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_s
plit=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)
# # ================================================================================
'''
```