Appendix:

Code:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

"""

To run this script we need rt\_data.txt, actor\_rating.csv, director\_rating.csv files in the same directory

"""

import pandas as pd

import numpy as np

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.feature\_extraction.text import CountVectorizer

import csv

from sklearn import model\_selection

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import accuracy\_score

from sklearn.neighbors import KNeighborsClassifier

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn.naive\_bayes import BernoulliNB

from sklearn.ensemble import RandomForestClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn.svm import SVC

from sklearn.svm import SVR

from sklearn.model\_selection import KFold

from sklearn.tree import DecisionTreeRegressor

from sklearn import linear\_model

import warnings

warnings.filterwarnings('ignore')

movies = pd.read\_csv('rt\_data.txt',sep='\t',header=0)

movies.shape #to see the shape of input file

list(movies)

movies

def convertPercentagetoNumber(x):

x= str(x)

x = x.replace("%", "")

return float(x)

def convertRuntimetoNumber(x):

x= str(x).strip()

x = x.split(" ")[0]

return float(x)

def convertCurrencytoNumber(x):

x= str(x)

x = x.replace("$", "").replace(",", "").replace(" ", "")

return float(x)

movies.describe()

movies.head(10)

#Handling outliers before updating missing Values

#==============================================================================

print("Cleaning the data ....")

try:

movies['audience\_score'] = movies['audience\_score'].replace('No Score Yet', '0%')

movies['audience\_score'] = movies['audience\_score'].apply(convertPercentagetoNumber)

movies['audience\_score'].fillna(movies['audience\_score'].median(axis=0),inplace=True)

print("Completed : cleaning of AudienceScore")

#print(movies['audience\_score'])

except Exception as e:

print('Exception in cleaning Audience\_Score column')

print('Exception is ::',e)

try:

movies['critic\_score'].replace('No Score Yet', '0%')

movies['critic\_score'] = movies['critic\_score'].apply(convertPercentagetoNumber)

movies['critic\_score'].fillna(movies['critic\_score'].median(axis=0), inplace=True)

print("Completed : cleaning of CriticScore")

# print(movies['critic\_score'])

except Exception as e:

print('Exception in cleaning Critic\_Score column')

print('Exception is ::',e)

try:

movies['Runtime'] = movies['Runtime'].replace('NONE', '0')

movies['Runtime'] = movies['Runtime'].apply(convertRuntimetoNumber)

#print(movies['Runtime'])

print("Completed : cleaning of cleanRunTime")

except Exception as e:

print('Exception in cleaning Runtime column')

print('Exception is ::',e)

try:

movies['Box Office'] = movies['Box Office'].replace('NONE', '0')

movies['Box Office'] = movies['Box Office'].replace('NA', '0')

movies['Box Office'] = movies['Box Office'].apply(convertCurrencytoNumber)

movies['Box Office'].fillna(0, inplace=True)

#print( movies['Box Office'])

print("Completed : cleaning of cleanBoxOffice")

except Exception as e:

print('Exception in cleaning Runtime column')

print('Exception is ::',e)

#==============================================================================

#Calculating column for Target Variable

#==============================================================================

try:

movies['RatingDiff'] = abs (movies['audience\_score'] - movies['critic\_score'])

except Exception as e:

print('Exception in calculateRatingDifference')

print('Exception is ::',e)

#==============================================================================

#Calculating Genre Feature

#==============================================================================

try:

genreVectorizor = TfidfVectorizer(lowercase=True, norm=None, stop\_words='english', use\_idf=False)

g= genreVectorizor.fit\_transform(movies['Genre'].values.astype('U')).toarray()

#print(genreVectorizor.get\_feature\_names())

df1 = pd.DataFrame(g, columns=genreVectorizor.get\_feature\_names())

frames = [movies, df1]

movies = pd.concat(frames,axis=1, join\_axes=[movies.index]) #,columns=genreVectorizor.get\_feature\_names())

print("Completed : Process Genre of movies")

except Exception as e:

print('Exception in processGenre')

print('Exception is ::',e)

#Calculating Director Feature based on Facebook Likes

# =============================================================================

director\_likes = {}

with open('director\_rating.csv',encoding="utf8") as csvfile:

reader = csv.DictReader(csvfile)

for row in reader:

director\_likes[row['director\_name']] = row['director\_facebook\_likes']

def getDirector1Score(x):

x = x.split(',')[0]

x= str(x)

if x in director\_likes:

x = director\_likes[x]

else:

x = 0

return float(x)

movies['Director\_1\_Score'] = movies['Directed By'].apply(getDirector1Score)

# =============================================================================

#Calculating ToP-3 Actors Feature based on Facebook Likes

# =============================================================================

actor\_likes = {}

with open('actor\_rating.csv',encoding="utf8") as csvfile:

reader = csv.DictReader(csvfile)

for row in reader:

actor\_likes[row['actor\_1\_name'].strip()] = row['actor\_1\_facebook\_likes']

def getactor1Score(x):

x= str(x)

x = x.split(',')[0]

if x in actor\_likes:

x = actor\_likes[x]

else:

x = 0

return float(x)

def getactor2Score(x):

try:

x= str(x)

x = x.split(',')

if len(x) >= 2:

x= x[1]

if x in actor\_likes:

x = actor\_likes[x]

else:

x = 0

else:

x = 0

return float(x)

except:

return float(0)

def getactor3Score(x):

try:

x= str(x)

x = x.split(',')

if len(x) >= 3:

x= x[2]

if x in actor\_likes:

x = actor\_likes[x]

else:

x = 0

else:

x = 0

return float(x)

except:

return float(0)

movies['actor\_names\_1'] = movies['actor\_names'].apply(getactor1Score)

movies['actor\_names\_2'] = movies['actor\_names'].apply(getactor2Score)

movies['actor\_names\_3'] = movies['actor\_names'].apply(getactor3Score)

# =============================================================================

#Calculating Studio Feature

# =============================================================================

try:

studio\_list\_per\_movie = list(map(str,(movies['Studio'])))

studioSet = set()

for i in studio\_list\_per\_movie:

split\_studio = list(map(str, i.split(',')))

for j in split\_studio:

studioSet.add(j.lower().strip())

studioNamesVectorizor = CountVectorizer(stop\_words='english', vocabulary = studioSet)

x= studioNamesVectorizor.fit\_transform(movies['Studio'].values.astype('U')).toarray()

df1 = pd.DataFrame(x, columns=studioNamesVectorizor.get\_feature\_names())

frames = [movies, df1]

movies = pd.concat(frames,axis=1, join\_axes=[movies.index])

print("Completed : Process Studio of movies")

except Exception as e:

print('Exception in processStudio')

print('Exception is ::',e)

# =============================================================================

#Calculating Writer Feature based on Dummies

# =============================================================================

try:

writer\_list\_per\_movie = list(map(str,(movies['Written By'])))

writerSet = set()

for i in writer\_list\_per\_movie:

split\_writer = list(map(str, i.split(',')))

for j in split\_writer:

writerSet.add(j.lower().strip())

writerNamesVectorizor = CountVectorizer(stop\_words='english', vocabulary = writerSet)

x= writerNamesVectorizor.fit\_transform(movies['Written By'].values.astype('U')).toarray()

df1 = pd.DataFrame(x, columns=writerNamesVectorizor.get\_feature\_names())

frames = [movies, df1]

movies = pd.concat(frames,axis=1, join\_axes=[movies.index])

print("Completed : Process Writer of movies")

except Exception as e:

print('Exception in processWriter')

print('Exception is ::',e)

# =============================================================================

#Removing Unwanted String based columns for modelling

# =============================================================================

print("Removing : Unwanted Columns")

movies = movies.drop(['movie\_id','actor\_names','actor\_links','synopsis','In Theaters','Genre','Studio','Directed By','Rating','Written By'],1)

movies = movies.fillna(0)

# =============================================================================

#Start Modelling based on above calculated Features

# =============================================================================

seed = 123

scoring = 'accuracy'

validation\_size = 0.30

y = abs (movies['audience\_score'] - movies['critic\_score'])

#mov = movies[['Runtime', 'actor\_names\_1','actor\_names\_2','actor\_names\_3','Director\_1\_Score']].copy()

mov = movies.iloc[:,2:25].copy()

#mov.info()

X\_train, X\_validation, Y\_train, Y\_validation = model\_selection.train\_test\_split(mov, y, test\_size=validation\_size, random\_state=seed)

# # # Spot Check Algorithms

models = []

#models.append(('LR', LogisticRegression()))

#models.append(('KNN', KNeighborsClassifier()))

#models.append(('CART', DecisionTreeClassifier()))

#models.append(('mlr',linear\_model.LinearRegression()))

#models.append(('RF', RandomForestClassifier(n\_estimators=2500, n\_jobs=15,criterion="entropy",max\_features='log2',random\_state=150,max\_depth=600,min\_samples\_split=163)))

models.append(('SVC', SVC()))

models.append(('SVR', SVR()))

print(">>>>>>>>>>> Create the Model <<<<<<<<<<<<<<")

results = []

names = []

from math import sqrt

for name, model in models:

model.fit(X\_train,Y\_train)

predicted=model.predict(X\_validation)

mse= mean\_squared\_error(Y\_validation.values,predicted)

print(name, 'mean square error:', mse)

rmse = sqrt(mse)

print(name, 'root mean square error:', rmse)

meanY = np.mean(Y\_validation)

meanP = np.mean(predicted)

accuracy= np.divide(meanP,meanY)

seed2=100

accuracyp = np.multiply(accuracy,seed2)

print('The accuracy of the model is:', accuracyp,'%')

# acc = (meanP/mean)

# print(acc)

# # # evaluate each model in turn

'''

results = []

names = []

for name, model in models:

kfold = model\_selection.KFold(n\_splits=10, random\_state=seed)

cv\_results = model\_selection.cross\_val\_score(model, X\_train,Y\_train, scoring=scoring)

results.append(cv\_results)

names.append(name)

msg = "%s: %f (%f)" % (name, cv\_results.mean(), cv\_results.std())

print(msg)

# # =============================================================================

'''