# -\*- coding: utf-8 -\*-

"""Created on Thu Jan 24 13:50:03 2019

@author: shaz-

"""

#########################################################################################################################

# Importing Packages

#########################################################################################################################

'''

Importing The Necessary Packages

'''

import json

import re

import requests

import warnings

import numpy as np

import pandas as pd

import mysql.connector

import urllib.request

from scipy import stats

import seaborn as sns

from bs4 import BeautifulSoup

from currency\_converter import CurrencyConverter

from matplotlib import pyplot as plt

import nltk

import unicodedata

import vaderSentiment

from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

from nltk.sentiment.vader import SentimentIntensityAnalyzer as SIA

from sklearn.preprocessing import scale

from sklearn.model\_selection import train\_test\_split

from sklearn import metrics as sm

from sklearn.svm import LinearSVC

from sklearn.feature\_extraction.text import TfidfVectorizer

warnings.filterwarnings('ignore')

sns.set(style="darkgrid", color\_codes=True)

#########################################################################################################################

# Defining Functions

#########################################################################################################################

class ImdbMovies:

model=''

vectorizer=''

mydb=''

'''Loading constructor, so when instance is instantiate it will load our model and as well

as it will create a connection with the database'''

def \_\_init\_\_(self,\*\*kwargs):

self.firstname=kwargs.get('firstname','Firstname Not Provided')

self.lastname=kwargs.get('lastname','LastName Not Provided')

self.mydb=self.DatabaseConnection('root','Sagar$256','imdbmovies')

print("\nPlease wait {}, while we're running the model.....".format(self.firstname))

self.model,self.vectorizer=self.UserReview\_SentimentAnalyzer()

print('''Done!!, you're good to go''')

print("#########################################################################################################################")

print("Welcome! {} {} to our movie search and data analysis program:\n".format(self.firstname.capitalize(),self.lastname.capitalize()))

print("#########################################################################################################################")

'''This is just to provide user freindly string when object is print'''

def \_\_str\_\_(self):

return '''What's going on {} {}, enjoy your movie buddy'''.format(self.firstname.capitalize(),self.lastname.capitalize())

'''Using Vader lexicon function to get the polarity'''

def sentiment\_lexicon(self,review, threshold=0.1):

sid = SIA()

ss = sid.polarity\_scores(review)

agg\_score = ss['compound']

if agg\_score >= threshold:

final\_sentiment = 'Positive'

else:

final\_sentiment = 'Negative'

return final\_sentiment

'''Sentiment analysis based on user review submited'''

def UserReview\_SentimentAnalyzer(self):

self.df=pd.read\_sql("select imdbid,User\_Review,Polarity from movies;",self.mydb)

# User\_Review

self.data = self.df['User\_Review']

self.data=pd.Series.to\_string(self.data) ## converted to string from pandas.Series

# for removing accented characters

self.normal = unicodedata.normalize('NFKD', self.data).encode('ASCII', 'ignore')

# sentiment\_vader\_lexicon:

self.list\_senti=[]

for i in self.df['User\_Review']:

self.list\_senti.append(self.sentiment\_lexicon(i))

self.list\_senti

#creating new column as sentiment which will have 0/1 values

self.df['polarity']=self.list\_senti

# assigning

self.features=self.df.loc[:,'User\_Review']

self.senti=self.df.loc[:,'polarity']

# Using TFIDF vectorizer

self.vectorizer = TfidfVectorizer(min\_df= 3, stop\_words="english", sublinear\_tf=True, norm='l2', ngram\_range=(1, 2))

self.final\_features = self.vectorizer.fit\_transform(self.features).toarray()

self.features\_train,self.features\_test,self.senti\_train,self.senti\_test=train\_test\_split(self.final\_features,self.senti,test\_size=0.2,random\_state=23)

# SVC model to get acc & class table

self.lsvm = LinearSVC()

self.l = self.lsvm.fit(self.features\_train,self.senti\_train)

return self.l,self.vectorizer

'''Predictor function that will help to analyse user review and provide the polarity'''

def Predict\_Sentiment(self,model,vectorizer,User\_Review):

# l=UserReview\_SentimentAnalyzer()

self.y = self.vectorizer.transform([self.User\_Review]).toarray()

self.z=model.predict(self.y)

return self.z

'''Displaying intial menu where user can select an option either to search the movie or analyse the movie '''

def displayMenu(self):

print("\nMenu:\n[S]earch Movie,[A]nalyze the data,[Q]uit:\n")

print("---------------------------------------------------------------------------")

Choice=''

flag=0

options={'s':'search','a':'analyse','q':'quit'}

try:

Choice=input("Please enter your option below:\n").lower()

if Choice=='':

flag=1

return Choice,flag

elif Choice not in options.keys():

flag=2

return Choice,flag

else:

return Choice,flag

except ValueError:

print("\nInvalid input...please enter S,A,Q from choices provided:\n")

self.displayMenu()

'''Budget and Review need to be extracted from IMDB website '''

def Extract\_Budget\_UserReview(self,imdbID):

c = CurrencyConverter()

CurrencyDict = {'$': 'USD', '£': 'GBP', '¥': 'JPY', '€': 'EUR', '₹': 'INR'}

url = 'http://www.imdb.com/title/{}/?ref\_=fn\_al\_nm\_1a'.format(imdbID)

data = requests.get(url)

soup = BeautifulSoup(data.text, 'html.parser')

Budget = 0

userReview = ""

#Extracting the user Review of the movie

movie = soup.findAll('div', {'class': 'user-comments'})

for res in movie:

userReview = res.span.strong.text

if userReview is None:

userReview='N/A'

#Extracting the Budget of the movie

for h4 in soup.find\_all('h4'):

if "Budget:" in h4:

Budget = h4.next\_sibling

match = re.search(r'([\D]+)([\d,]+)', Budget)

output = (match.group(1).replace('\xa0', ''),

match.group(2).replace(',', ''))

if len(output[0]) == 1:

Budget = round(

(c.convert(output[1], CurrencyDict[output[0]], 'USD')/1000000), 2)

elif len(output[0]) == 3 and output[0] == 'XAF':

Budget = round((float(output[1])\*0.00174637)/1000000, 2)

elif len(output[0]) == 3 and output[0] == 'FRF':

Budget = round((float(output[1])\*0.17)/1000000, 2)

elif len(output[0]) == 3 and output[0] == 'IRR':

Budget = round((float(output[1])\*0.0000237954)/1000000, 2)

elif len(output[0]) == 3 and output[0] == 'PKR':

Budget = round((float(output[1])\*0.007225614)/1000000, 2)

elif len(output[0]) == 3 and output[0] == 'NPR':

Budget = round((float(output[1])\*87.0521)/1000000, 2)

elif len(output[0]) == 3 and output[0] != 'FRF':

Budget = round(

c.convert(output[1], output[0], 'USD')/1000000, 2)

return Budget,userReview

'''Extracting movie details from API'''

def getMovieData(self,Movietitle):

try:

url = "http://www.omdbapi.com/?t={}&apikey=5ddb11dd".format(Movietitle)

print("Retrieving the data of \"{}\" now…".format(Movietitle))

api\_request = requests.get(url)

source = json.loads(api\_request.content)

except requests.RequestException as e:

print(f"ERROR: {e.reason}")

return source

'''Establishing the database connection'''

def DatabaseConnection(self,user, passwd, database):

mydb=''

try:

mydb = mysql.connector.connect(host='localhost',

user=user,

passwd=passwd,

db=database)

except:

print("""The login credentials you entered are not valid for

the database you indicated. Please check your login details and try

again.""")

return mydb

'''This function will sepearte each word from genre and stack it in long format'''

def explode(self,df, lst\_cols, fill\_value=''):

# make sure `lst\_cols` is a list

if lst\_cols and not isinstance(lst\_cols, list):

lst\_cols = [lst\_cols]

# all columns except `lst\_cols`

idx\_cols = df.columns.difference(lst\_cols)

# calculate lengths of lists

lens = df[lst\_cols[0]].str.len()

if (lens > 0).all():

# ALL lists in cells aren't empty

return pd.DataFrame({

col: np.repeat(df[col].values, lens)

for col in idx\_cols

}).assign(\*\*{col: np.concatenate(df[col].values) for col in lst\_cols}) \

.loc[:, df.columns]

else:

# at least one list in cells is empty

return pd.DataFrame({

col: np.repeat(df[col].values, lens)

for col in idx\_cols

}).assign(\*\*{col: np.concatenate(df[col].values) for col in lst\_cols}) \

.append(df.loc[lens == 0, idx\_cols]).fillna(fill\_value) \

.loc[:, df.columns]

'''This Function will put the data extracted from API and from webscraping into movie database '''

def DataIntoDatabase(self,MovieData, mydb, mycursor):

self.budget,self.User\_Review = self.Extract\_Budget\_UserReview(MovieData['imdbID'])

self.pred=self.Predict\_Sentiment(self.model,self.vectorizer,self.User\_Review)

self.polarity=''.join(self.pred)

if MovieData['Metascore'] == 'N/A':

metascore = 0

else:

metascore = (float((MovieData['Metascore']))/10)

if MovieData['imdbRating']=='N/A':

imdb\_rating=0

else:

imdb\_rating = float(MovieData['imdbRating'])

if MovieData['Released']=='N/A':

release\_year=0

else:

release\_year=int(MovieData['Released'].split(' ')[2])

if MovieData['Poster']=="N/A":

image\_url='Image Not Available'

else:

image\_url=MovieData['Poster']

sql = """INSERT INTO movies(IMDBID, Title, Genre, Year, URL, Audience\_Rating, Critic\_Rating, Budget\_In\_Millions, User\_Review,Polarity)

VALUES (%s, %s,%s, %s,%s,%s,%s,%s,%s,%s)

ON DUPLICATE KEY UPDATE

Audience\_Rating=values(Audience\_Rating),

Critic\_Rating=values(Critic\_Rating),

Budget\_In\_Millions=values(Budget\_In\_Millions),

User\_Review=values(User\_Review),

Polarity=values(Polarity);"""

val=[(MovieData['imdbID'],MovieData['Title'],

MovieData['Genre'],release\_year,image\_url,

imdb\_rating,metascore,self.budget,self.User\_Review,self.polarity)]

mycursor.executemany(sql, val)

mydb.commit()

'''This function will fetch the data from database from the title provided by the user'''

def getDataFromDatabase(self,UserInputTitle):

mydb=self.mydb

self.mycursor=mydb.cursor()

self.mycursor.execute("""select title,genre,year,audience\_rating,critic\_rating,polarity

from movies where title like %s limit 1""",("%" + self.UserInputTitle + "%",))

self.myresult = self.mycursor.fetchall()

return self.myresult

'''

This is use to display info about the movie title provided by the user,

at the same time if the movie title doesn't exist the it will make an entry into the database

and then it will fetch the data from database to display info.

'''

def DisplayMovieInfo(self,UserInputTitle):

mydb=self.mydb

mycursor=mydb.cursor()

try:

myresult=self.getDataFromDatabase(self.UserInputTitle)

if not myresult:

MovieData=self.getMovieData(self.UserInputTitle)

if MovieData['Response']=='False':

print("Sorry!!! The Movie Doesn't Exist.....:(")

else:

self.DataIntoDatabase(MovieData,mydb,mycursor)

myresult=self.getDataFromDatabase(self.UserInputTitle)

if myresult[0][5]=='Positive':

res="Good Choice! & you can enjoy this with your buddy. :)"

else:

res="Well, you've decide so enjoy this with your popcorn. :)."

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print("Title: {}".format(myresult[0][0]))

print("Genre: {}".format(myresult[0][1]))

print("Year: {}".format(myresult[0][2]))

print("Audience Rating: {}".format(myresult[0][3]))

print("Critic Rating: {}".format(myresult[0][4]))

print("What's My Sugesstion: {}".format(res))

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

else:

if myresult[0][5]=='Positive':

res="Your can enjoy this with your buddy!"

else:

res="Well you've decide so enjoy with your popcorn."

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print("Title: {}".format(myresult[0][0]))

print("Genre: {}".format(myresult[0][1]))

print("Year: {}".format(myresult[0][2]))

print("Audience Rating: {}".format(myresult[0][3]))

print("Critic Rating: {}".format(myresult[0][4]))

print("What's My Sugesstion: {}".format(res))

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

except:

print('''Sorry its doesnt exist...please try once again''')

'''This Function will fetch the data by year'''

def getDataByYear(self,FirstRange,SecondRange):

self.movieData=pd.read\_sql("""select title,genre,year,audience\_rating,critic\_rating,budget\_in\_millions

from movies

where url!='N/A' and

genre!='N/A'and

year!=0 and

Audience\_rating!=0 and

critic\_rating!=0 and

budget\_in\_millions!=0 and

user\_review <> ''and polarity <> '' and Year BETWEEN {} and {};""".format(FirstRange,SecondRange),self.mydb)

return self.movieData

'''Getting an input title from the user'''

def getChoiceMovie(self):

try:

self.UserInputTitle=input("Please enter the title of the movie:\n")

if self.UserInputTitle=='':

print("No Input Provided.")

else:

self.DisplayMovieInfo(self.UserInputTitle)

except ValueError:

print("\nSome error occured.....please check the input provided")

'''This fucntion will fetch the data from the database & process it while formatting the data in long format'''

def DataAnalysis(self):

self.movieData=''

self.year=pd.read\_sql('''select max(year) as Max\_Year, min(year) as Min\_Year

from movies where url!='N/A' and

genre!='N/A'and year!=0 and

Audience\_rating!=0 and

critic\_rating!=0 and

budget\_in\_millions!=0 and

user\_review <> '' and

Polarity <>'' ''',self.mydb)

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print("Data is avialable from {} to {}.\nEnter the range so as to bring you the analysis".format(self.year.Min\_Year[0],self.year.Max\_Year[0]))

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

try:

self.FirstRange=input("Please Enter The Range 1:\n")

self.SecondRange=input("Please Enter The Range 2:\n")

if self.FirstRange=='' or self.SecondRange=='':

print("No input was provided.\n")

self.DataAnalysis()

elif self.FirstRange > self.SecondRange:

print('\nProvided range is invalid...since the first range cannot be greater than the second.\n')

self.DataAnalysis()

elif len(self.FirstRange)!=4 or len(self.SecondRange)!=4:

print("Provided range is invalid...\n")

self.DataAnalysis()

else:

self.movieData=self.getDataByYear(int(self.FirstRange),int(self.SecondRange))

self.movieData.genre = self.movieData.genre.str.split(',')

self.movieData = self.explode(self.movieData,['genre'])

self.movieData.genre=self.movieData.genre.str.strip(' ')

self.movieData.genre=self.movieData.genre.astype('category')

self.movieData.audience\_rating=self.movieData.audience\_rating\*10

self.movieData.critic\_rating=self.movieData.critic\_rating\*10

except ValueError:

print("Please provide correct input...since the entered value is not a number.\n")

self.DataAnalysis()

return self.movieData

def OptionChoice(self):

print("""Please Choose option below:

1. Display Top 10 Rated Movies

2. Display Top 10 High Budget Movies

3. Display Critic Vs Audience Rating

4. Distribution of Critic or Audience Rating

5. Display Stack Distribution of Budget

6. Display Boxplot

7. Display Barplot\n

""")

def OptionChoiceDist(self):

self.optionDist={1:'Critic Rating Distribution',2:'Audience Rating Distribution'}

print("""Please Enter:\n[1] Critic Rating Distribution\n[2] Audience Rating Distribution\n""")

try:

self.Dist=int(input(">"))

if self.Dist not in self.optionDist.keys():

print("Sorry please enter your choice from the option below\n")

self.OptionChoiceDist()

elif self.Dist==1:

self.DisplayHistogram(self.movieData,'critic rating')

elif self.Dist==2:

self.DisplayHistogram(self.movieData,'audience rating')

except ValueError:

print("Invalid input provided.")

self.OptionChoiceDist()

def OptionChoiceBox(self):

self.optionbox={1:'Critic Rating Boxplot',2:'Audience Rating Boxplot'}

print("""Please Enter:\n[1] Display boxplot for critic rating by genre\n[2] Display boxplot for audience rating by genre\n""")

try:

self.box=int(input(">"))

if self.box not in self.optionbox.keys():

print("Sorry please enter your choice from the option below\n")

self.OptionChoiceBox()

elif self.box==1:

self.DisplayBoxplot(self.movieData,'genre','critic rating')

elif self.box==2:

self.DisplayBoxplot(self.movieData,'genre','audience rating')

except ValueError:

print("Invalid input provided.")

self.OptionChoiceBox()

def OptionChoiceBar(self):

self.optionbar={1:'Genre bar plot',2:'Year bar plot'}

print("""Please Enter:\n[1] Display barplot to display Data by Genre\n[2] Display barplot to display Data by year:\n""")

try:

self.bar=int(input(">"))

if self.bar not in self.optionbar.keys():

print("Sorry please enter your choice from the option below\n")

self.OptionChoiceBar()

elif self.bar==1:

self.catPlot(self.movieData,'genre')

elif self.bar==2:

self.catPlot(self.movieData,'year')

except ValueError:

print("Invalid input provided.")

self.OptionChoiceBar()

def DisplayCricticAudienceRating(self,movieData):

#Joint Plot Critic Rating Vs Audience Rating

sns.set(style='whitegrid')

sns.jointplot(data=self.movieData,x='critic\_rating',y='audience\_rating')

j = sns.JointGrid(data=self.movieData,x='critic\_rating',y='audience\_rating')

j = j.plot\_joint(plt.scatter,color="g", s=40, edgecolor="black")

j = j.plot\_marginals(sns.distplot, kde=False,)

j = j.annotate(stats.pearsonr,loc="upper left")

j.set\_axis\_labels('Critic Ratings','Audience Rating')

plt.show()

# Histogram

def DisplayHistogram(self,movieData,column):

column=column.title()

LabelDictCol = {'Critic Rating':'critic\_rating','Audience Rating':'audience\_rating','Budget In Millions':'budget\_in\_millions'}

sns.set(style = 'whitegrid')

fig,ax=plt.subplots()

fig.set\_size\_inches(11.7,8.27)

plt.hist(movieData[LabelDictCol[column]],bins=15,color='black')

plt.title("{} Distribution".format(column),fontsize=20)

plt.ylabel("Frequency",fontsize=15)

plt.xlabel("{} (%)".format(column),fontsize=15)

plt.show()

# Stack distribution

def DisplayStackedHistogram(self,movie):

list1=[]

GenreLabels=[]

for gen in movie.genre.cat.categories:

list1.append(movie[movie.genre==gen].budget\_in\_millions)

GenreLabels.append(gen)

sns.set(style='whitegrid')

fig,ax=plt.subplots()

fig.set\_size\_inches(11.7,8.27)

plt.hist(list1,bins=30,stacked=True,rwidth=1,label=GenreLabels)

plt.title("Movie Budget Distribution",fontsize=20)

plt.ylabel("Number of Movies",fontsize=15)

plt.xlabel("Budget$$$",fontsize=15)

plt.legend(frameon=True,fancybox=True,prop={'size':10},framealpha=1)

plt.show()

# how critic rating is dtributted accross different genre

def DisplayBoxplot(self,data,column1,column2):

column1=column1.title()

column2=column2.title()

LabelDictCol = {'Critic Rating':'critic\_rating','Audience Rating':'audience\_rating','Budget In Millions':'budget\_in\_millions','Genre':'genre','Year':'year'}

fig,ax=plt.subplots()

fig.set\_size\_inches(11.7,8.27)

sns.boxplot(data=data,x=LabelDictCol[column1],y=LabelDictCol[column2],palette='vlag',whis="range")

ax.yaxis.grid(True)

ax.xaxis.grid(True)

plt.title('{} Vs {} Boxplot'.format(column1,column2),fontsize=20)

plt.xlabel('{}'.format(column1),fontsize=15)

plt.ylabel('{}'.format(column2),fontsize=15)

plt.xticks(rotation=30)

sns.despine(trim=True, left=True)

plt.show()

'''Function is use to display barplot for number of movie by genre or year'''

def catPlot(self,data,column):

column=column.title()

LabelDictCol = {'Critic Rating':'critic\_rating','Audience Rating':'audience\_rating','Budget In Millions':'budget\_in\_millions','Genre':'genre','Year':'year'}

sns.catplot(y=LabelDictCol[column], kind="count", palette="ch:.25", data=data)

plt.title('Barplot For {}'.format(column.capitalize()),fontsize=20)

plt.ylabel('{}'.format(column.capitalize()),fontsize=15)

plt.xlabel('')

plt.show()

'''Display top 10 movie w.r.t genre or year'''

def getTop10(self,data):

p=sns.factorplot(aspect=1.5,y='title',x='audience\_rating',data=data.sort\_values(['audience\_rating','critic\_rating'],ascending=False).drop(['genre'],axis=1).drop\_duplicates().head(10),palette="ch:.25",kind='bar')

p.set(xlim=(10,100))

sns.set\_style("ticks",{"xtick.major.size":8,"ytick.major.size":8})

plt.title('Top 10 Rated Movies',fontsize=20)

plt.ylabel('Title',fontsize=15)

plt.xlabel('Audience Rating',fontsize=15)

sns.despine(trim=True,left=True)

plt.show()

'''Function will display top 10 movie w.r.t budget'''

def getTop10HighBudgetMovie(self,data):

sns.factorplot(aspect=1.5,y='title',x='budget\_in\_millions',data=data.sort\_values(['budget\_in\_millions'],ascending=False).drop(['genre'],axis=1).drop\_duplicates().head(10),palette="ch:.25",kind='bar')

sns.set\_style("ticks",{"xtick.major.size":8,"ytick.major.size":8})

plt.title('Top 10 High Budget Movies',fontsize=20)

plt.ylabel('Title',fontsize=15)

plt.xlabel('Budget In Millions',fontsize=15)

sns.despine(trim=True,left=True)

plt.show()

'''This is function will be call after the object is created and its link with mulitple functions from above'''

def DisplayTheDetails(self):

self.options={'s':'search','a':'analyse','q':'quit'}

while True:

Choice,flag = self.displayMenu()

if flag==1 and Choice=='':

print("Please select the option from the menu:\n")

break

self.displayMenu()

elif flag==2 and Choice not in self.options.keys():

print("Please select the option from the menu:\n")

break

self.displayMenu()

elif flag==0 and Choice in self.options.keys():

if Choice == 's':

self.getChoiceMovie()

break

elif Choice == 'a':

self.optionAnalyze={1:'Display Top 10 Rated Movies',2:'Display Top 10 High Budget Movies',3: 'Display Critic Vs Audience',

4:'Distribution of Critic Vs Audience',5:'Display Stack Distribution of Budget',6:'Display Boxplot',7:'Display Barplot',

8:'Display Dashboard'}

self.movieData=self.DataAnalysis()

try:

self.OptionChoice()

choice=int(input("Please enter the option below:\n"))

if choice not in self.optionAnalyze.keys():

print("Sorry please enter your choice from the option below ")

break

self.OptionChoice()

if choice == 1:

self.getTop10(self.movieData)

elif choice==2:

self.getTop10HighBudgetMovie(self.movieData)

elif choice==3:

self.DisplayCricticAudienceRating(self.movieData)

elif choice==4:

self.OptionChoiceDist()

elif choice==5 :

self.DisplayStackedHistogram(self.movieData)

elif choice==6:

self.OptionChoiceBox()

elif choice==7:

self.OptionChoiceBar()

except ValueError:

print("Sorry! please enter a number.")

self.DisplayTheDetails()

break

elif Choice == 'q':

break

check\_again=input("Do you want to check again? Y/n:\n")

if check\_again.lower() != 'n':

self.DisplayTheDetails()

else:

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Thanks for your participation, GoodBye!!!")

#########################################################################################################################

# The Main Part for Displaying Movie info to the user

#########################################################################################################################

def MainFunction():

try:

FirstName = input("Please Enter Your First Name:\n")

LastName = input("Please Enter Your Last Name:\n")

if FirstName=='' or LastName=='':

print("Input cannot be blank...")

else:

User = ImdbMovies(firstname=FirstName, lastname=LastName)

User.DisplayTheDetails()

except:

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Please provide a valid input")

MainFunction()

MainFunction()