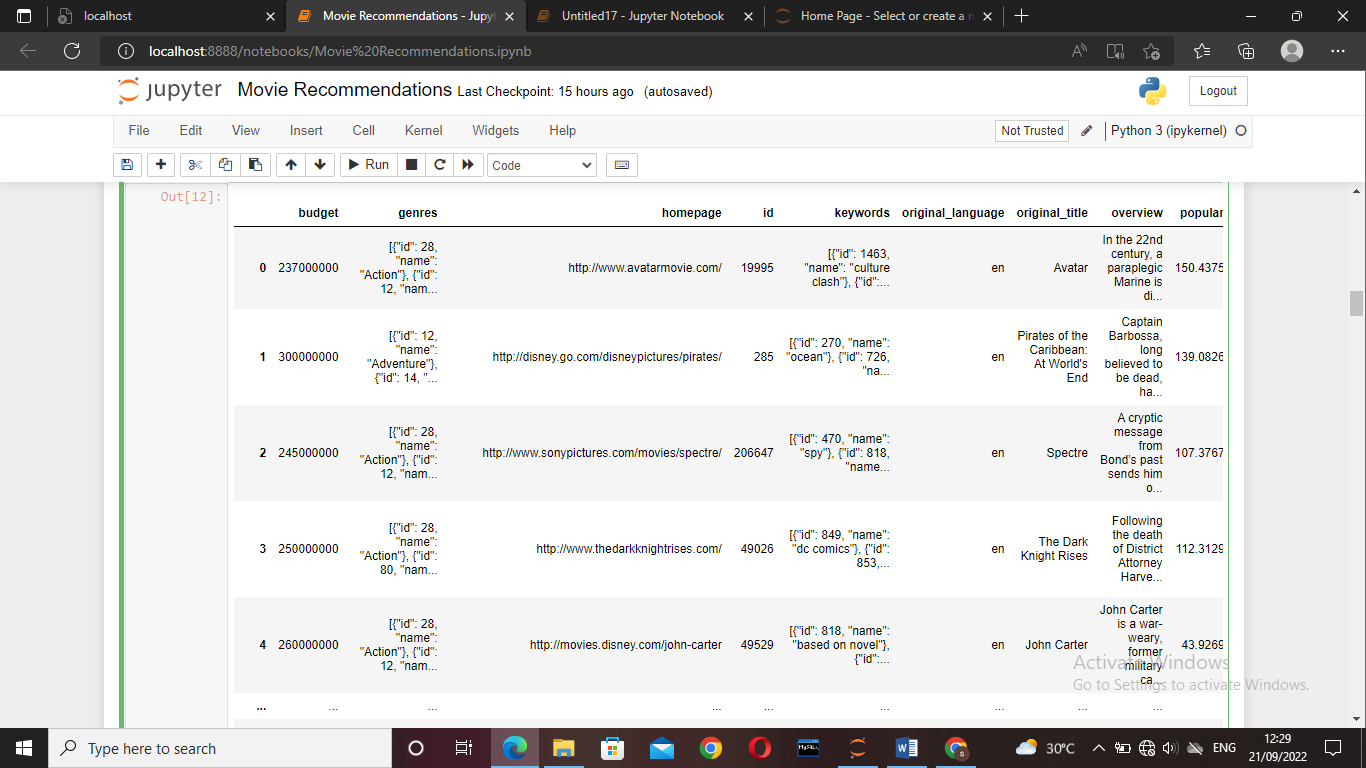
credits.info()

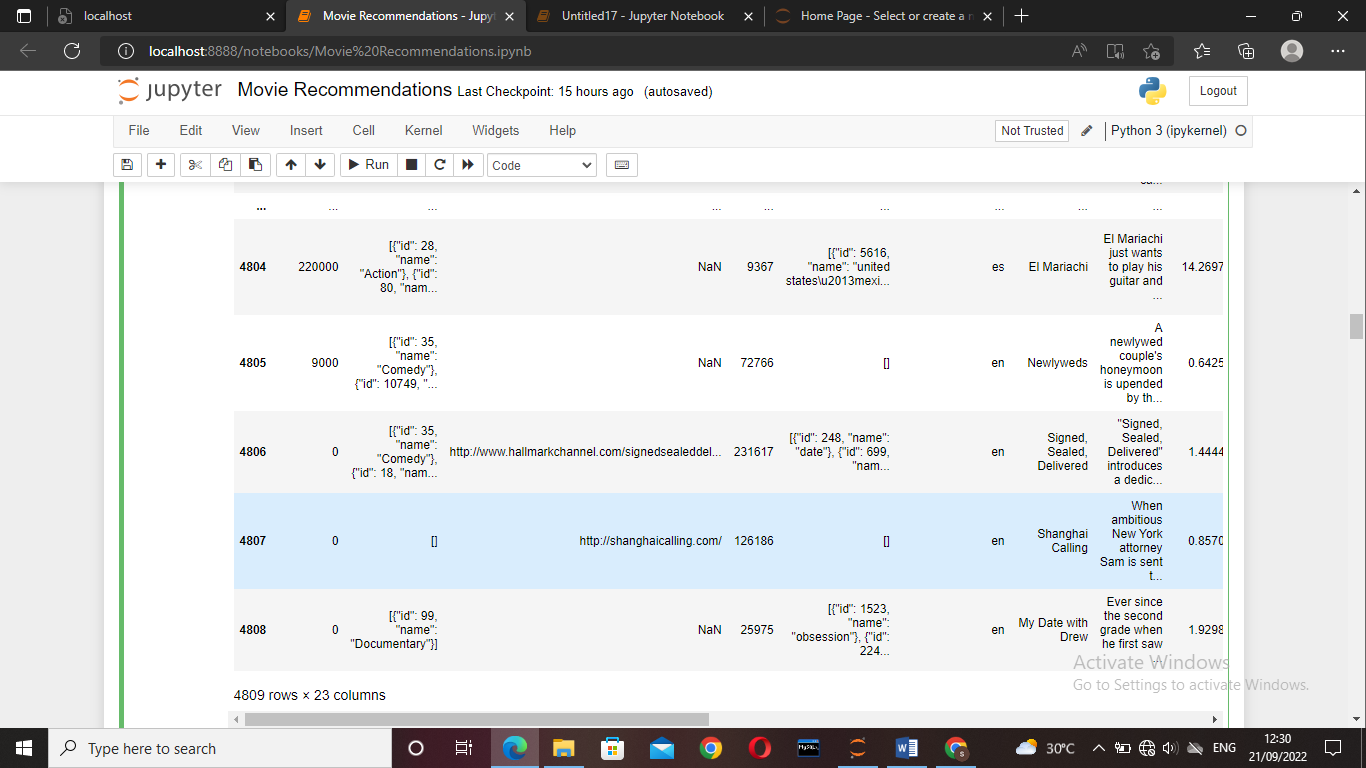


* Since there are two files we merge them, as it is easy to use. These data set can be merge using both title and Id.

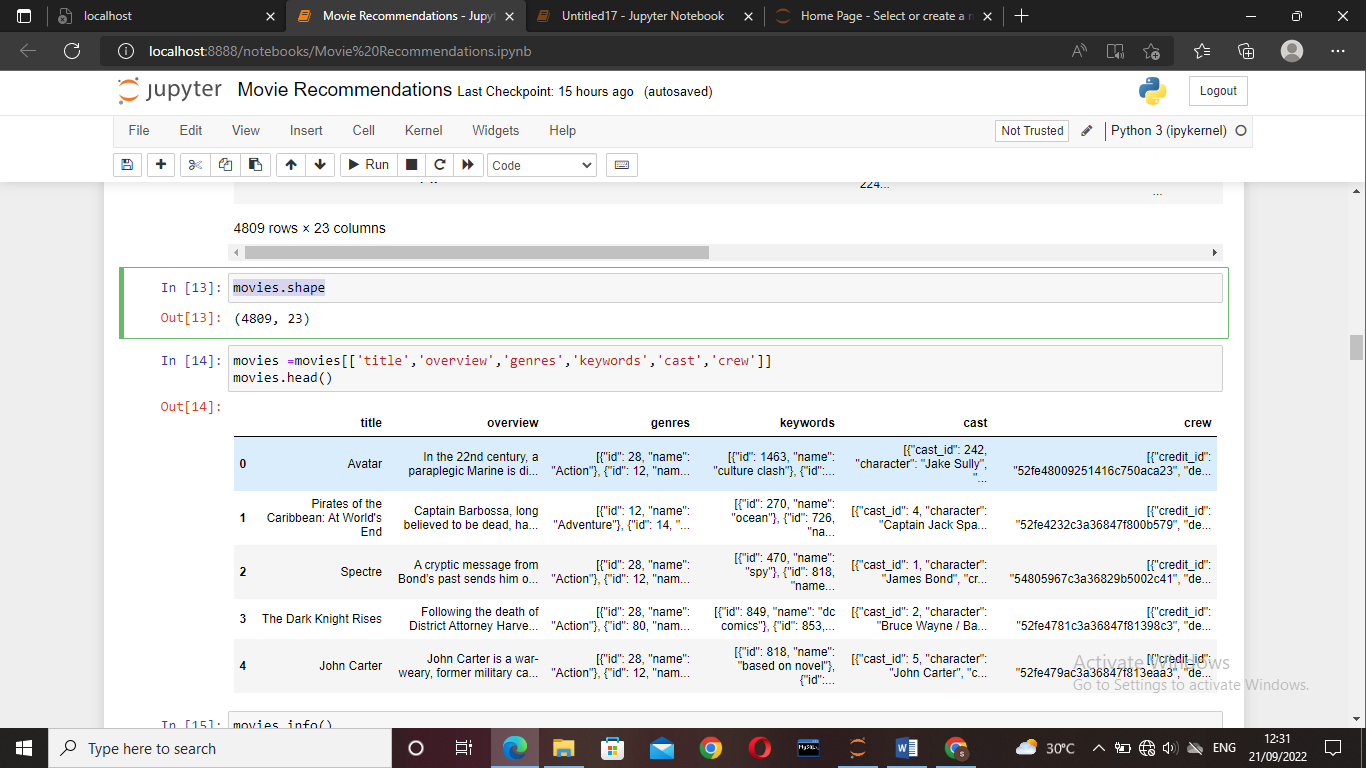
movies = movies.merge(credits,on='title')

movies





movies.shape



* movies = movies[['title','overview','genres','keywords','cast','crew']]

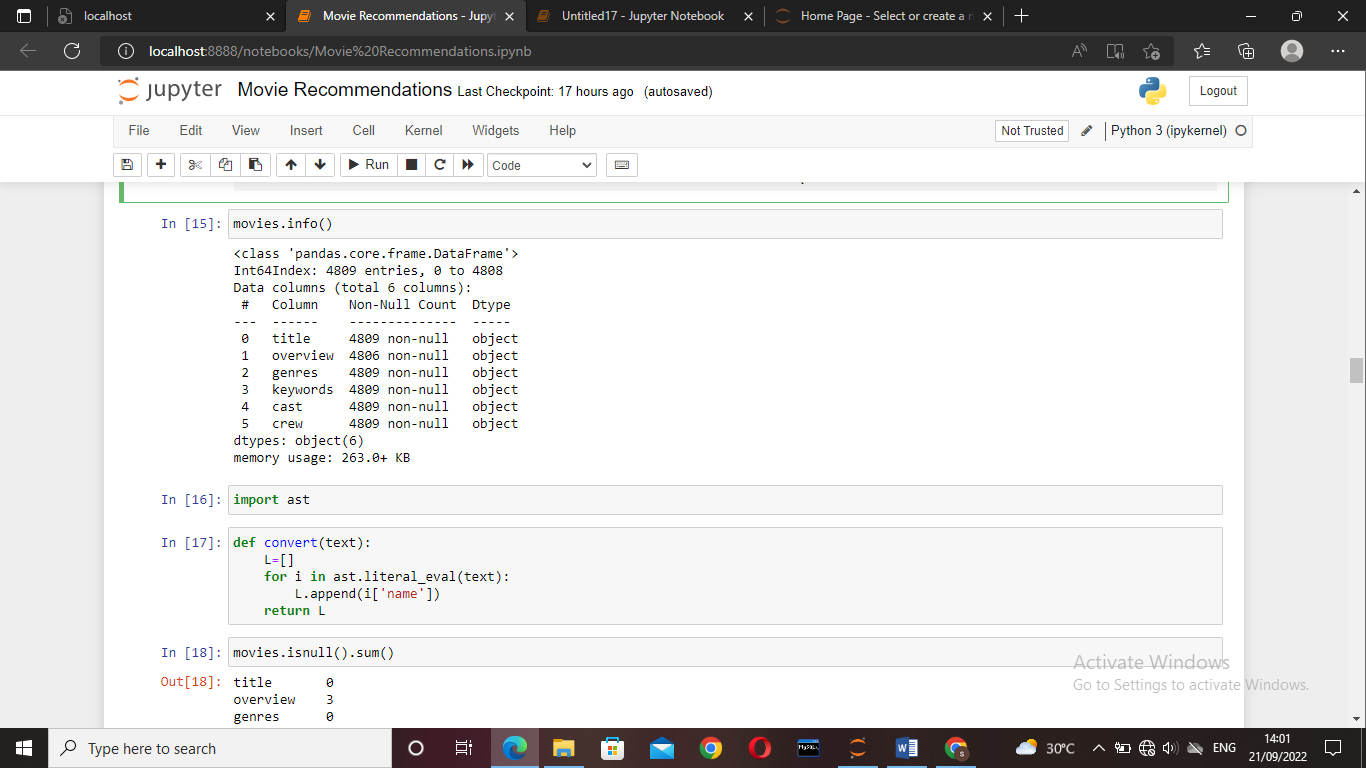
Only the need columns are taken.

movies =movies[['title','overview','genres','keywords','cast','crew']]

movies.head()



movies.info()



#Abstaract syntax tree

#interact with python code itself and it can modified with

Import ast

def convert(text):

L=[]

#literal\_eval function helps to find the type of value in a stored in a file

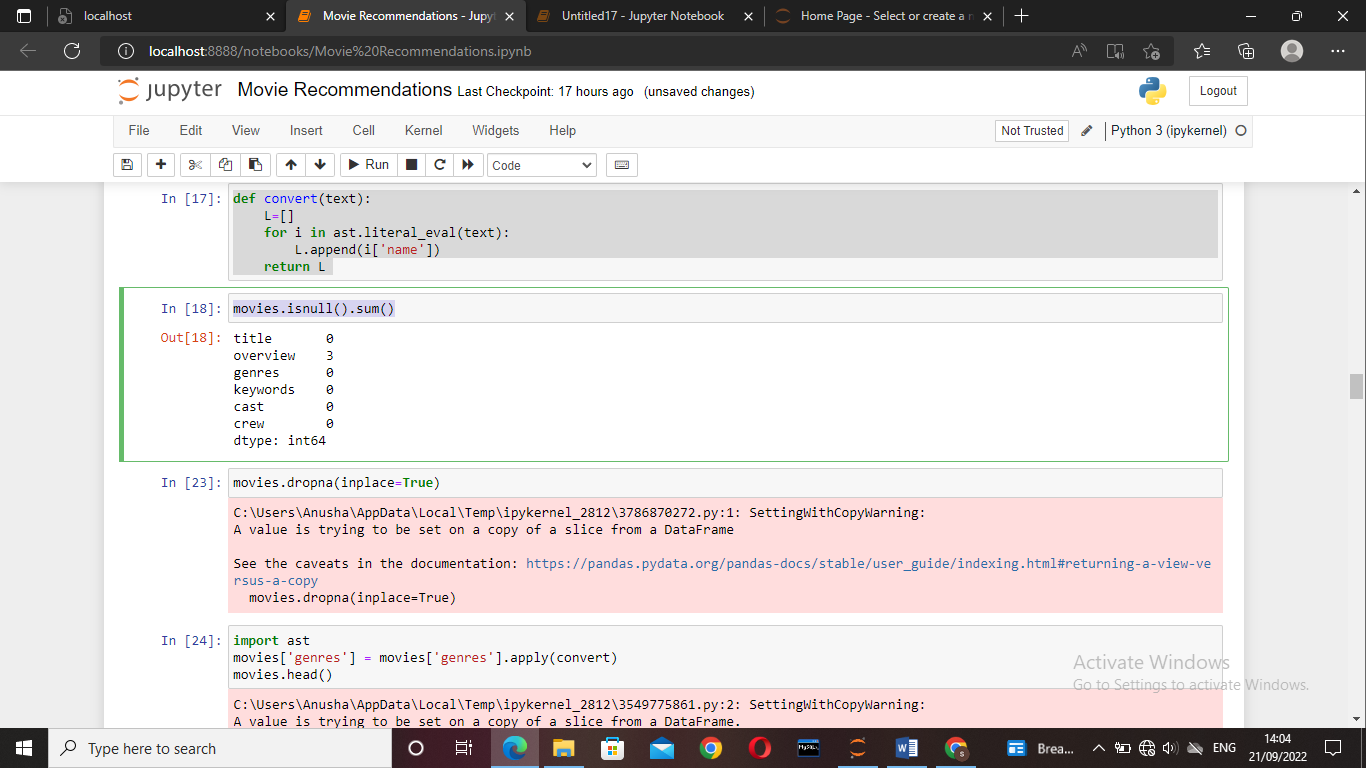
for i in ast.literal\_eval(text):

L.append(i['name'])

return L

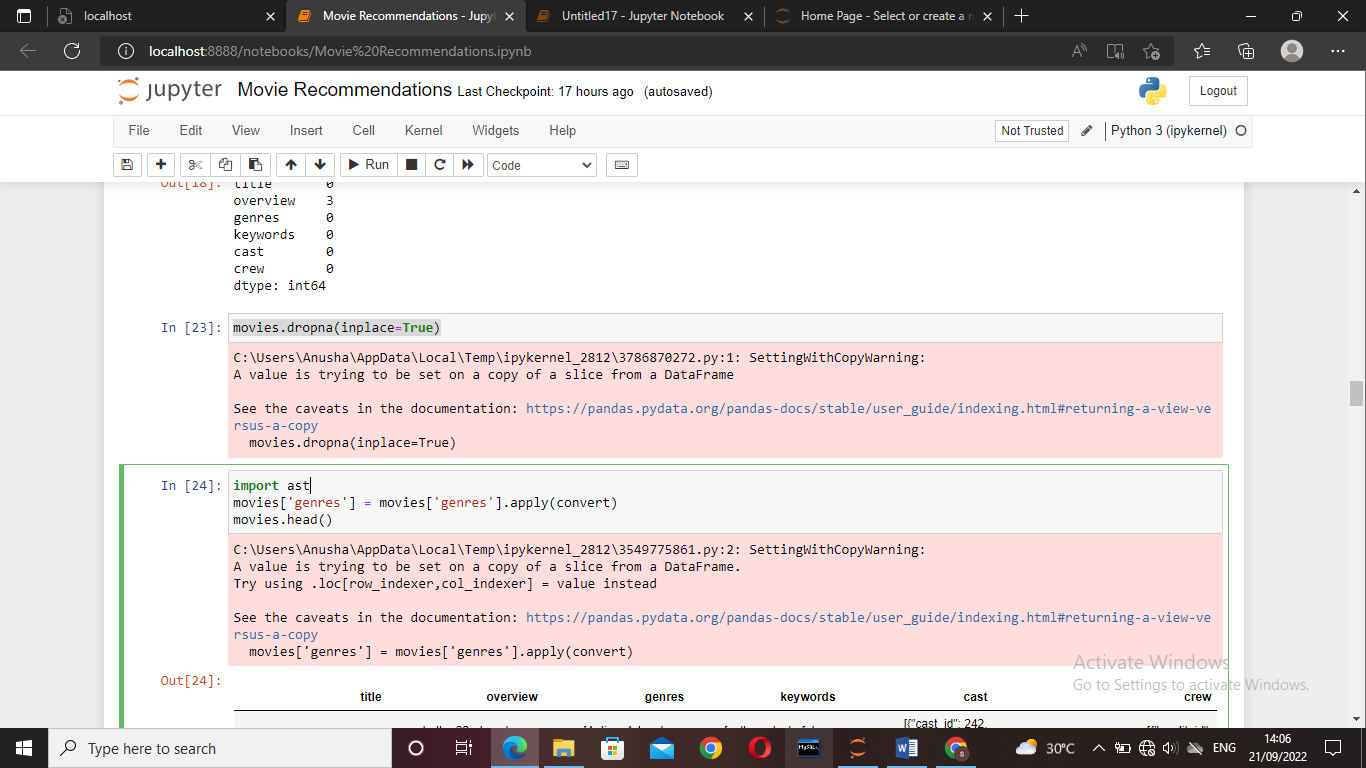
* To find the columns which has null values movies.isnull().sum() is used.

movies.isnull().sum()



Since in the data the null values are very less dropna() function is used.

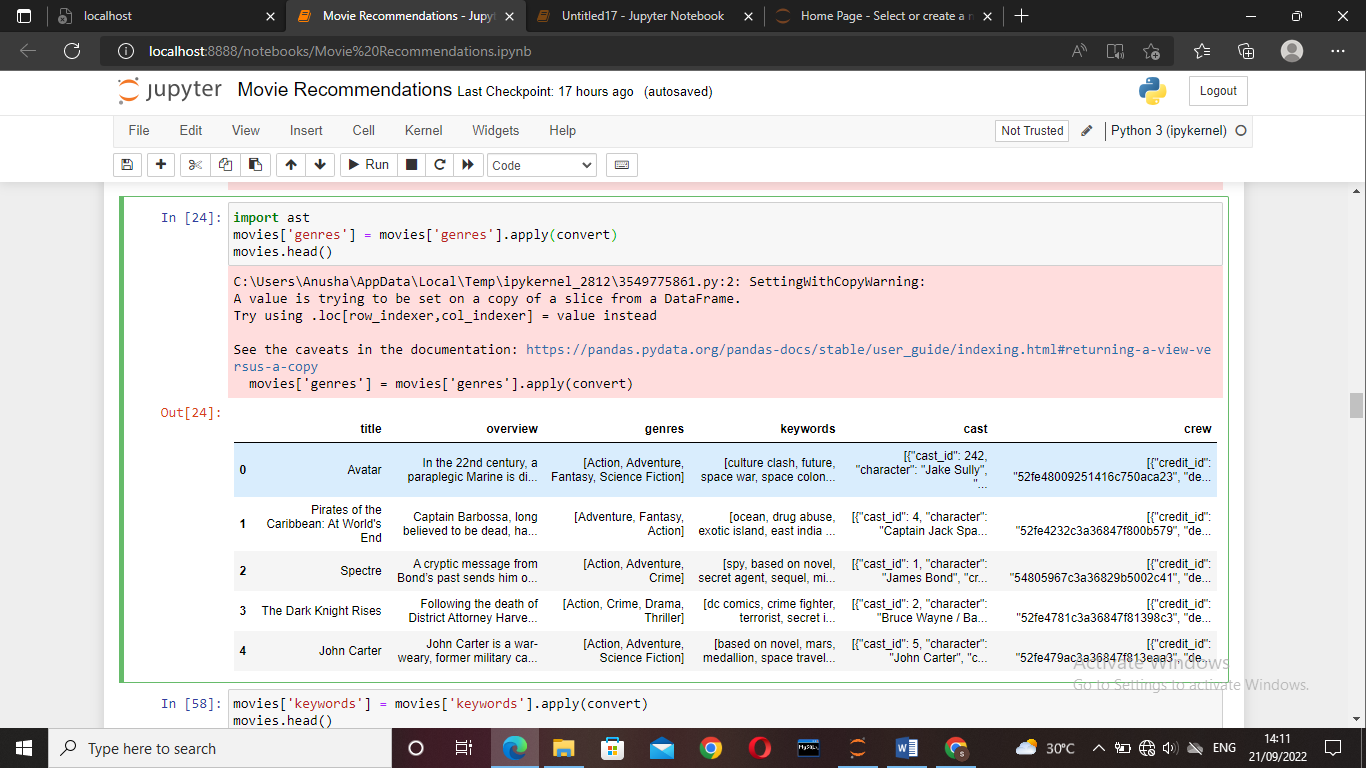
movies.dropna(inplace=True)



import ast

movies['genres'] = movies['genres'].apply(convert)

movies.head()



movies['keywords'] = movies['keywords'].apply(convert)

movies.head()

def convert3(text):

L = []

counter = 0

for i in ast.literal\_eval(text):

if counter < 3:

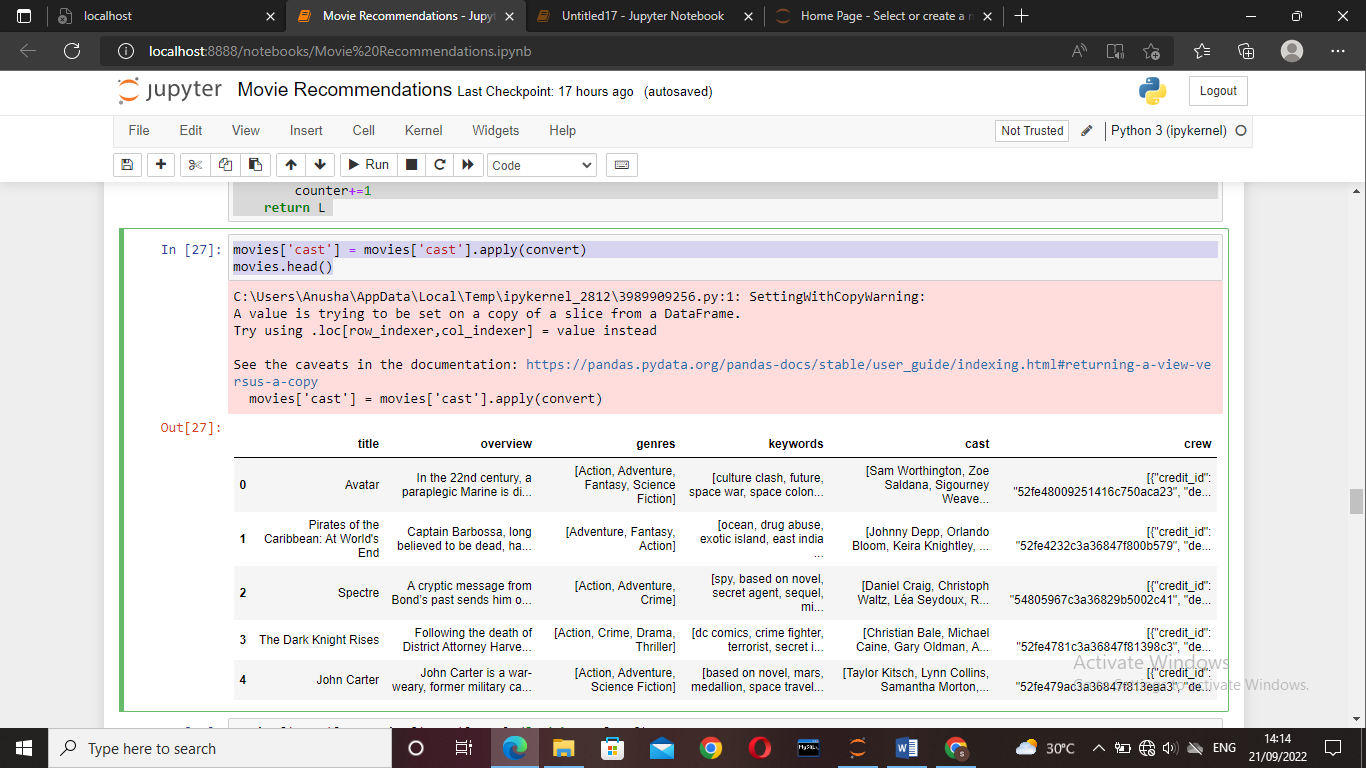
L.append(i['name'])

counter+=1

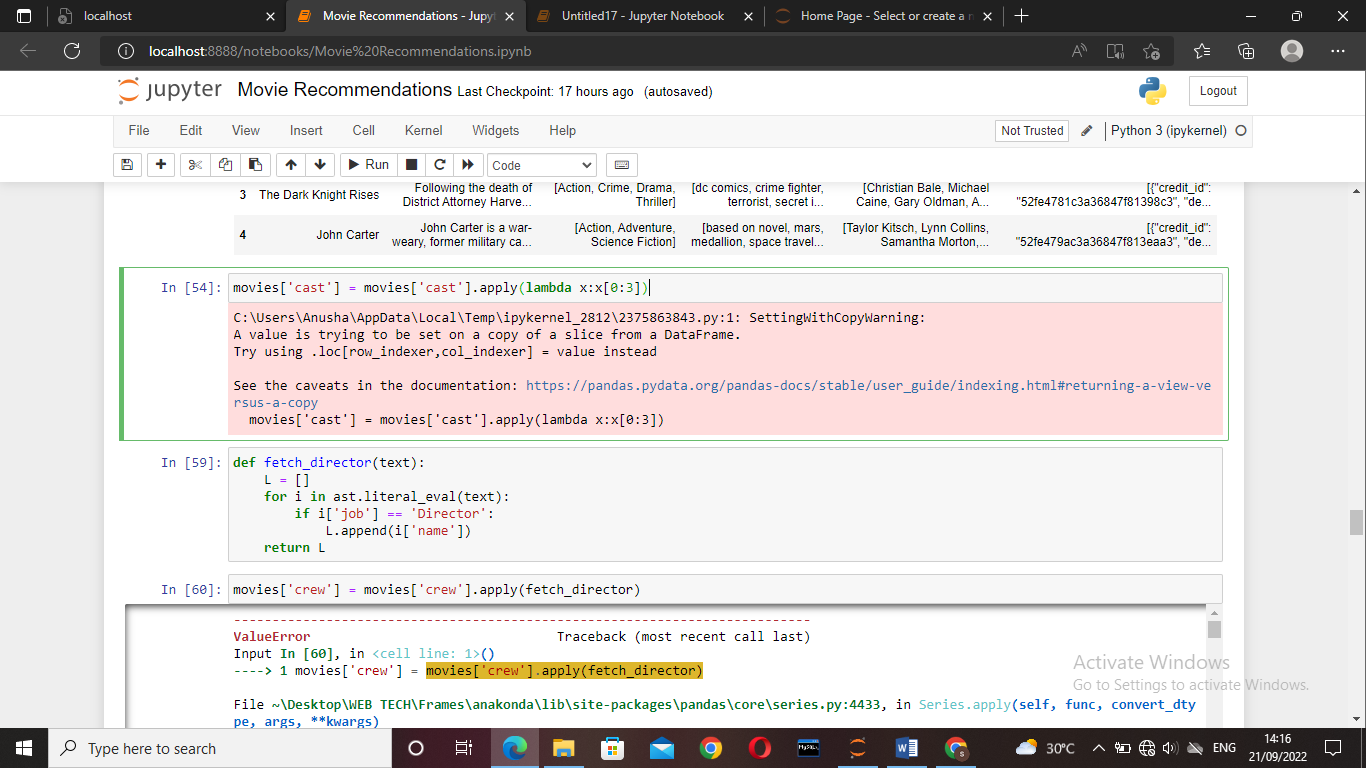
return L

movies['cast'] = movies['cast'].apply(convert)

movies.head()



movies['cast'] = movies['cast'].apply(lambda x:x[0:3])



*Most of the dataset are in dictionary with different types of data in it. But only one type of data is needed so to extract the data function is written but to extract the names because the indices are not integers*

* So ast is imported and literal\_eval is used The all the needed keywords are extracted from every column.

def fetch\_director(text):

L = []

for i in ast.literal\_eval(text):

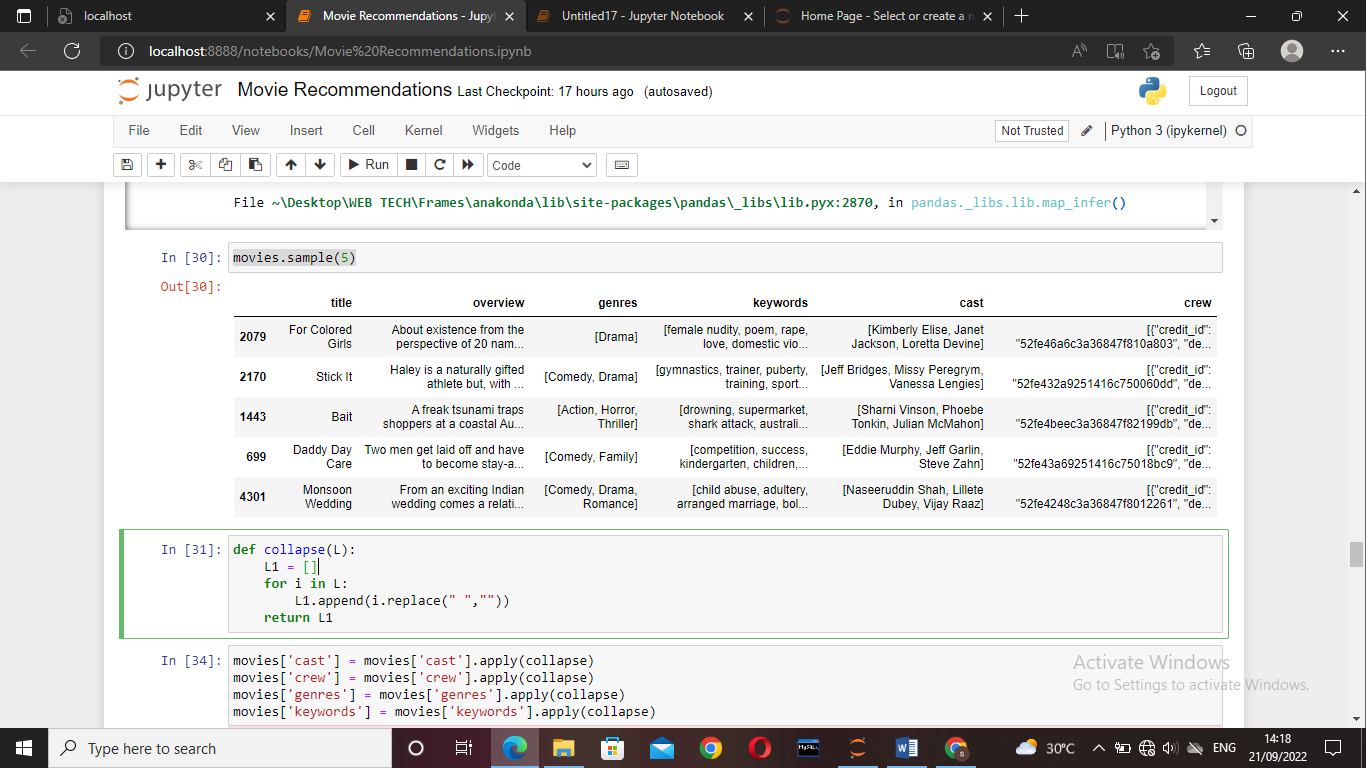
if i['job'] == 'Director':

L.append(i['name'])

return L

movies['crew'] = movies['crew'].apply(fetch\_director)

movies.sample(5)



def collapse(L):

L1 = []

for i in L:

L1.append(i.replace(" ",""))

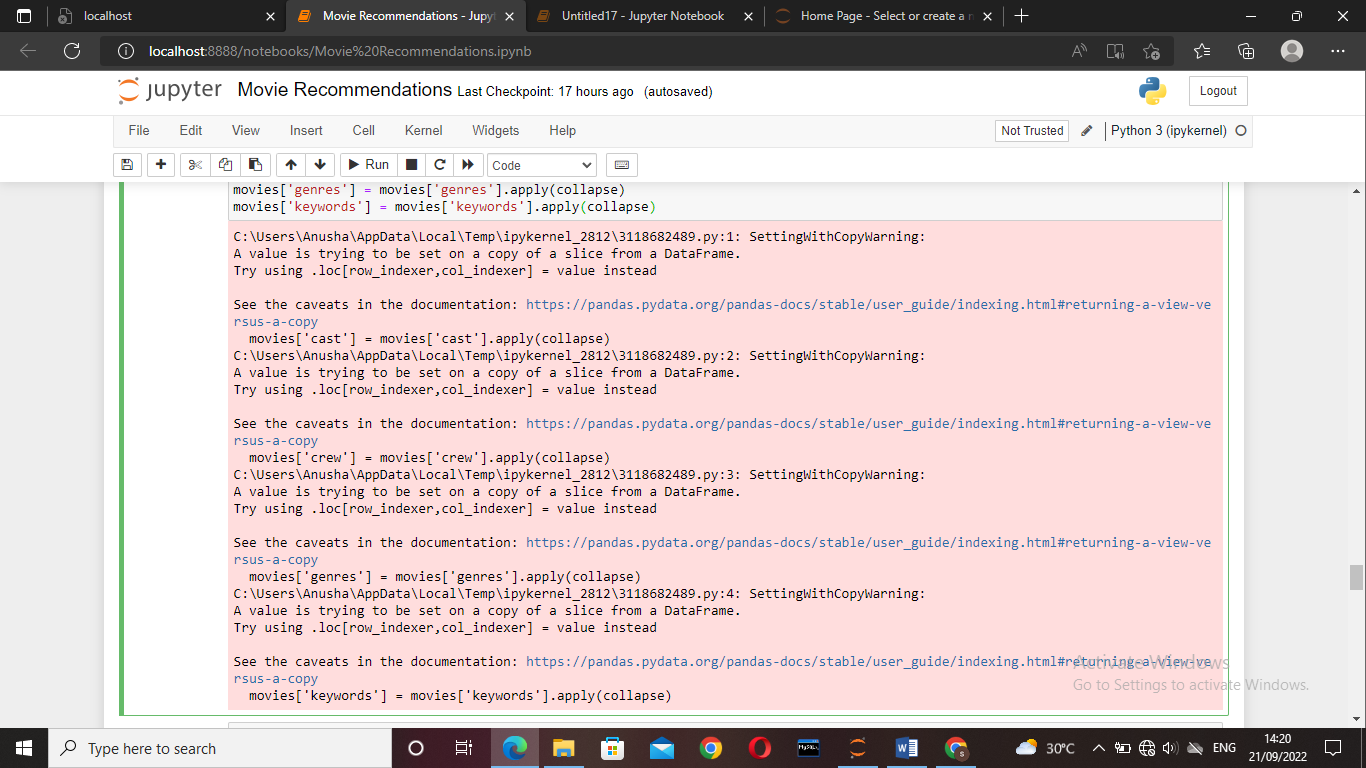
return L1

movies['cast'] = movies['cast'].apply(collapse)

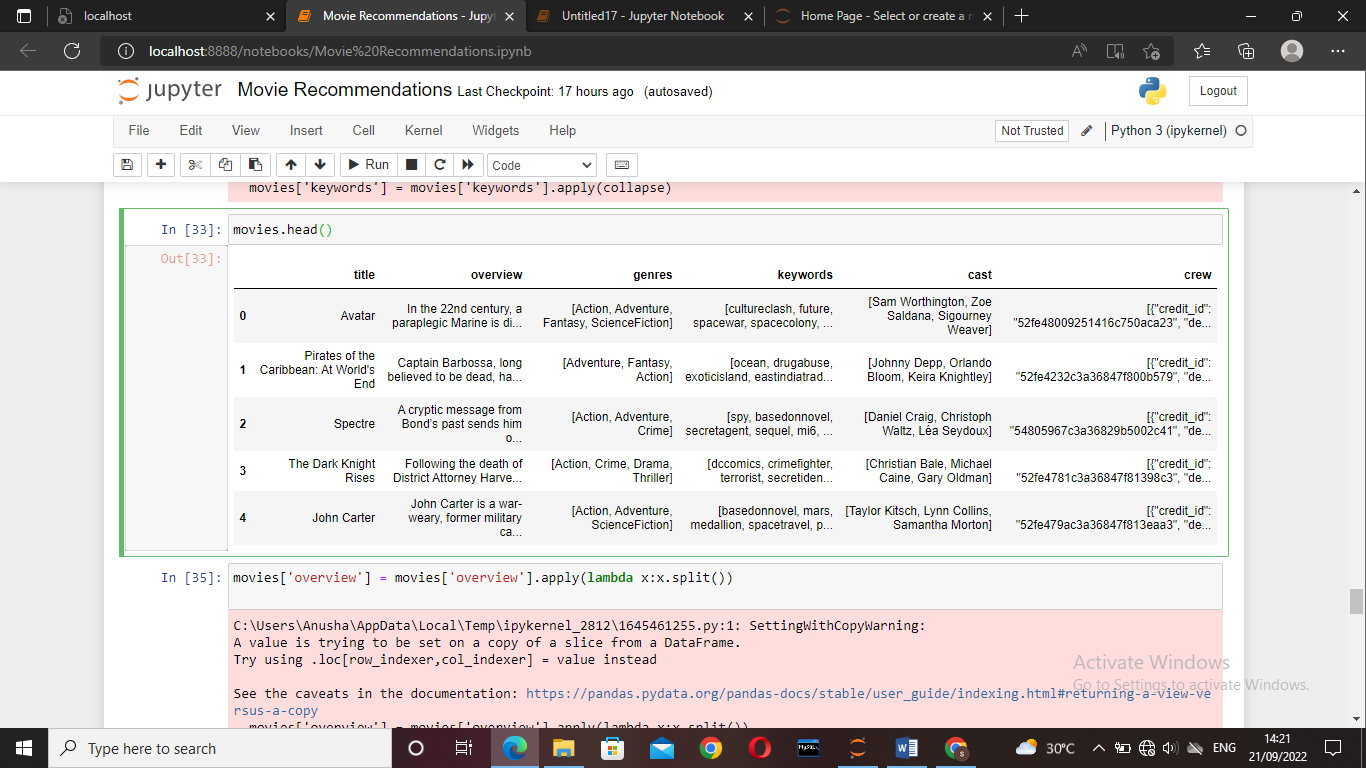
movies['crew'] = movies['crew'].apply(collapse)

movies['genres'] = movies['genres'].apply(collapse)

movies['keywords'] = movies['keywords'].apply(collapse)

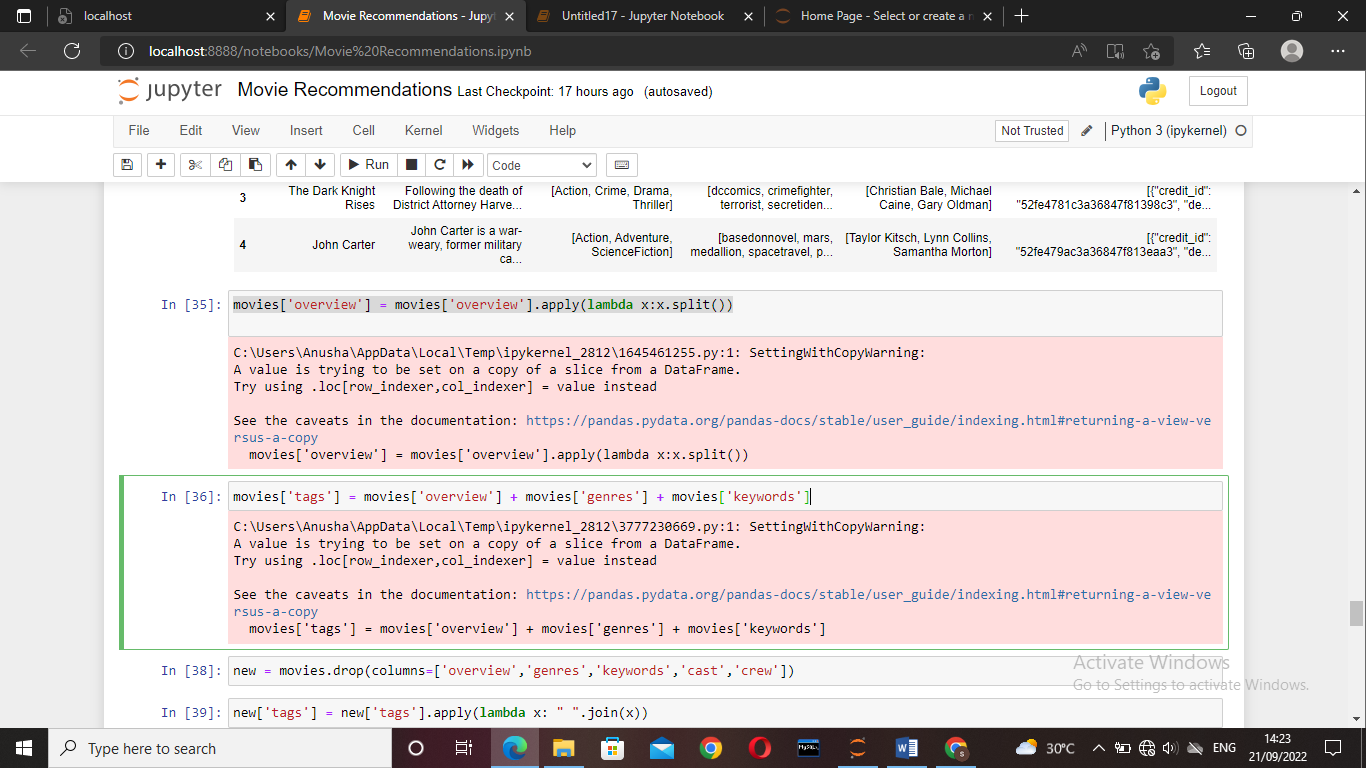


movies.head()

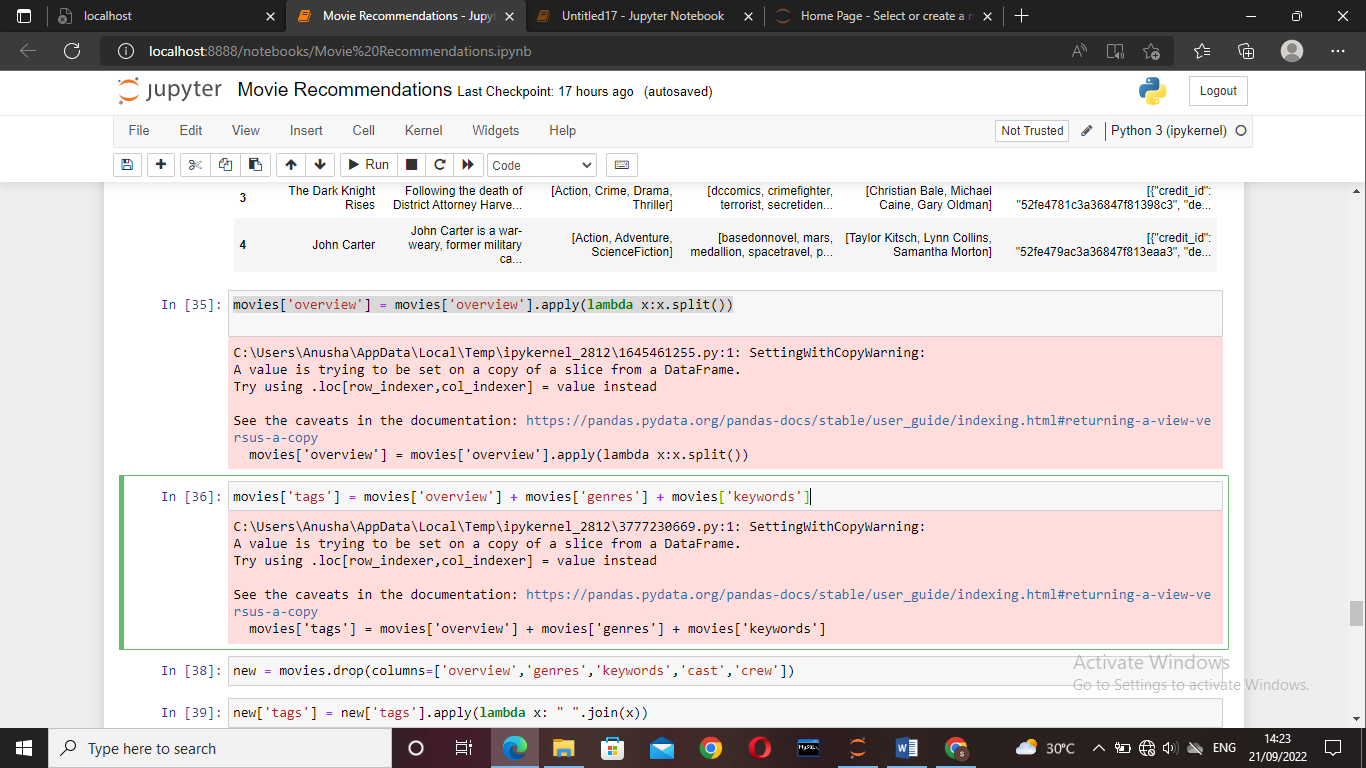


* Cast and crew has different ways to extract as only top three are needed in the cast so lambd function to take only top three values. In crew only director is needed since some people watch as the director directed the movie.

movies['overview'] = movies['overview'].apply(lambda x:x.split())



movies['tags'] = movies['overview'] + movies['genres'] + movies['keywords']



* Transformation is used on the keyword, genre, cast, crew to remove the spaces between the words as it will confuse the system.

new = movies.drop(columns=['overview','genres','keywords','cast','crew'])

new['tags'] = new['tags'].apply(lambda x: " ".join(x))

***VECTORIZATION:***

In this we use bag of words technique.here movies consider as vectors.

* Steps used for vectorization process:

1. Combined tags of the movies.
2. Take the most repeated words more than 5000 words.
3. Calculate frequency of each word.
4. From sklearn countvectoriz class is used it produces “spicy sparse matrix” so we converted into numpy array.
5. Then we apply stimming process by using NLTK which is “natural language processing library”.

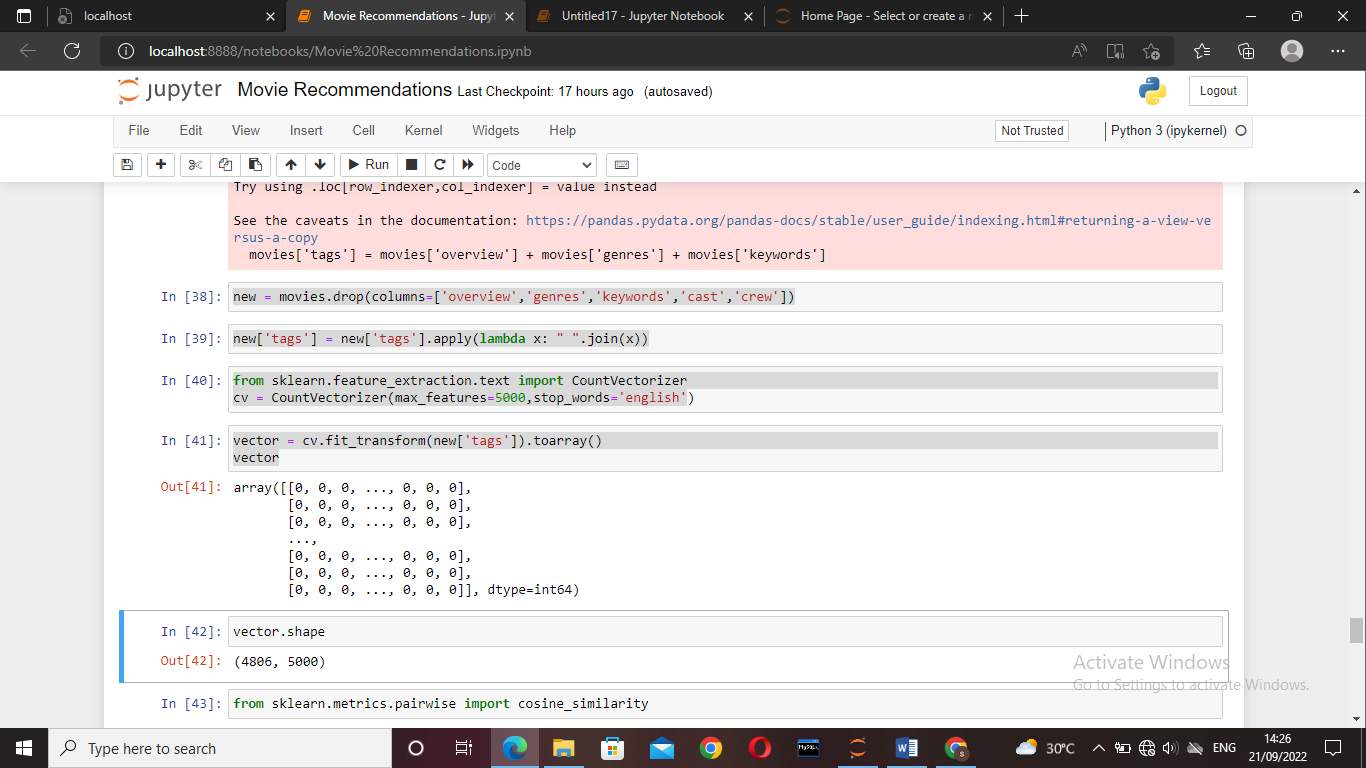
Here we use cosine distance which takes theta to find the similarity between two vectors as euclidain distance not a reliable method for higher dimensions

from sklearn.feature\_extraction.text import CountVectorizer

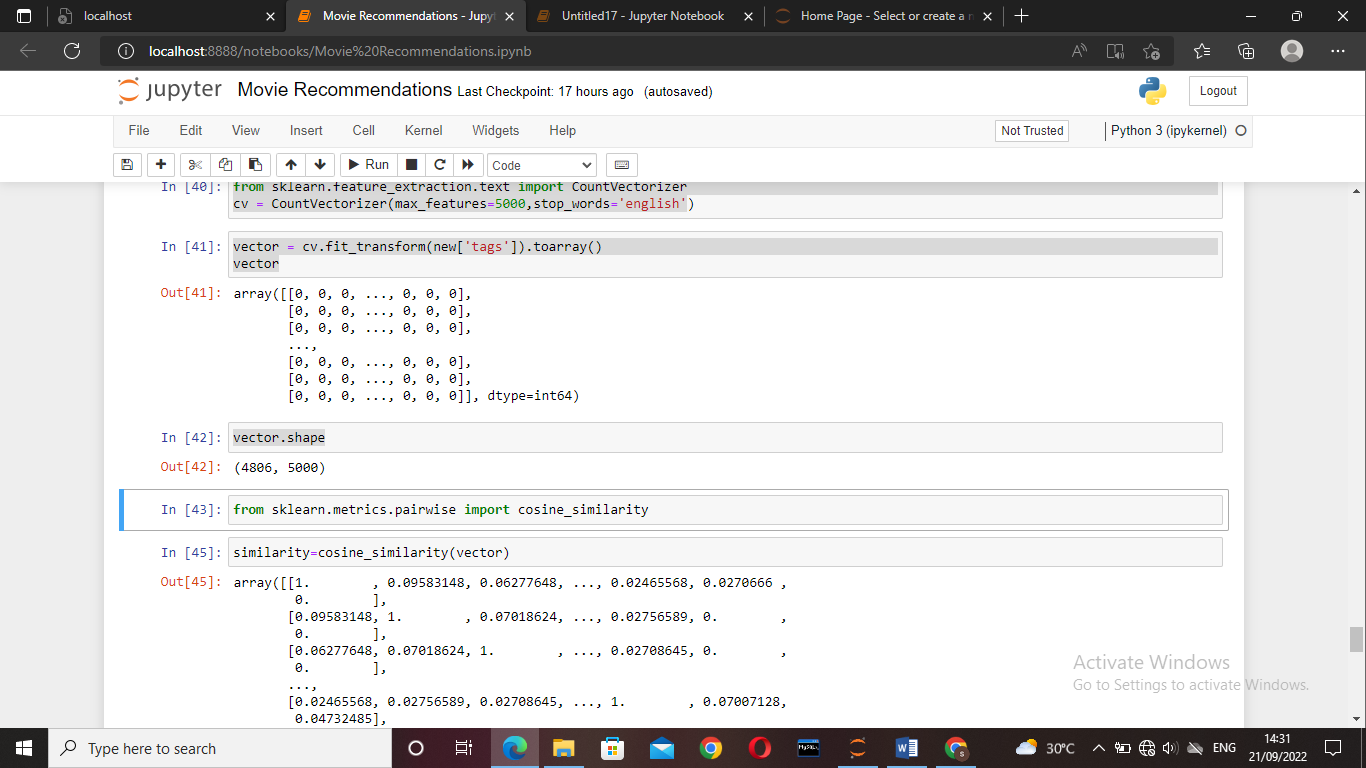
cv = CountVectorizer(max\_features=5000,stop\_words='english'

vector = cv.fit\_transform(new['tags']).toarray()

vector

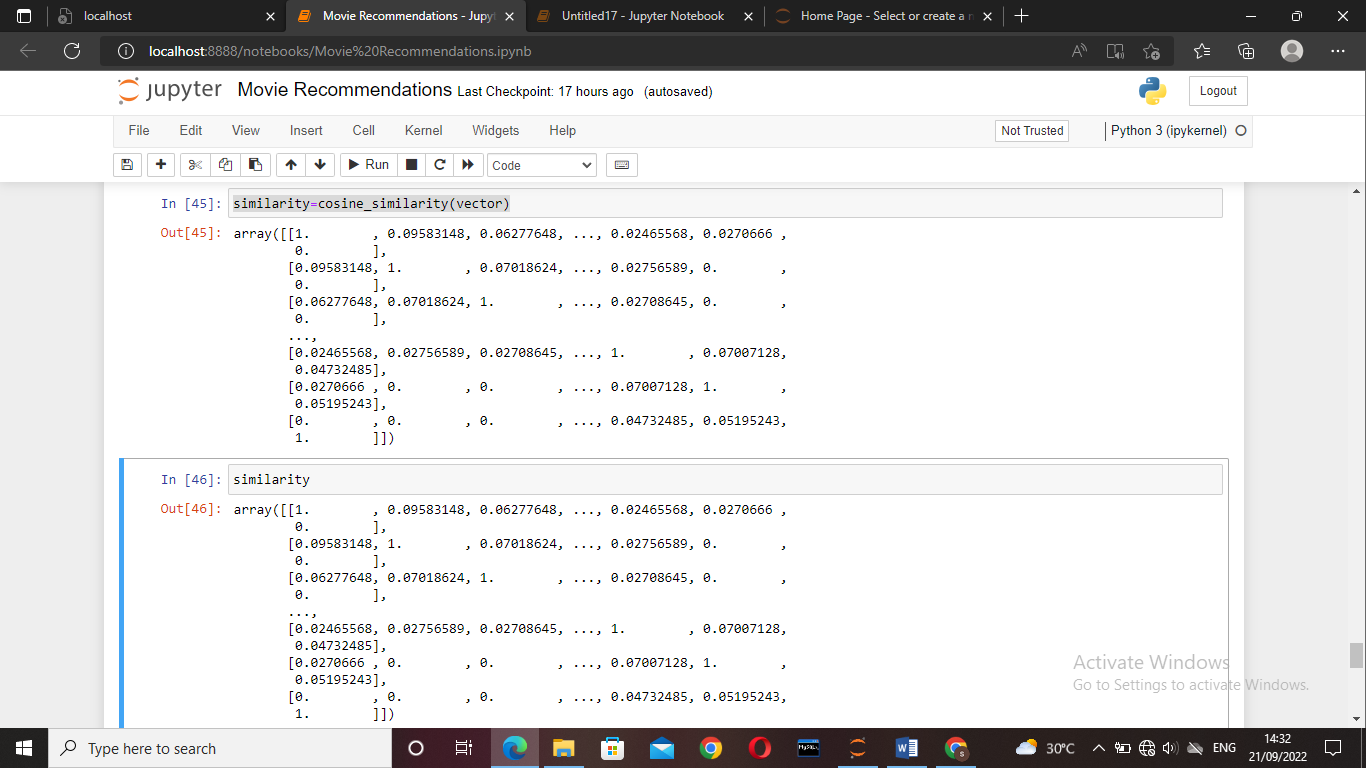
****

vector.shape

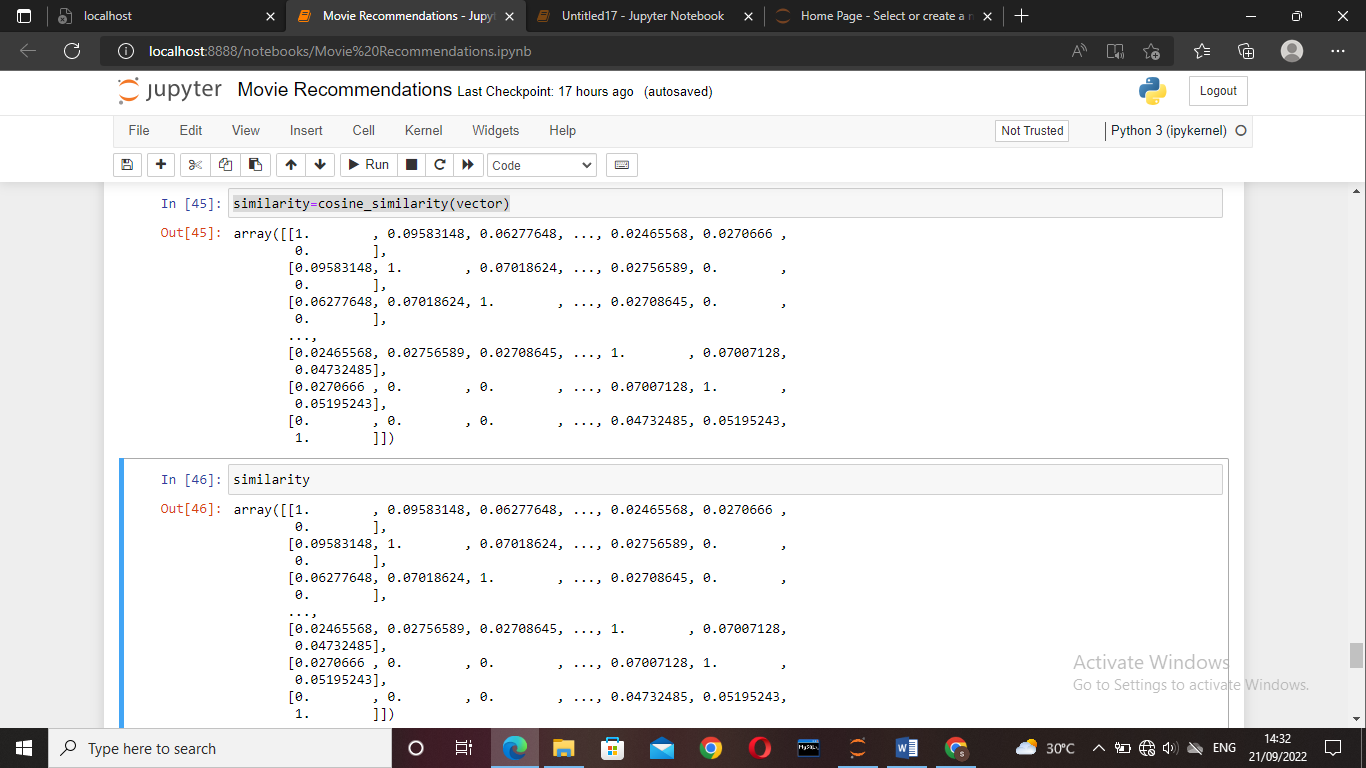


from sklearn.metrics.pairwise import cosine\_similarity

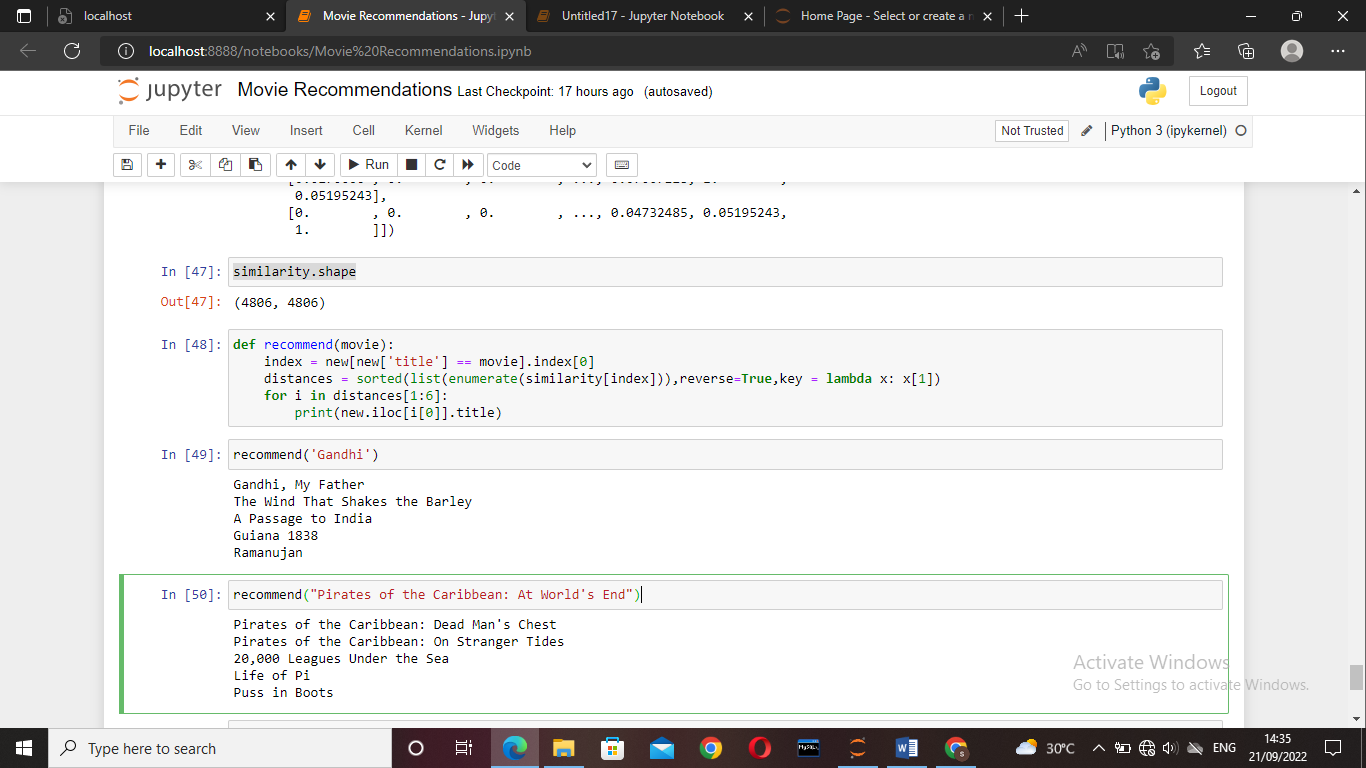
similarity=cosine\_similarity(vector)



Similarity



similarity.shape



def recommend(movie):

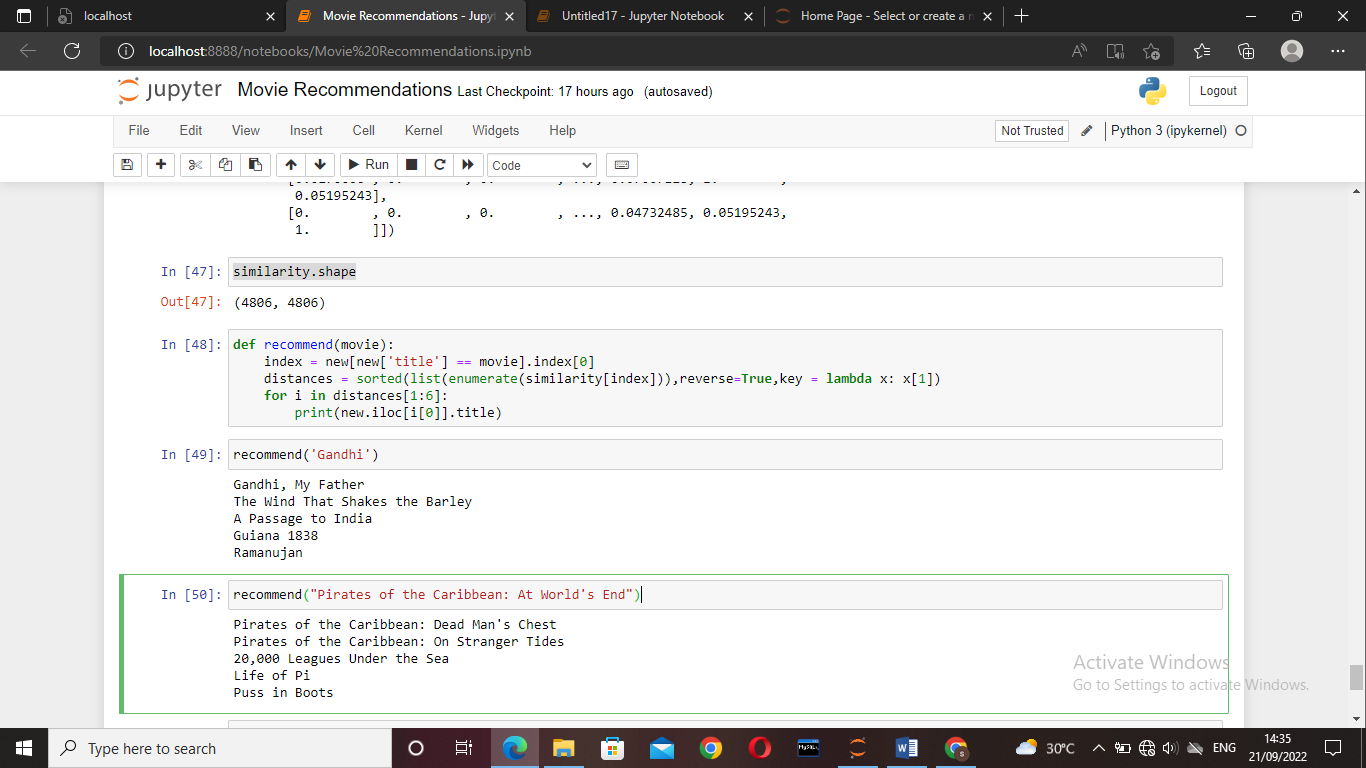
index = new[new['title'] == movie].index[0]

distances = sorted(list(enumerate(similarity[index])),reverse=True,key = lambda x: x[1])

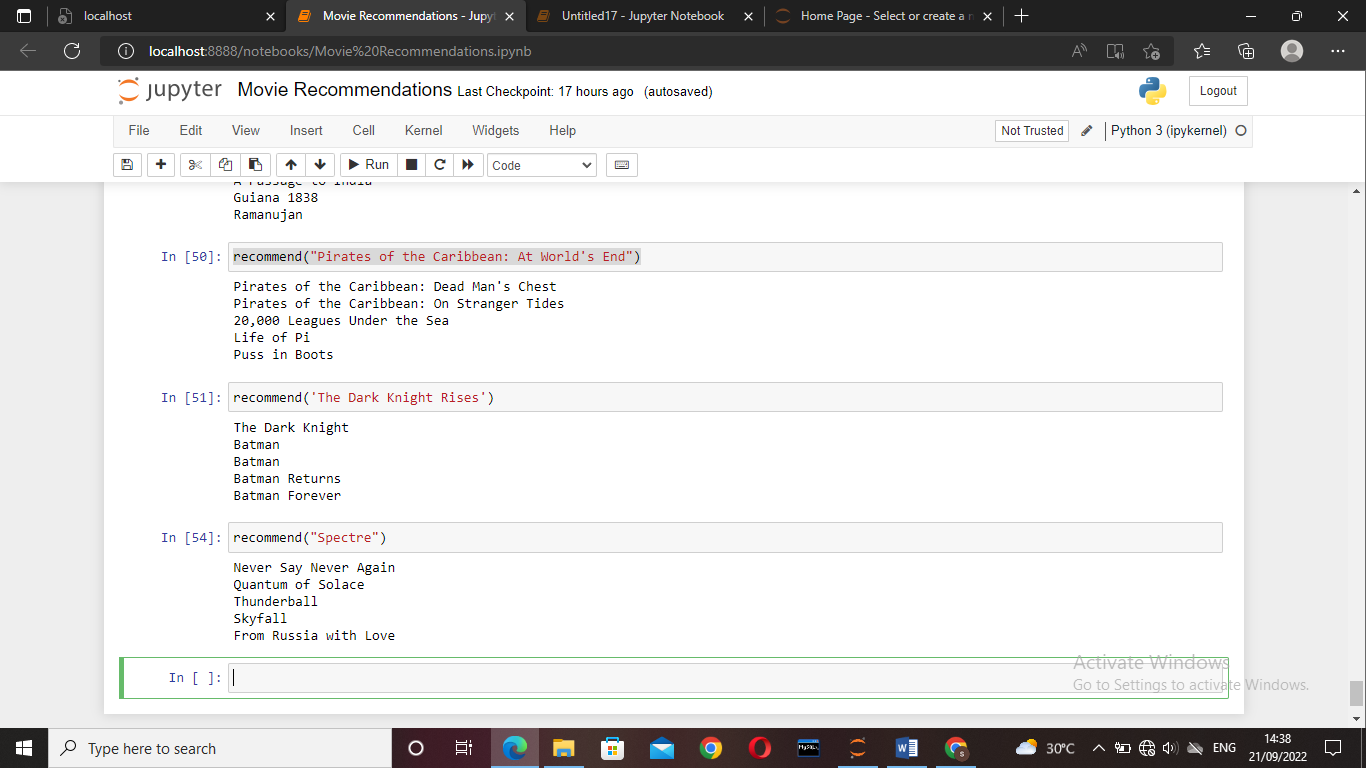
for i in distances[1:6]:

print(new.iloc[i[0]].title)

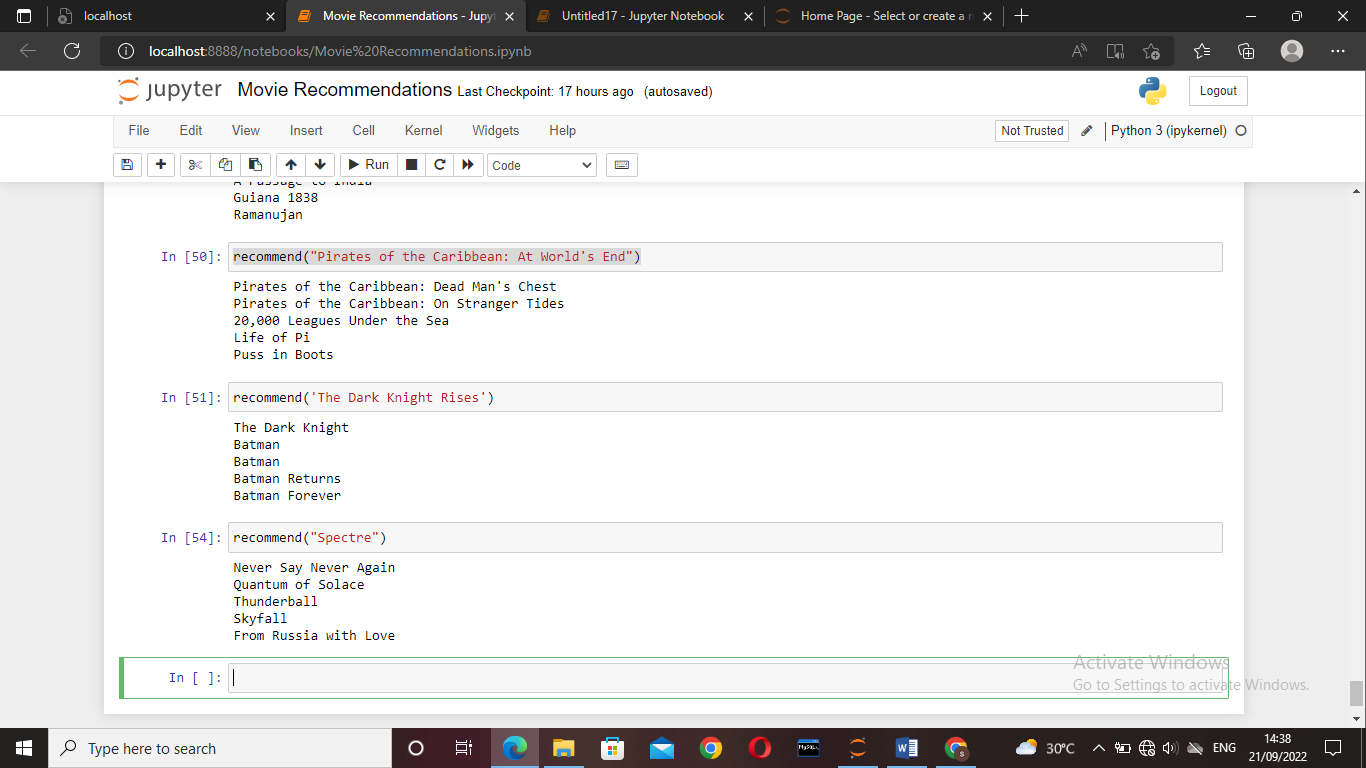
recommend('Gandhi')



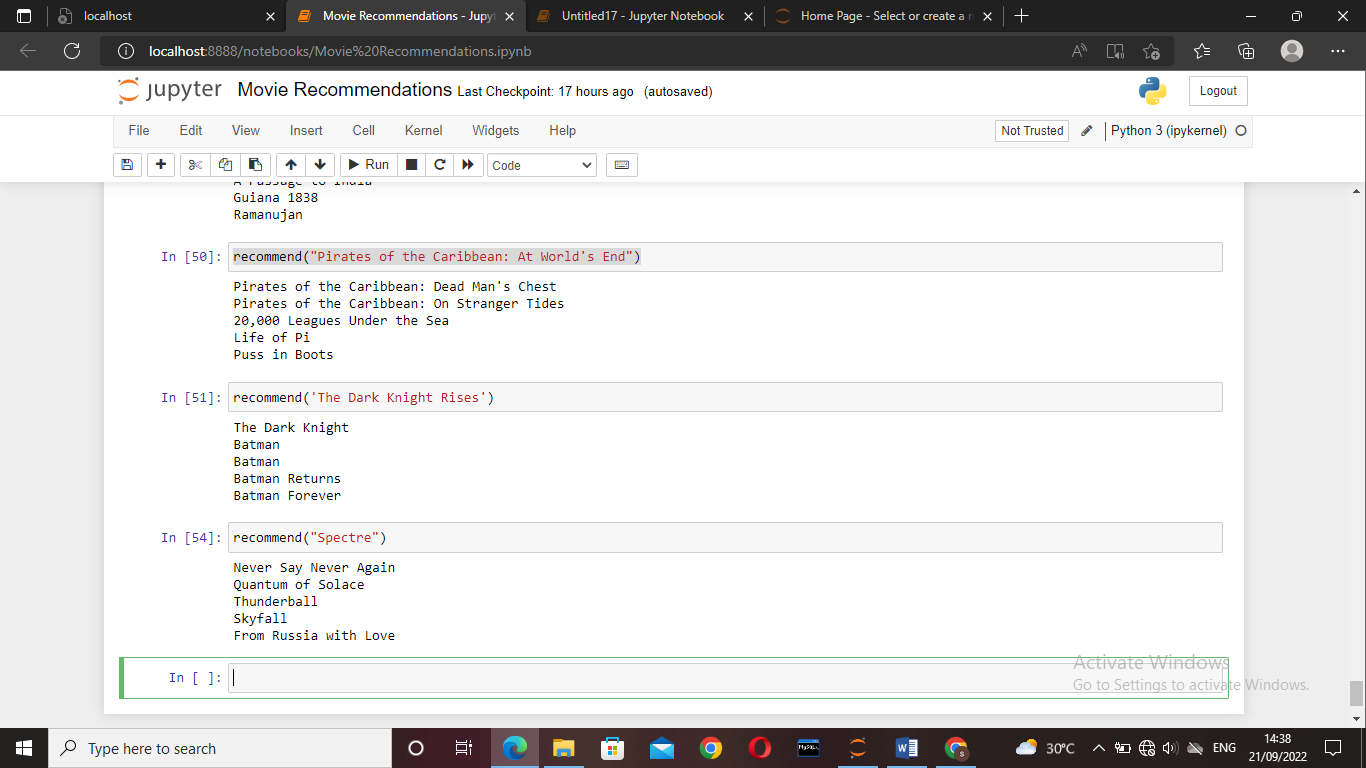
recommend("Pirates of the Caribbean: At World's End")



recommend('The Dark Knight Rises')



recommend("Spectre")



When the cosine similarity is nearer 1 then there are more similar else when it is near zero the dissimilar.

The steps involved in recommending model

* Find the index of the user given movie
* Finding the similarity

1. First five similar movies are need
2. Sort the similarities enumerate is used so that the movie title can be linked.
3. Recommend the top similar movies.

*These are the steps involved in the construction of movie recommendation model and this model can be used for any type of the recommendation system.*

**Yours Sincerely,**

***ID: s180887,***

***Cse-2c,***

***S180887@rguktsklm.ac.in***