Furkan Gündoğan 1821221005

In [101…
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [102…
```
imdb = pd.read_csv('IMDB-Movie-Data.csv')
```

In [103…
```
imdb.drop(labels=["Rank","Description","Actors"], axis=1,inplace=True)
```

Before start, I took off Rank, Description, Actors columns. Beacuse Rank is just index value which given by dataset-creator. Description and Actors are have long texts.

1) Print the first 5 elements of your DataFrame using the command head(). How many features are there and what are their types (e.g., numeric, nominal)?

```
My dataset: IMDB's 1000 movies datas from 2006 to 2016.
My dataset has 9 features;
    Nominal features:
        Title: Title of Movie
        Genre: Genres of Movie
        Director: Director of Movie
    Numeric features:
        Year: Release Year of Movie
        Runtime: Total duration of Movie
        Rating: Average ratings of audiences
        Votes: Total vote count of audiences
        Revenue: Earned money from movie
        Metascore: Average ratings of film critics
```

In [104…
```
imdb.head()
```

Out[104…

	Title	Genre	Director	Year	Runtime	Rating	Votes	Revenue	Metascore
0	Guardians of the Galaxy	Action,Adventure,Sci-Fi	James Gunn	2014	121	8.1	757074	333.13	NaN
1	Prometheus	Adventure,Mystery,Sci-Fi	Ridley Scott	2012	124	7.0	485820	126.46	65.0
2	Split	Horror,Thriller	M. Night Shyamalan	2016	117	7.3	157606	138.12	62.0
3	Sing	Animation,Comedy,Family	Christophe Lourdelet	2016	108	7.2	60545	270.32	59.0
4	Suicide Squad	Action,Adventure,Fantasy	David Ayer	2016	123	6.2	393727	325.02	40.0

2) Compute and display summary statistics for each numeric feature available in the dataset. These must include the minimum value, maximum value, mean, range, standard deviation, variance, count, and 25:50:75% percentiles.

In [105…
```
# 2.a) Year
a = np.array(imdb.Year)
print("2.a) Year\n");
```

```
print("Sum: ",a.sum());
print("Max: ", a.max());print("Min: ",a.min());
print("Range: ",a.max()-a.min());
print("Mean ", a.mean());
print("Std: ",a.std());print("Variance: ",a.var());
print("Count: ",len(a));
print("\nPercentiles");
print("25%:",np.percentile(a,25));
print("50%:",np.percentile(a,50));
print("75%:",np.percentile(a,75));
```

```
2.a) Year

Sum:  2012783
Max:  2016
Min:  2006
Range:  10
Mean  2012.783
Std:  3.2043581260527043
Variance:  10.267911
Count:  1000

Percentiles
25%: 2010.0
50%: 2014.0
75%: 2016.0
```

In [106…

```
# 2.b) Runtime
b = np.array(imdb.Runtime)
print("2.b) Runtime\n");
print("Sum: ",b.sum());
print("Max: ", b.max());print("Min: ",b.min());
print("Range: ",b.max()-b.min());
print("Mean ", b.mean());
print("Std: ",b.std());print("Variance: ",b.var());
print("Count: ",len(b));
print("\nPercentiles");
print("25%:",np.percentile(b,25));
print("50%:",np.percentile(b,50));
print("75%:",np.percentile(b,75));
```

```
2.b) Runtime

Sum:  113172
Max:  191
Min:  66
Range:  125
Mean  113.172
Std:  18.80150036566231
Variance:  353.49641599999995
Count:  1000

Percentiles
25%: 100.0
50%: 111.0
75%: 123.0
```

In [107…

```
# 2.c) Rating
c = np.array(imdb.Rating)
print("2.c) Rating\n");
print("Sum: ",c.sum());
print("Max: ", c.max());print("Min: ",c.min());
print("Range: ",c.max()-c.min());
print("Mean ", c.mean());
```

```
print("Std: ",c.std());print("Variance: ",c.var());
print("Count: ",len(c));
print("\nPercentiles");
print("25%:",np.percentile(c,25));
print("50%:",np.percentile(c,50));
print("75%:",np.percentile(c,75));
```

```
2.c) Rating

Sum:  6723.2
Max:  9.0
Min:  1.9
Range:  7.1
Mean  6.723199999999999
Std:  0.9449559566455994
Variance:  0.8929417599999999
Count:  1000

Percentiles
25%: 6.2
50%: 6.8
75%: 7.4
```

In [108…

```
# 2.d) Votes
d = np.array(imdb.Votes)
print("2.d) Votes\n");
print("Sum: ",d.sum());
print("Max: ", d.max());print("Min: ",d.min());
print("Range: ",d.max()-d.min());
print("Mean ", d.mean());
print("Std: ",d.std());print("Variance: ",d.var());
print("Count: ",len(d));
print("\nPercentiles");
print("25%:",np.percentile(d,25));
print("50%:",np.percentile(d,50));
print("75%:",np.percentile(d,75));
```

```
2.d) Votes

Sum:  169808255
Max:  1791916
Min:  61
Range:  1791855
Mean  169808.255
Std:  188668.2425873257
Variance:  35595705760.989975
Count:  1000

Percentiles
25%: 36309.0
50%: 110799.0
75%: 239909.75
```

In [109…

```
# 2.e) Revenue (Million $)
e = np.array(imdb.Revenue)
print("2.e) Revenue (Million $)");
print("Calculations could not be performed because of missing values\n");
print("Sum: ",e.sum());
print("Max: ", e.max());print("Min: ",e.min());
print("Range: ",e.max()-e.min());
print("Mean ", e.mean());
print("Std: ",e.std());print("Variance: ",e.var());
print("Count: ",len(e));
print("\nPercentiles");
```

```
print("25%:",np.percentile(e,25));
print("50%:",np.percentile(e,50));
print("75%:",np.percentile(e,75));
```

```
2.e) Revenue (Million $)
Calculations could not be performed because of missing values

Sum:  nan
Max:  nan
Min:  nan
Range:  nan
Mean  nan
Std:  nan
Variance:  nan
Count:  1000

Percentiles
25%: nan
50%: nan
75%: nan
```

In [110…

```
# 2.f) Metascore
f = np.array(imdb.Metascore)
print("2.f) Metascore");
print("Calculations could not be performed because of missing values\n");
print("Sum: ",f.sum());
print("Max: ", f.max());
print("Min: ",f.min());
print("Range: ",f.max()-f.min());
print("Mean ", f.mean());
print("Std: ",f.std());print("Variance: ",f.var());
print("Count: ",len(f));
print("\nPercentiles");
print("25%:",np.percentile(f,25));
print("50%:",np.percentile(f,50));
print("75%:",np.percentile(f,75));
```

```
2.f) Metascore
Calculations could not be performed because of missing values

Sum:  nan
Max:  nan
Min:  nan
Range:  nan
Mean  nan
Std:  nan
Variance:  nan
Count:  1000

Percentiles
25%: nan
50%: nan
75%: nan
```

3) Detect the count/percentage of missing values in every column of the dataset and apply an appropriate approach to deal with the missing values.

In [111…

```
imdb.isnull().sum()
```

Out[111…

```
Title          0
Genre          0
Director       0
Year           0
Runtime        0
```

```
Rating          0
Votes           0
Revenue       128
Metascore      65
dtype: int64
```

There are 2 columns that have missing values: Revenue and Metascore

In [112…
```
rev_missing_count=np.count_nonzero(np.isnan(imdb.Revenue));
print("Missing value count of Revenue:",rev_missing_count,"in",len(imdb.Revenue))
print("Percentage: ",rev_missing_count*100/len(imdb.Revenue),"%")

meta_missing_count=np.count_nonzero(np.isnan(imdb.Metascore))
print("Missing value count of Metascore:",meta_missing_count,"in",len(imdb.Revenue))
print("Percentage: ",meta_missing_count*100/len(imdb.Metascore),"%")
# np.count_nonzero counts true values inside parameter, so
# we can get count of missing values
```

```
Missing value count of Revenue: 128 in 1000
Percentage:  12.8 %
Missing value count of Metascore: 65 in 1000
Percentage:  6.5 %
```

Handling missing values of Revenue:
Revenue has too many missing values. But since it is an important feature, I decided to fill the missing values with the median instead of drop it.

In [113…
```
imdb["Revenue"].fillna(imdb["Revenue"].median(),inplace=True)
```

In [114…
```
imdb.isnull().sum()
```

Out[114…
```
Title          0
Genre          0
Director       0
Year           0
Runtime        0
Rating         0
Votes          0
Revenue        0
Metascore     65
dtype: int64
```

Handling missing values of Metascore:
I found it convenient to fill in the missing Metascore values with Rating values. Because both are a kind of ratings of audiences.

In [115…
```
imdb["Metascore"].fillna(imdb["Rating"]*10,inplace=True)
```

In [116…
```
imdb.isnull().sum()
```

Out[116…
```
Title         0
Genre         0
Director      0
Year          0
Runtime       0
Rating        0
Votes         0
Revenue       0
```

```
Metascore    0
dtype: int64
```

My dataset has no longer have a feature with missing value

In [117…

```
imdb.head()
```

Out[117…

	Title	Genre	Director	Year	Runtime	Rating	Votes	Revenue	Metascore
0	Guardians of the Galaxy	Action,Adventure,Sci-Fi	James Gunn	2014	121	8.1	757074	333.13	81.0
1	Prometheus	Adventure,Mystery,Sci-Fi	Ridley Scott	2012	124	7.0	485820	126.46	65.0
2	Split	Horror,Thriller	M. Night Shyamalan	2016	117	7.3	157606	138.12	62.0
3	Sing	Animation,Comedy,Family	Christophe Lourdelet	2016	108	7.2	60545	270.32	59.0
4	Suicide Squad	Action,Adventure,Fantasy	David Ayer	2016	123	6.2	393727	325.02	40.0

4) Data visualization. To illustrate the feature distributions, apply the following visualization methods:

a. Histogram

b. Scatter plot

c. Boxplot

d. Barplot

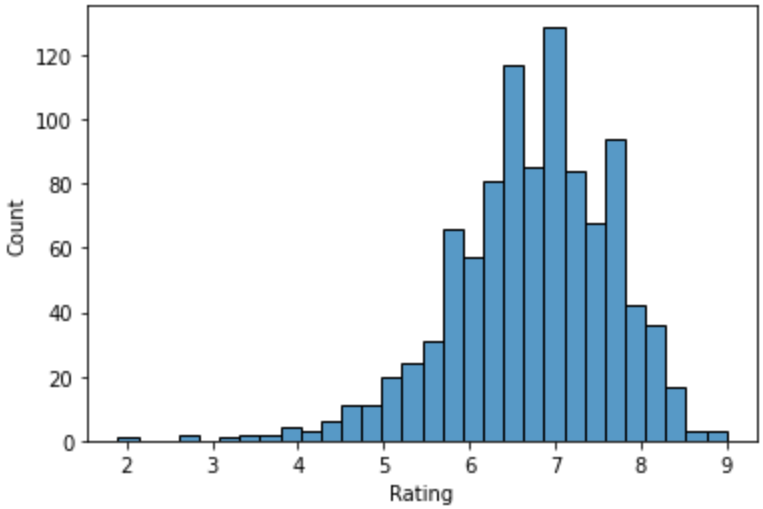
e. Pair plot

In [118…

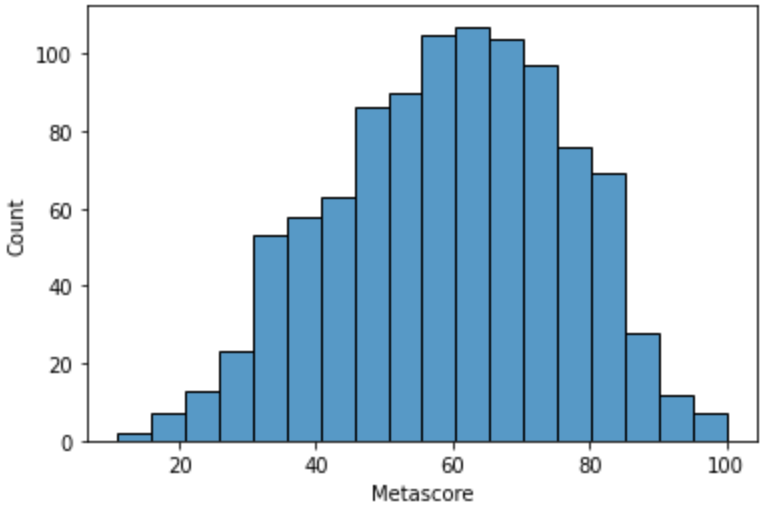
```
print("4.a) Histogram")
copy=imdb.copy()
print("")
print("Rating-Count histogram is left-skewed. We can see that most of the movies have beer
sns.histplot(copy["Rating"])
plt.show()

print("Metascore-Count histogram seems symmetric, looks like Rating-Count.")
sns.histplot(copy["Metascore"])
plt.show()

print("Runtime-Count histogram is right-skewed. Most of the movies are 90-120 minutes. The
sns.histplot(copy["Runtime"])
plt.show()

print("Year-Count histogram is left-skewed. In my dataset, film count increases every year
sns.histplot(copy["Year"])
plt.show()

print("Revenue-Count histogram is right-skewed. We can see that the number of movies that
sns.histplot(copy["Revenue"])
plt.show()

print("Votes-Count histogram is right skewed. The number of movies voted by large audience
sns.histplot(copy["Votes"])
plt.show()
```

4.a) Histogram

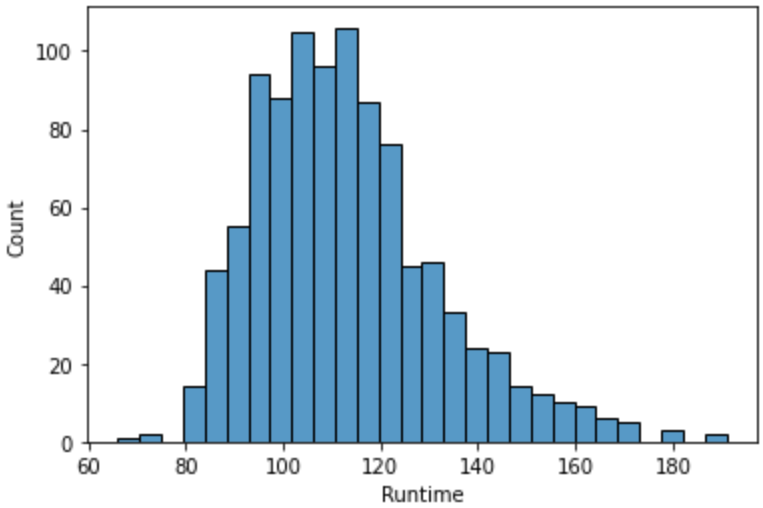
Rating-Count histogram is left-skewed. We can see that most of the movies have been rated between 6-8.



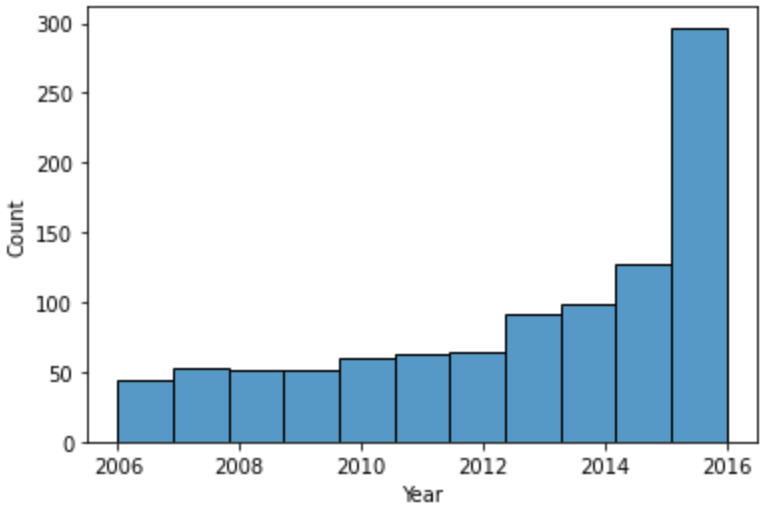
Metascore-Count histogram seems symmetric, looks like Rating-Count.



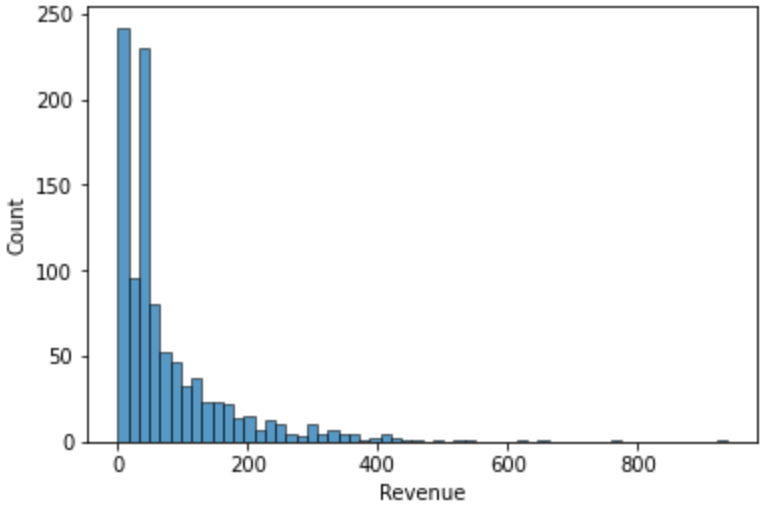
Runtime-Count histogram is right-skewed. Most of the movies are 90-120 minutes. There are only few films over 3 hours or less than 80 minutes



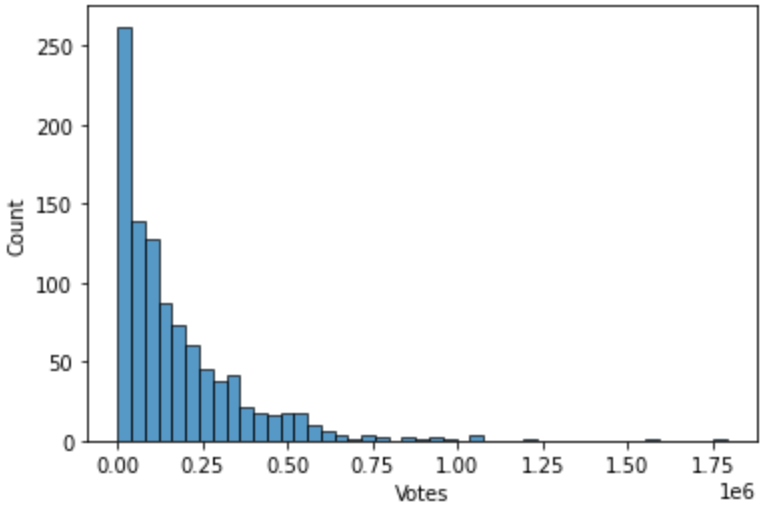
Year-Count histogram is left-skewed. In my dataset, film count increases every year.



Revenue-Count histogram is right-skewed. We can see that the number of movies that have ma
de huge profits is very few.



Votes-Count histogram is right skewed. The number of movies voted by large audiences is qu
ite low.

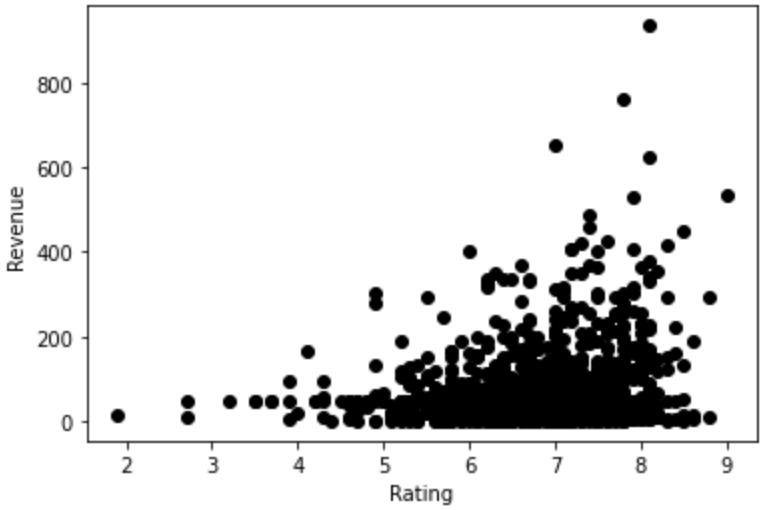


In [119…

```
print("4.b) Scatter Plot")
print("")

print("Rating-Revenue -> Exponential positive correlation. It can be said with certainty t
plt.plot(copy["Rating"],copy["Revenue"], 'o', color='black');
plt.xlabel("Rating");
plt.ylabel("Revenue");
```

4.b) Scatter Plot

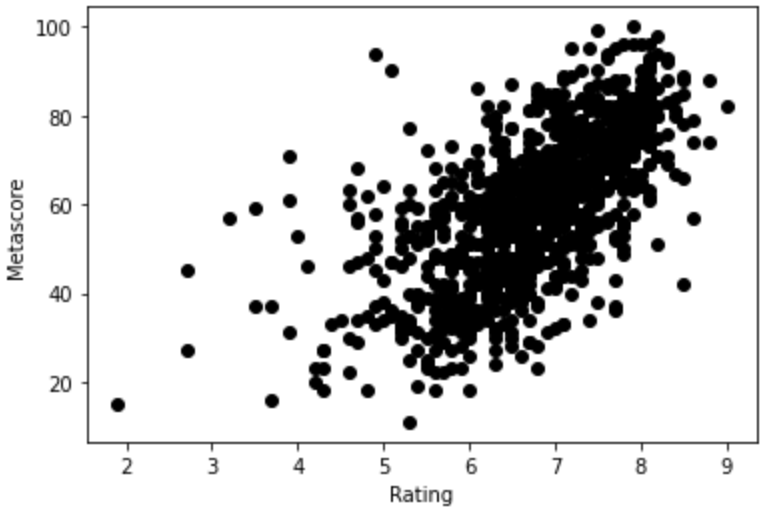
```
Rating-Revenue -> Exponential positive correlation. It can be said with certainty that a l
ow-rated film earned little revenue.
```



In [120…

```
print("Rating-Metascore -> Positive correlation. But also there are outliers.")
print("These outliers mean film-critics and standard-audiences have opposite opinions abou
plt.plot(copy["Rating"],copy["Metascore"], 'o', color='black');
plt.xlabel("Rating")
plt.ylabel("Metascore");
```

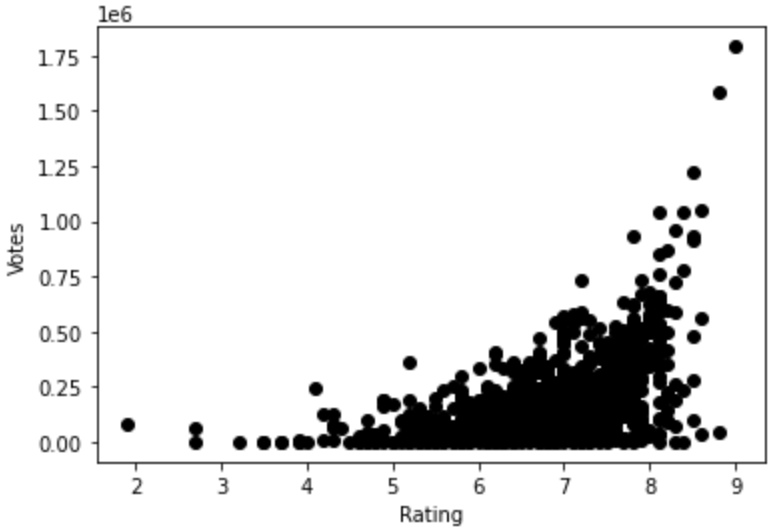
```
Rating-Metascore -> Positive correlation. But also there are outliers.
These outliers mean film-critics and standard-audiences have opposite opinions about some
films.
```



In [121…

```
print("Rating-Votes -> Exponential positive correlation.")
print("Its certain that; number of votes of a low-rated-movie is low.")
print("And the movie with a high rating is likely to have a large number of votes.")
plt.plot(copy["Rating"],copy["Votes"], 'o', color='black');
plt.xlabel("Rating")
plt.ylabel("Votes");
```

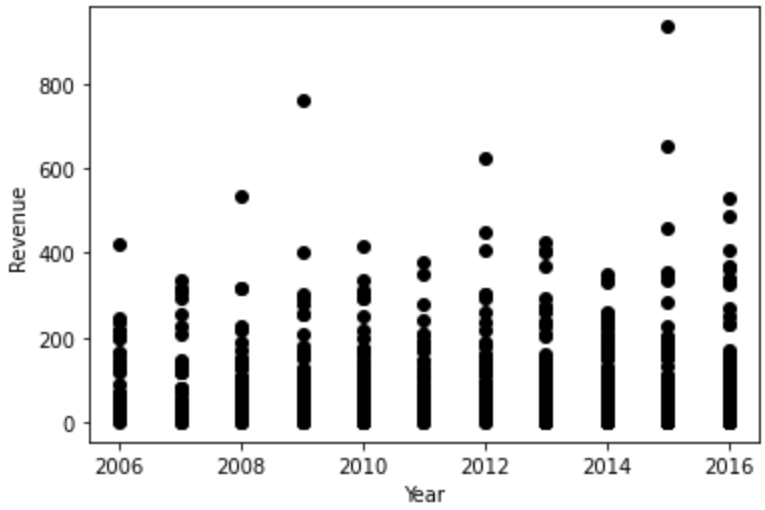
```
Rating-Votes -> Exponential positive correlation.
Its certain that; number of votes of a low-rated-movie is low.
And the movie with a high rating is likely to have a large number of votes.
```



In [122…

```
print("Year-Revenue -> No correlation.")
plt.plot(copy["Year"],copy["Revenue"], 'o', color='black');
plt.xlabel("Year")
plt.ylabel("Revenue");
```

```
Year-Revenue -> No correlation.
```



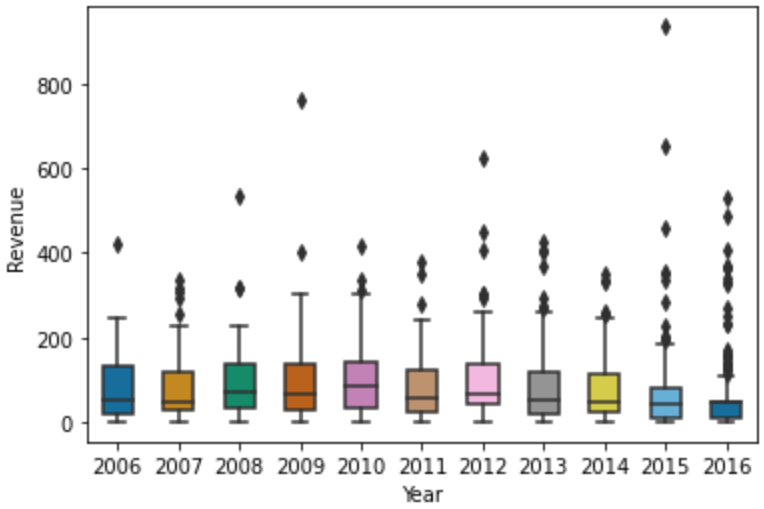
In [123…

```
print("4.c) Boxplot")
print("")
print("Year-Revenue")
print("Box plots are mostly under 300mn $. 2015 has highest outlier. 2016 has most outlie
sns.boxplot(y='Revenue', x='Year',
                data=copy,
                  width=0.5,
                  palette="colorblind")
plt.show()

print("Year-Runtime")
print("They are mostly symmetric. 2009-2012-2016 have bigest intervals.")
sns.boxplot(y='Runtime', x='Year',
                  data=copy,
                  width=0.5,
                  palette="colorblind")
plt.show()

print("Year-Votes")
print("Intervals looks same except 2015 and 2016. They have small intervals and many outli
sns.boxplot(y='Votes', x='Year',
                  data=copy,
```

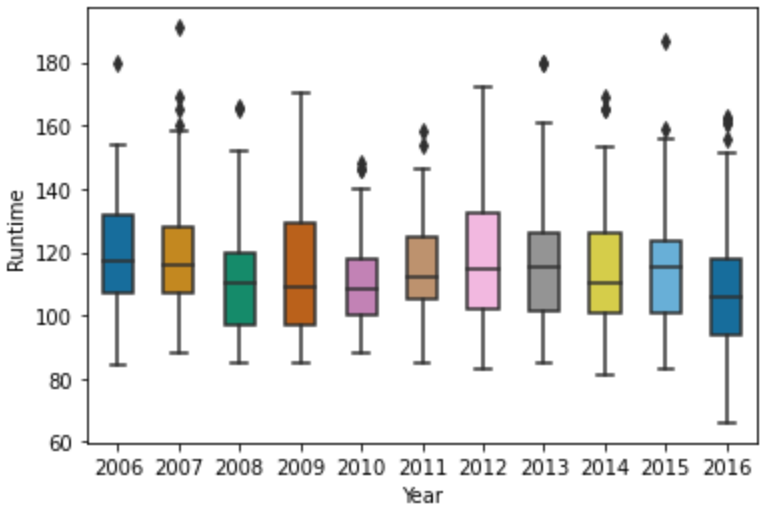
```
                    width=0.5,
                    palette="colorblind")
plt.show()
```

4.c) Boxplot

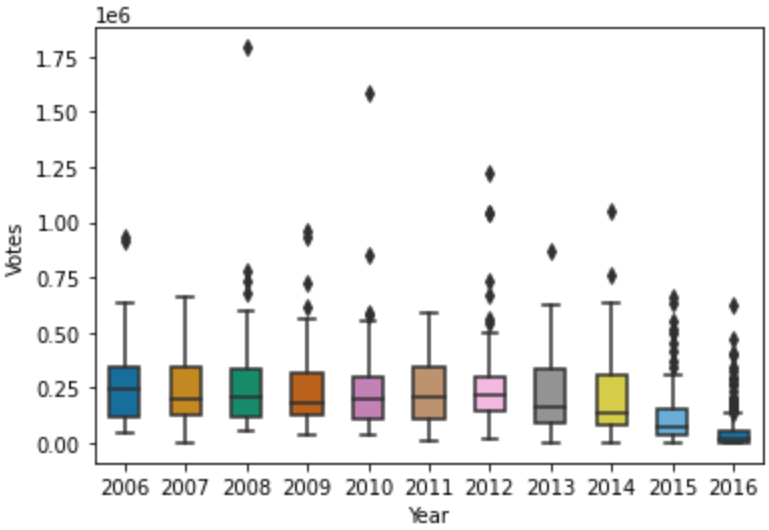
Year-Revenue
Box plots are mostly under 300mn $. 2015 has highest outlier. 2016 has most outliers.



Year-Runtime
They are mostly symmetric. 2009-2012-2016 have bigest intervals.



Year-Votes
Intervals looks same except 2015 and 2016. They have small intervals and many outliers. Bu
t highest outlier in 2008 by The Dark Knight (2.4mn Votes)

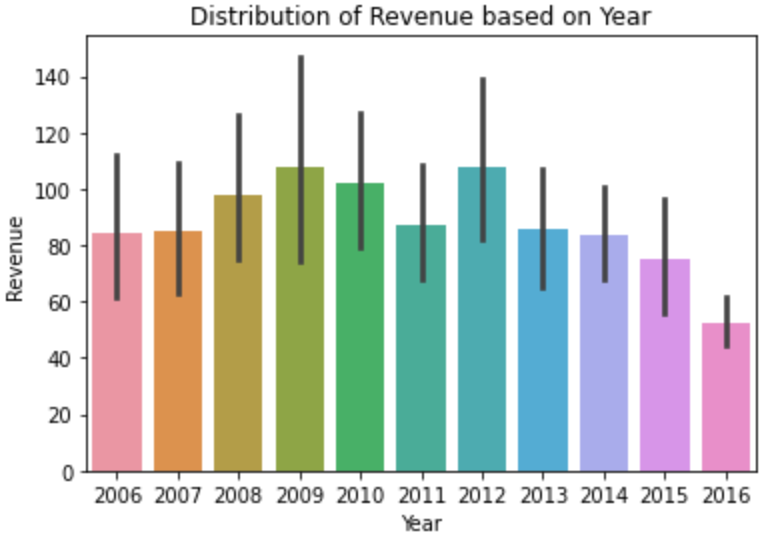


In [124…

```
print("4.d) Barplot")
print("")

print("The distribution of Ratings based on Years seems equal on Barplot")
sns.barplot(x="Year", y="Rating", data=copy)
plt.title("Distribution of Rating based on Year")
plt.show()

print("The distribution of Revenues based on Years seems symmetric on Barplot")
sns.barplot(x="Year", y="Revenue", data=copy)
plt.title("Distribution of Revenue based on Year")
plt.show()
```

```
4.d) Barplot

The distribution of Ratings based on Years seems equal on Barplot
```


```
The distribution of Revenues based on Years seems symmetric on Barplot
```

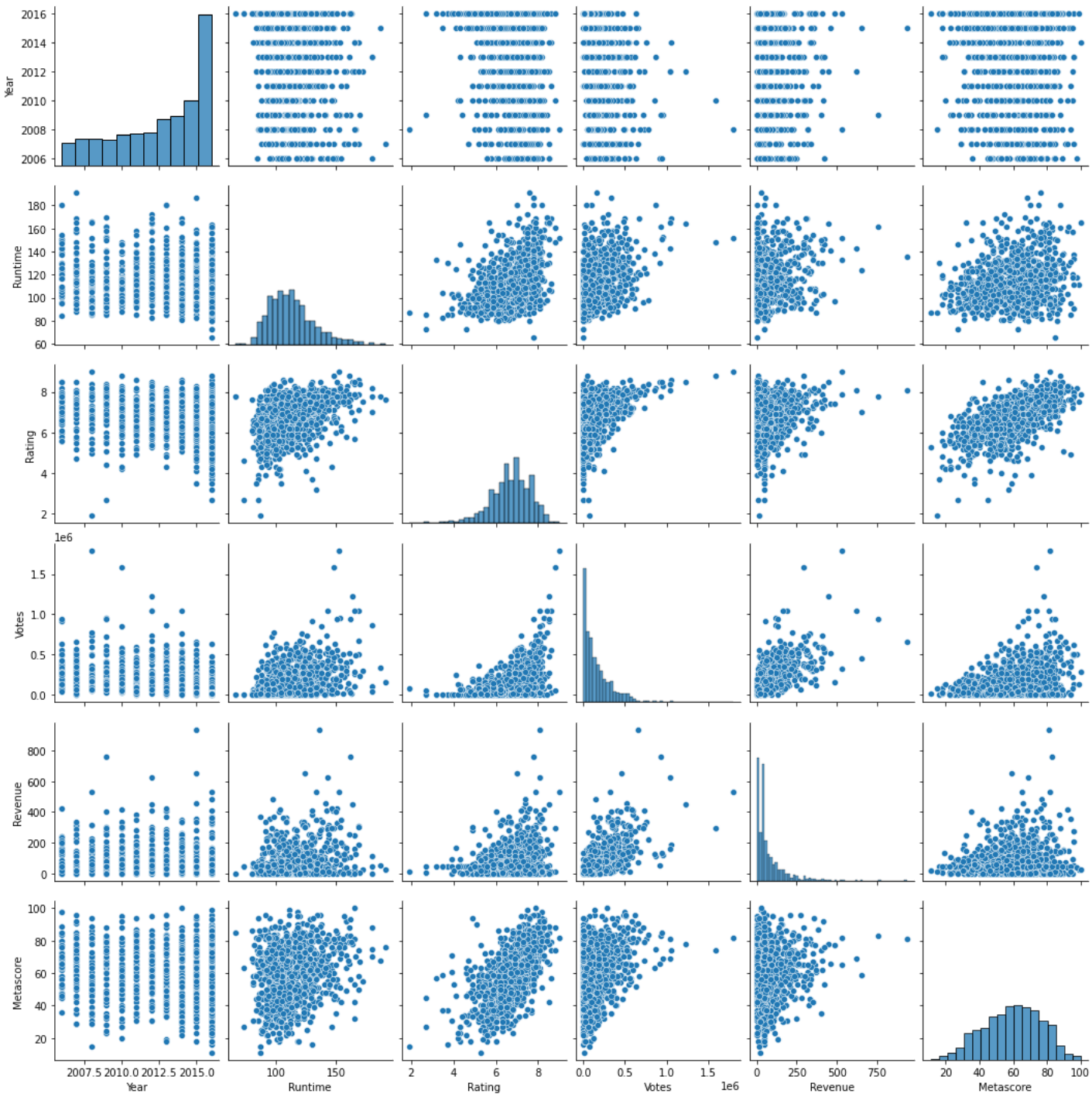


In [125…

```
print("4.e) Pair Plot")
print("")
print("Pair-plot -> Relationships between all numerical columns:")
print("Positive-correlation pairs:")
print("          Rating-Runtime")
print("          Rating-Metascore")
print("          Rating-Votes")
print("          Rating-Revenue")
print("          Votes-Runtime")
print("          Votes-Revenue")
print("          Votes-Metascore")
print("          Runtime-Revenue")
```

```
print("          Metascore-Revenue")
print("Negative-correlation pairs: None")
print("No-correlation pairs:")
print("          Year-Any")
print("          Runtime-Metascore")


sns.pairplot(copy)
plt.show()
```

```
4.e) Pair Plot

Pair-plot -> Relationships between all numerical columns:
Positive-correlation pairs:
          Rating-Runtime
          Rating-Metascore
          Rating-Votes
          Rating-Revenue
          Votes-Runtime
          Votes-Revenue
          Votes-Metascore
          Runtime-Revenue
          Metascore-Revenue
Negative-correlation pairs: None
No-correlation pairs:
          Year-Any
          Runtime-Metascore
```



5.) Apply PCA to perform dimensionality reduction on the dataset within 2-dimensional (feature) space.

I need to scale dataset before apply PCA.
Firstly, i converted Genre and Director features by LabelEncoder.
After that i did scale some of my numeric features: Genre, Director, Year, Votes, Revenue.
Then i dropped Title column, because its unique and i did not need to use this feature.

In [126…

```
from sklearn.preprocessing import LabelEncoder

le_genre = LabelEncoder()
le_genre.fit(imdb["Genre"])
encoded_genre_training = le_genre.transform(imdb["Genre"])
imdb["Genre"] = encoded_genre_training
```

```
le_director = LabelEncoder()
le_director.fit(imdb["Director"])
encoded_director_training = le_director.transform(imdb["Director"])
imdb["Director"] = encoded_director_training


from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

genre_scaled = np.array(imdb["Genre"]).reshape(-1, 1)
director_scaled = np.array(imdb["Director"]).reshape(-1, 1)
year_scaled = np.array(imdb["Year"]).reshape(-1, 1)
runtime_scaled = np.array(imdb["Runtime"]).reshape(-1, 1)
votes_scaled = np.array(imdb["Votes"]).reshape(-1, 1)
revenue_scaled = np.array(imdb["Revenue"]).reshape(-1, 1)

copy_imdb = imdb.copy()
copy_imdb["Genre"] = scaler.fit_transform(genre_scaled)
copy_imdb["Director"] = scaler.fit_transform(director_scaled)
copy_imdb["Year"] = scaler.fit_transform(year_scaled)
copy_imdb["Runtime"] = scaler.fit_transform(runtime_scaled)
copy_imdb["Votes"] = scaler.fit_transform(votes_scaled)
copy_imdb["Revenue"] = scaler.fit_transform(revenue_scaled)
```

In [127…

```
copy_imdb.drop(labels=["Title"], axis=1,inplace=True)
```

In [128…

```
copy_imdb.head()
```

Out[128…

	Genre	Director	Year	Runtime	Rating	Votes	Revenue	Metascore
0	-1.459278	-0.335620	0.379795	0.416350	8.1	3.112690	2.623377	81.0
1	-0.258280	1.027352	-0.244355	0.575911	7.0	1.674960	0.494284	65.0
2	1.526988	0.343173	1.003945	0.203601	7.3	-0.064676	0.614405	62.0
3	-0.144672	-1.197578	1.003945	-0.275084	7.2	-0.579129	1.976315	59.0
4	-1.524197	-1.030574	1.003945	0.522724	6.2	1.186839	2.539829	40.0

My dataset has 8 dimensions. I will reduce it to 2 with applying PCA.
Then i will visualize with 'Rating' feature, to see distrubuton of 'being greater or smaller than 7.5'

In [129…

```
from sklearn.preprocessing import StandardScaler
#                                          4=rating column
X, y = copy_imdb.iloc[:,:], copy_imdb.iloc[:,4]

X_pca1 = X.copy()
#todo: scale the input data
X_pca1=StandardScaler().fit_transform(X_pca1)

from sklearn.decomposition import PCA
pca=PCA(n_components=2)
X_redu1=pca.fit_transform(X_pca1)

#reduced data
low = X_redu1[y < 7.5]
high = X_redu1[y >= 7.5]
comp =[0,1]
```

```
fig = plt.figure(figsize = (12,8))
plt.scatter(low[:,0].reshape(1,-1), low[:,1].reshape(1,-1), c = 'red', label = '<7.5', alp
plt.scatter(high[:,0].reshape(1,-1), high[:,1].reshape(1,-1), c = 'green', label = '>=7.5'
plt.legend(['<7.5', '>=7.5'])

myconclusion="As a conclusion, if we assume rating=7.5 is a success criteria, we can say t
plt.title(myconclusion)
plt.show()
```

Classification

KNN (K-Nearest Neighbors) Algorithm

```
K value means the number of neighbors to look at. When a value comes, the distance
between the incoming value is
calculated by taking the nearest K element.
After the distance is calculated, the value is assigned to the appropriate class.
```

Implementation

I wanted to do a classification analysis for the movies in my dataset based on whether they bring high or low profits.

In [130…

```
imdb2=imdb.copy()
imdb2
```

Out[130…

	Title	Genre	Director	Year	Runtime	Rating	Votes	Revenue	Metascore
0	Guardians of the Galaxy	11	265	2014	121	8.1	757074	333.130	81.0
1	Prometheus	85	518	2012	124	7.0	485820	126.460	65.0
2	Split	195	391	2016	117	7.3	157606	138.120	62.0
3	Sing	92	105	2016	108	7.2	60545	270.320	59.0

	Title	Genre	Director	Year	Runtime	Rating	Votes	Revenue	Metascore
4	Suicide Squad	7	136	2016	123	6.2	393727	325.020	40.0
...	...	...	...	...	...	...	...	...	...
995	Secret in Their Eyes	144	69	2015	111	6.2	27585	47.985	45.0
996	Hostel: Part II	190	176	2007	94	5.5	73152	17.540	46.0
997	Step Up 2: The Streets	170	324	2008	98	6.2	70699	58.010	50.0
998	Search Party	61	549	2014	93	5.6	4881	47.985	22.0
999	Nine Lives	121	59	2016	87	5.3	12435	19.640	11.0

1000 rows × 9 columns

- I took off Title, Year, Runtime and columns.

- Beacuse i think they won't be very effective to determine film profit classification.

- I also took of Metascore because Rating column is enough.

In [131…
```
imdb2.drop(labels=["Title","Year","Runtime","Metascore"], axis=1,inplace=True)
```

I changed Revenue column values to ones and zeros according to their revenue value

High profit: >= 200 mn

Low profit: < 200 mn

In [132…
```
imdb2.loc[imdb2.Revenue < 200, 'Revenue'] = 0
imdb2.loc[imdb2.Revenue >= 200, 'Revenue'] = 1
```

In [133…
```
imdb2
```

Out[133…

	Genre	Director	Rating	Votes	Revenue
0	11	265	8.1	757074	1.0
1	85	518	7.0	485820	0.0
2	195	391	7.3	157606	0.0
3	92	105	7.2	60545	1.0
4	7	136	6.2	393727	1.0
...	...	...	...	...	...
995	144	69	6.2	27585	0.0
996	190	176	5.5	73152	0.0
997	170	324	6.2	70699	0.0
998	61	549	5.6	4881	0.0
999	121	59	5.3	12435	0.0

1000 rows × 5 columns

I divided dependent and independent variables into different arrays

Independent variables; Genre, Director, Rating and Votes

Target dependent variable: Revenue

In [134…
```
my_x = imdb2.iloc[:, [0,1,2,3]].values
```

In [135…
```
my_y = imdb2.iloc[:, 4].values
```

I divided the 1000-element dataset into two parts as train and test.

- 750 data to train

- 250 data to test

In [136…
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(my_x, my_y, test_size = 0.25, random_s
```

Then data is normalized with StandardScaler to get more successful results.

In [137…
```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

Scaled train dataset

In [138…
```
X_train
```

Out[138…
```
array([[-1.464874  ,  0.36312237, -0.01351204,  0.93753282],
       [-0.56540909,  1.2730897 , -0.42296793, -0.1815421 ],
       [-1.464874  , -1.16802026,  0.29357987,  1.55173568],
       ...,
       [-0.61447081,  1.30033423, -0.5253319 , -0.5155196 ],
       [-0.54905518,  0.61922096, -0.11587602, -0.29552619],
       [ 1.23352072, -0.68306761,  1.01012767,  0.41001941]])
```

Trainings and Predictions

Using the KNN algorithm from K=1 to 10, we trained the model with 750 training data and tested with 250 data.

The Euclidean function is used in distance calculation. As an alternative Manhattan, Minkowski and Hamming can also be used.

For every k value, I put every accuracy score in a list.

K=1 has highest accuracy, K=3 has second highest.

In [139…
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
m=11
acc_list=np.zeros((m-1))
```

```
cm=[]
for i in range (1,m):
    classifier = KNeighborsClassifier(n_neighbors=i, metric='euclidean', p = 2)
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    score=round(accuracy_score(y_test,y_pred),3)
    acc_list[i-1]=score
acc_list
```

Out[139…
```
array([0.92 , 0.904, 0.908, 0.892, 0.896, 0.888, 0.892, 0.884, 0.892,
       0.88 ])
```

Analyzing k=1

Training

In [140…
```
# knn k=1
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=1, metric='euclidean', p = 2)
classifier.fit(X_train, y_train)
```

Out[140…
```
KNeighborsClassifier(metric='euclidean', n_neighbors=1)
```

Prediction

In [141…
```
y_pred = classifier.predict(X_test)
```

Confusion Matrix

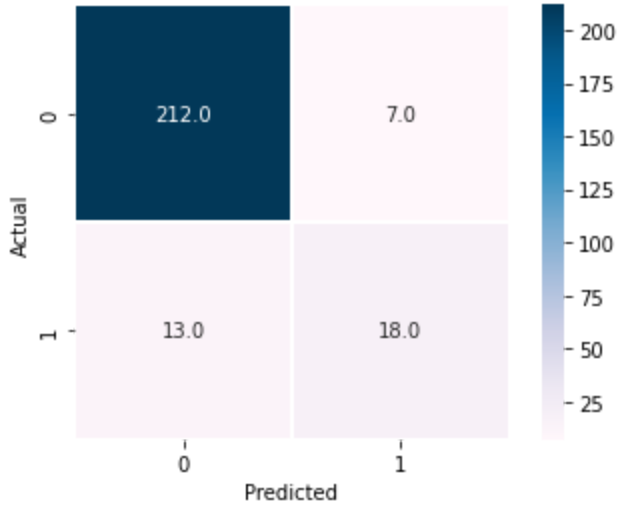
In [142…
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[142…
```
array([[212,   7],
       [ 13,  18]], dtype=int64)
```

Confusion Matrix Heatmap

In [143…
```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
score=round(accuracy_score(y_test,y_pred),3)
print("Accuracy:",score)
cm1=cm
sns.heatmap(cm1,annot=True,fmt=".1f",linewidths=.2,square=True,cmap="PuBu")
plt.xlabel("Predicted");
plt.ylabel("Actual");
plt.show()
```

```
Accuracy: 0.92
```



Classification Report

In [144…

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.94      0.97      0.95       219
         1.0       0.72      0.58      0.64        31

    accuracy                           0.92       250
   macro avg       0.83      0.77      0.80       250
weighted avg       0.91      0.92      0.92       250
```

Result k=1

Accuracy: 92%

Out of 219 low-profit films;
TN 212 were correctly predicted as low profit.
FN 7 of them were incorrectly predicted as high profit.

Out of 31 high-profit movies;
TP 18 of them were correctly predicted as high profit.
FP: 13 were incorrectly predicted as low profit.

Precision and recall value are okay for low-profit-movie predictions.
But precision and recall values of the high-profit-movie predictions are not high enough.
So we can say that this model is not very reliable even accuracy is high.

Analyzing k=3

Training

In [145…

```
# knn k=3
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=3, metric='euclidean', p = 2)
classifier.fit(X_train, y_train)
```

Out[145…

```
KNeighborsClassifier(metric='euclidean', n_neighbors=3)
```

Prediction and Confusion Matrix

In [146…

```
y_pred = classifier.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[146…

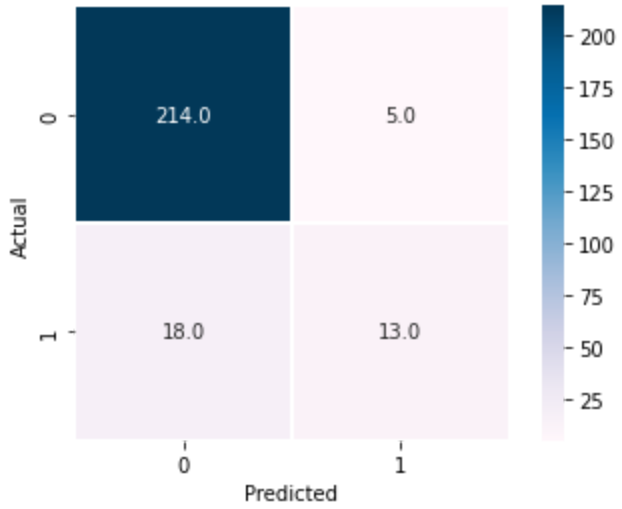
```
array([[214,   5],
       [ 18,  13]], dtype=int64)
```

Confusion Matrix Heatmap

In [147…

```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
score=round(accuracy_score(y_test,y_pred),3)
print("Accuracy:",score)
cm1=cm
sns.heatmap(cm1,annot=True,fmt=".1f",linewidths=.2,square=True,cmap="PuBu")
plt.xlabel("Predicted");
plt.ylabel("Actual");
plt.show()
```

```
Accuracy: 0.908
```



Classification Report

In [148…

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.92      0.98      0.95       219
         1.0       0.72      0.42      0.53        31

    accuracy                           0.91       250
   macro avg       0.82      0.70      0.74       250
weighted avg       0.90      0.91      0.90       250
```

Result k=3

Accuracy 90%

Out of 219 low-profit films;
TN 214 were correctly predicted as low profit.
FN 5 of them were incorrectly predicted as high profit.

```
Out of 31 high-profit movies;
TP 13 of them were correctly predicted as high profit.
FP: 18 were incorrectly predicted as low profit.

For low-profit-movie predictions; Recall is higher than k=1.
But for high-profit-movie predictions; Recall values are lower than k=1
So we should not use this model even accuracy looks high.
```

Comparison of Results

```
Accuracy was high at both k values.
K=1 seems more successful than K=3.
However, it cannot be said that k=1 is a very reliable model.
```

Project Summary

While working on project

I learned how to apply data understanding data preprocessing concepts:

- how to handle missing variables
- applying normalization
- visualizing the results with graphs then data and correlation analysis
- how to apply PCA to perform dimensionality reduction on dataset

I also learned model creation to solve a classification problem:

- preparing train and test data.
- training data with KNN algorithm and making predictions
- making confusion matrix and analyzing results

In [ ]: