Machine Learning Final project report

Healthcare & Ai

1- Mohammad Ahmed Yousry Attia 20221441349 (Healthcare)
2- Clara Magdy Ghaly 20221424647 (Healthcare)
3- Menna tallah Ali Ra'fat 20221376690 (Healthcare)
4- Doaa Samir 2103114 (Ai)

Problem 1: Sales prediction

Introduction

Retailers like Walmart, IKEA, Big Basket, Big Bazaar use sales forecasting for sale predictions of product requirements. Sales forecasting helps business owners get a clear idea of what products are in demand. Accurate sales forecasting will reduce wastage to a significant level and determine the incremental impact on future budgets. In this project, you are required to build a machine learning model for sales prediction.

Dataset

We will use the dummy Shampoo dataset which explained in the following link and can be downloaded from the same link
https://www.kaggle.com/datasets/redwankarimsony/shampoo-saled-dataset

Method

- Use simple linear regression to build your model.

```
import pandas as pd

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from pylab import rcParams
```

```
[2] df=pd.read_csv("shampoo_sales.csv")
```

```
[3] df.isnull().sum()
```

```
Month    0
Sales    0
dtype: int64
```

At first we imported the needed libraries , read the data set shampoo sales file

In the last step we made a check if there is null inside the data set and actually there wasn't .

4. Splitting the data into training and test. 70:30

```
[4] train    =   df[0:int(len(df)*0.7)]
    test     =   df[int(len(df)*0.7):]

    train.shape
```

```
(25, 2)
```

```
[5] from sklearn.linear_model import LinearRegression
```

```
[6] LinearRegression_train = train.copy()
    LinearRegression_test = test.copy()
```

We split the data into test and train to apply linear regression on it and imported sklearn library to add linear regression functions

Linear Regression

```
train_time = [i+1 for i in range(len(train))]
test_time = [i+26 for i in range(len(test))]
print('Training Time instance','\n',train_time)
print('Test Time instance','\n',test_time)
```

```
Training Time instance
 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]
Test Time instance
 [26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36]
```

```
[8] LinearRegression_train = train.copy()
    LinearRegression_test = test.copy()
    LinearRegression_train['time'] = train_time
    LinearRegression_test['time'] = test_time

    print('First few rows of Training Data','\n',LinearRegression_train.head(),'\n')
    print('Last few rows of Training Data','\n',LinearRegression_train.tail(),'\n')
    print('First few rows of Test Data','\n',LinearRegression_test.head(),'\n')
    print('Last few rows of Test Data','\n',LinearRegression_test.tail(),'\n')
```

Output:

```
First few rows of Training Data
    Month  Sales  time
0  1-01  266.0     1
1  1-02  145.9     2
2  1-03  183.1     3
3  1-04  119.3     4
4  1-05  180.3     5

Last few rows of Training Data
    Month  Sales  time
20  2-09  289.9    21
21  2-10  421.6    22
22  2-11  264.5    23
23  2-12  342.3    24
24  3-01  339.7    25

First few rows of Test Data
    Month  Sales  time
25  3-02  440.4    26
26  3-03  315.9    27
27  3-04  439.3    28
28  3-05  401.3    29
29  3-06  437.4    30

Last few rows of Test Data
    Month  Sales  time
31  3-08  407.6    32
32  3-09  682.0    33
33  3-10  475.3    34
34  3-11  581.3    35
35  3-12  646.9    36
```

Now that our training and test data has been modified, let us go ahead use linear regression

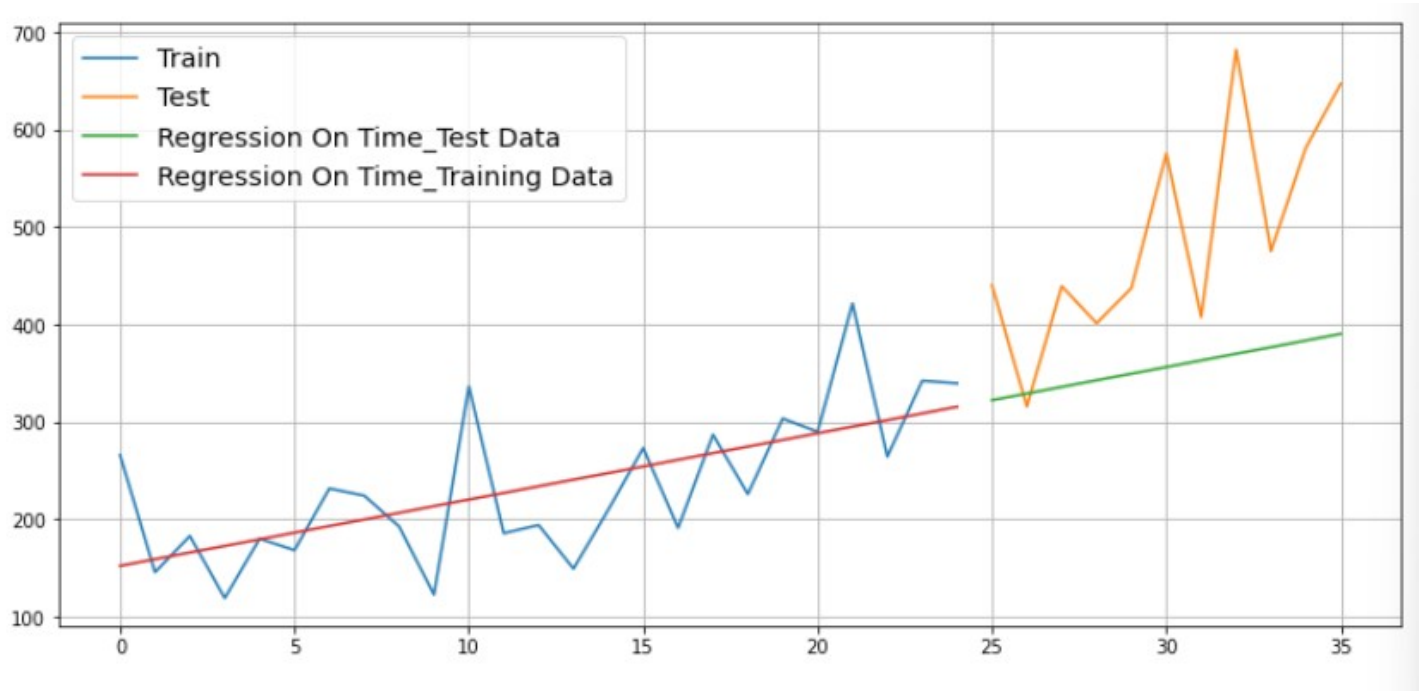
to build the model on the training data and test the model on the test data

```
[9]  from sklearn.linear_model import LinearRegression
     lr = LinearRegression()
     lr.fit(LinearRegression_train[['time']],LinearRegression_train['Sales'].values)
     LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
     train_predictions_model1          = lr.predict(LinearRegression_train[['time']])
     LinearRegression_train['RegOnTime'] = train_predictions_model1

     test_predictions_model1          = lr.predict(LinearRegression_test[['time']])
     LinearRegression_test['RegOnTime'] = test_predictions_model1

     plt.figure(figsize=(13,6))
     plt.plot( train['Sales'], label='Train')
     plt.plot(test['Sales'], label='Test')
     plt.plot(LinearRegression_test['RegOnTime'], label='Regression On Time_Test Data')
     plt.plot(LinearRegression_train['RegOnTime'], label='Regression On Time_Training Data')
     plt.legend(loc='best')
     plt.rcParams['axes.labelsize'] = 16
     plt.rcParams['axes.titlesize'] = 16
     plt.legend(fontsize="x-large")
     plt.grid();
```

Output :



Evaluation (Linear Regression)

```
[10] from sklearn import metrics
     ## Mean Absolute Percentage Error - Function Definition

     def MAPE(y, yhat):
         y, yhat = np.array(y), np.array(yhat)
         try:
             mape =  round(np.sum(np.abs(yhat - y)) / np.sum(y) * 100,2)
         except:
             print("Observed values are empty")
             mape = np.nan
         return mape
```

Output :

```
[11] rmse_model1_train = metrics.mean_squared_error(train['Sales'],train_predictions_model1,squared=False)
     mape_model1_train = MAPE(train['Sales'],train_predictions_model1)
```

```
[12] ## Test Data - RMSE and MAPE

     rmse_model1_test = metrics.mean_squared_error(test['Sales'],test_predictions_model1,squared=False)
     mape_model1_test = MAPE(test['Sales'],test_predictions_model1)
```

```
[13] resultsDf = pd.DataFrame({'Model': 'RegressionOnTime',
                                                    'Test RMSE': [rmse_model1_test],
                               'Test MAPE': [mape_model1_test]},
                              index=['Model 1'])
     resultsDf
```

	Model	Test RMSE	Test MAPE
Model 1	RegressionOnTime	164.563291	27.94

Problem 2: Patient's Sickness Prediction System

Introduction

Traditional healthcare systems became increasingly challenging to cater to the needs of millions of patients. But, with the advent of ML, the paradigm shifted towards value-based treatment.

Dataset

Use the cancer dataset we used in Assignment 2

Method

- Apply clustering, PCA, and t-SNE. Assume the last column to be a feature this time not a label
- Explain the patterns and comment on them

```
[2]  import pandas as pd
     from sklearn.preprocessing import StandardScaler
     from sklearn.cluster import KMeans
     from sklearn.metrics import silhouette_score
```

```
[3]  from sklearn.manifold import TSNE
     from keras.datasets import mnist
     from sklearn.datasets import load_iris
     from numpy import reshape
     import seaborn as sns
     import pandas as pd
```

```
[4]  medical_df=pd.read_csv("Heart.csv")
```

```
     medical_df.isnull().sum()
```

```
Id          0
Age         0
Sex         0
ChestPain   0
RestBP      0
Chol        0
Fbs         0
RestECG     0
MaxHR       0
ExAng       0
Oldpeak     0
Slope       0
Ca          0
Thal        0
AHD         0
dtype: int64
```

At first we imported the needed libraries , read the data set heart file

In the last step we made a check if there is null inside the data set and actually there wasn't .

```
medical_df.head(5)
```

	Id	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope	Ca	Thal	AHD
0	1	63	1	1	145	233	1	2	150	0	2.3	3	0	0	0
1	2	67	1	2	160	286	0	2	108	1	1.5	2	3	1	1
2	3	67	1	2	120	229	0	2	129	1	2.6	2	2	2	1
3	4	37	1	3	130	250	0	0	187	0	3.5	3	0	1	0
4	5	41	0	0	130	204	0	2	172	0	1.4	1	0	1	0

```
[ ] X=medical_df[["Age","Chol"]]
    X
```

	Age	Chol
0	63	233
1	67	286
2	67	229
3	37	250
4	41	204
...	...	...
298	45	264
299	68	193
300	57	131
301	57	236
302	38	175

303 rows × 2 columns

We checked the data to see the content and then filtered it into age and chol only which helps us to preform our task easily and putted those columns inside variable called x

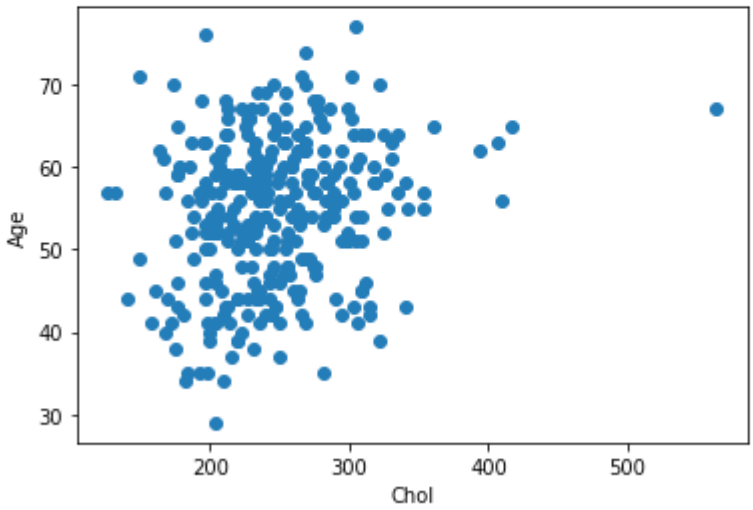
```
[ ]  X.describe()
```

	Age	Chol
count	303.000000	303.000000
mean	54.438944	246.693069
std	9.038662	51.776918
min	29.000000	126.000000
25%	48.000000	211.000000
50%	56.000000	241.000000
75%	61.000000	275.000000
max	77.000000	564.000000

We used the method describe to get us more information about the data which helps us doing the clustering we got the mean and mini and max.

```
import matplotlib.pyplot as plt
plt.scatter(X.Chol,X.Age)
plt.xlabel("Chol")
plt.ylabel("Age")
```

```
Text(0, 0.5, 'Age')
```

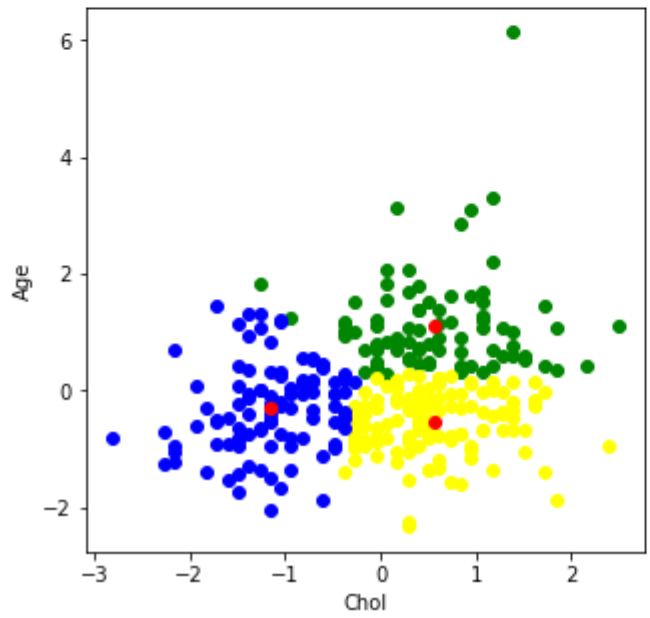


We plotted the data using matplot library

Using Kmeans clustring

```
fig,ax=plt.subplots(figsize=(5,5))
# 1st cluster
plt.scatter(X[K_labels == 0,0],X[K_labels == 0,1],c="green",label="cluster1")
#2nd cluster
plt.scatter(X[K_labels == 1,0],X[K_labels == 1,1],c="yellow",label="cluster2")
#3rd cluster
plt.scatter(X[K_labels == 2,0],X[K_labels == 2,1],c="blue",label="cluster3")
#3 cluster centroids
plt.scatter(k_centroid[:,0],k_centroid[:,1],c="r",label="centroids")
plt.xlabel("Chol")
plt.ylabel("Age")
```

```
Text(0, 0.5, 'Age')
```



We use 3 centroids in kMeans

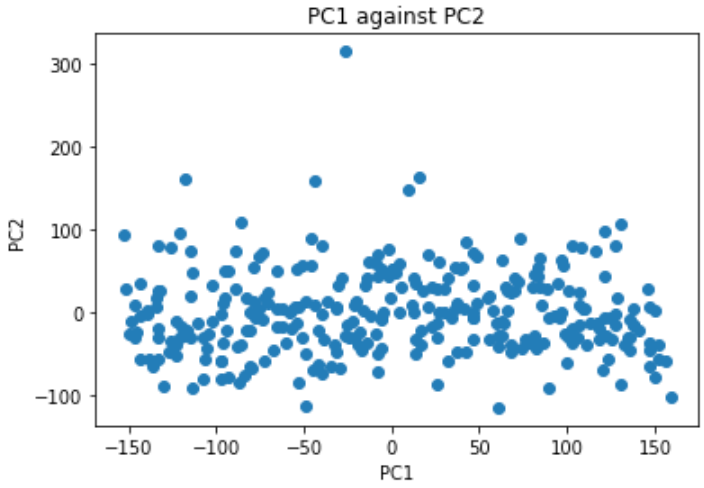
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)

principalComponents = pca.fit_transform(medical_df)

principalDataframe = pd.DataFrame(data = principalComponents, columns = ['PC1', 'PC2'])
print(f'Total number of components used after PCA : {pca.n_components_}')
```

```
Total number of components used after PCA : 2
```

```
plt.scatter(principalDataframe.PC1, principalDataframe.PC2)
plt.title('PC1 against PC2')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```



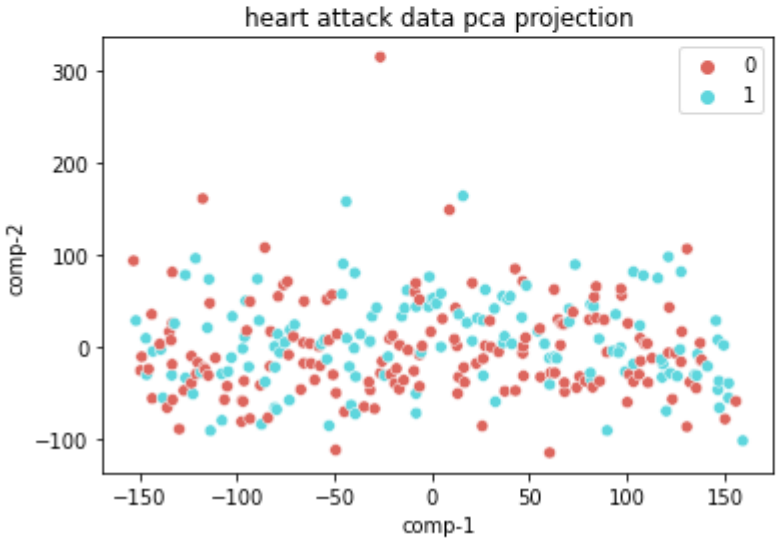
We apply PCA with 2 components on the data and plot it using scatter plot

```
[ ]  y=medical_df.AHD
```

```
x= pd.DataFrame()
x["y"] = y
x["comp-1"] = principalComponents[:,0]
x["comp-2"] = principalComponents[:,1]

sns.scatterplot(x="comp-1", y="comp-2", hue=x.y.tolist(),
                palette=sns.color_palette("hls",2),
                data=x).set(title="heart attack data pca projection")
```

```
[Text(0.5, 1.0, 'heart attack data pca projection')]
```

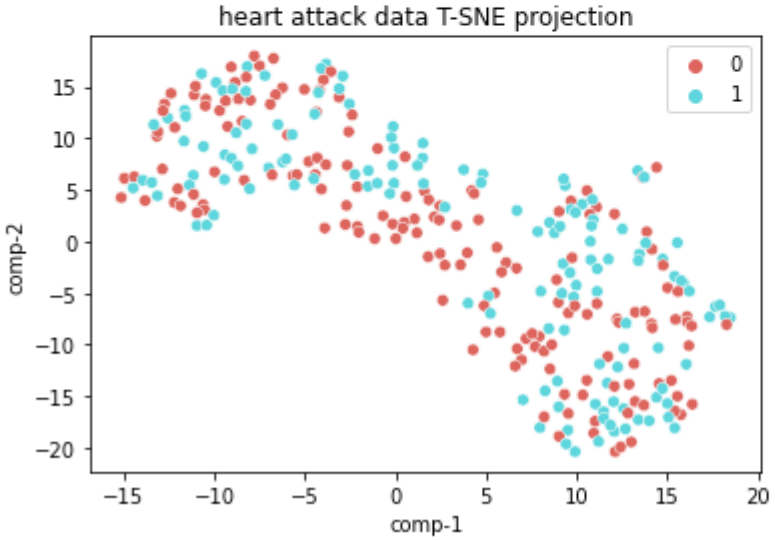


Another solution for PCA with 2 components using seaborn colors

```
x= pd.DataFrame()
x["y"] = y
x["comp-1"] = z[:,0]
x["comp-2"] = z[:,1]

sns.scatterplot(x="comp-1", y="comp-2", hue=x.y.tolist(),
                palette=sns.color_palette("hls",2),
                data=x).set(title="heart attack data T-SNE projection")
```

```
[Text(0.5, 1.0, 'heart attack data T-SNE projection')]
```

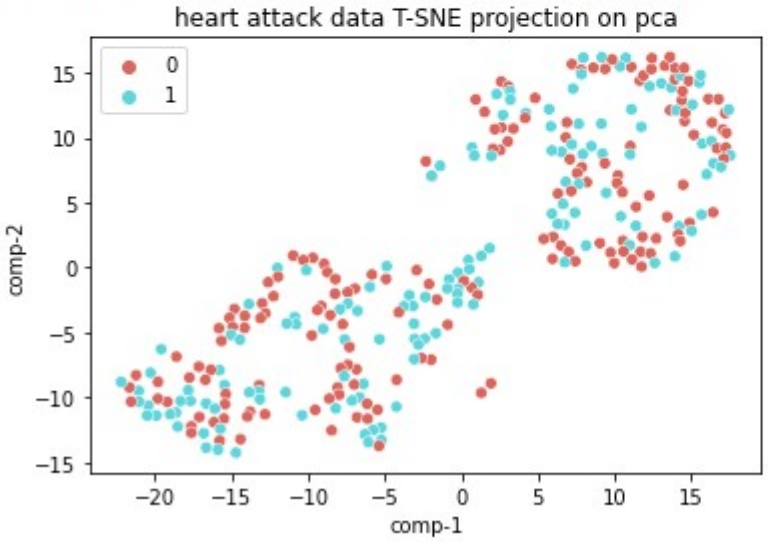


We apply T-SNE on the data and plot it

```
x= pd.DataFrame()
x["y"] = y
x["comp-1"] = z[:,0]
x["comp-2"] = z[:,1]

sns.scatterplot(x="comp-1", y="comp-2", hue=x.y.tolist(),
                palette=sns.color_palette("hls",2),
                data=x).set(title="heart attack data T-SNE projection on pca")
```

```
[Text(0.5, 1.0, 'heart attack data T-SNE projection on pca')]
```



We apply T-SNE on PCA and plot it

Problem 3: Student University Recommendation (Bonus)

Introduction

Build a recommender system based on what is learned in the lecture which helps selecting a university since the criteria are usually confusing. Based on the student dataset and user profile, a list of 10 best universities will be suggested such that it maximizes the chances of a student getting admission into those universities.

Data

Download the data from this link, explanation is also there

Assumptions

You can simplify the data and use other techniques we will learned in the previous lectures to process the features and other parameters. But if you made any assumptions, make sure you list them in the report

At first we replaced the university data set with movies and ratings data set

From amazon which is easier and similar to that we worked on the lecture we applied a recommendation system on it as we took in the last Sunday

It will recommend you the movie according to the ratings of the user and which category you like more using the KNN nearest neighbor

```
# code
import numpy as np
import pandas as pd
import sklearn
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Our main libraries

```
ratings = pd.read_csv("https://s3-us-west-2.amazonaws.com/recommender-tutorial/ratings.csv")
ratings.head()

movies = pd.read_csv("https://s3-us-west-2.amazonaws.com/recommender-tutorial/movies.csv")
movies.head()
```

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

Here is our data set from amazon as I said in the description above

```
n_ratings = len(ratings)
n_movies = len(ratings['movieId'].unique())
n_users = len(ratings['userId'].unique())

print(f"Number of ratings: {n_ratings}")
print(f"Number of unique movieId's: {n_movies}")
print(f"Number of unique users: {n_users}")
print(f"Average ratings per user: {round(n_ratings/n_users, 2)}")
print(f"Average ratings per movie: {round(n_ratings/n_movies, 2)}")
```

Here I calculated number of ratings and movies and users

To take the average of rating per user and per movie which will help us in the recommendation

```
Number of ratings: 100836
Number of unique movieId's: 9724
Number of unique users: 610
Average ratings per user: 165.3
Average ratings per movie: 10.37
```

```
user_freq = ratings[['userId', 'movieId']].groupby('userId').count().reset_index()
user_freq.columns = ['userId', 'n_ratings']
user_freq.head()


# Find Lowest and Highest rated movies:
mean_rating = ratings.groupby('movieId')[['rating']].mean()
# Lowest rated movies
lowest_rated = mean_rating['rating'].idxmin()
movies.loc[movies['movieId'] == lowest_rated]
# Highest rated movies
highest_rated = mean_rating['rating'].idxmax()
movies.loc[movies['movieId'] == highest_rated]
# show number of people who rated movies rated movie highest
ratings[ratings['movieId']==highest_rated]
# show number of people who rated movies rated movie lowest
ratings[ratings['movieId']==lowest_rated]
```

Here I am trying to Find Lowest and Highest rated movies by calculating the mean rating and show the lowest rated movies and the highest ones and number of peoples who rated

	userId	movieId	rating	timestamp
13633	89	3604	0.5	1520408880

the above movies has very low dataset. We will use bayesian average

Now, we created user-item matrix using scipy csr matrix and also used Map Id to help us in doing user mapper and movie mapper to connect the data together and make a recommendation using KNN (we searched online for this)

```
movie_stats = ratings.groupby('movieId')[['rating']].agg(['count', 'mean'])
movie_stats.columns = movie_stats.columns.droplevel()

# Now, we create user-item matrix using scipy csr matrix
from scipy.sparse import csr_matrix

def create_matrix(df):

  N = len(df['userId'].unique())
  M = len(df['movieId'].unique())
  # Map Ids to indices
  user_mapper = dict(zip(np.unique(df["userId"]), list(range(N))))
  movie_mapper = dict(zip(np.unique(df["movieId"]), list(range(M))))

  # Map indices to IDs
  user_inv_mapper = dict(zip(list(range(N)), np.unique(df["userId"])))
  movie_inv_mapper = dict(zip(list(range(M)), np.unique(df["movieId"])))

  user_index = [user_mapper[i] for i in df['userId']]
  movie_index = [movie_mapper[i] for i in df['movieId']]

  X = csr_matrix((df["rating"], (movie_index, user_index)), shape=(M, N))

  return X, user_mapper, movie_mapper, user_inv_mapper, movie_inv_mapper

X, user_mapper, movie_mapper, user_inv_mapper, movie_inv_mapper = create_matrix(ratings)
```

And after this we called nearest neighbor library

```
from sklearn.neighbors import NearestNeighbors
"""
Find similar movies using KNN
"""
def find_similar_movies(movie_id, X, k, metric='cosine', show_distance=False):

  neighbour_ids = []

  movie_ind = movie_mapper[movie_id]
  movie_vec = X[movie_ind]
  k+=1
  kNN = NearestNeighbors(n_neighbors=k, algorithm="brute", metric=metric)
  kNN.fit(X)
  movie_vec = movie_vec.reshape(1,-1)
  neighbour = kNN.kneighbors(movie_vec, return_distance=show_distance)
  for i in range(0,k):
    n = neighbour.item(i)
    neighbour_ids.append(movie_inv_mapper[n])
  neighbour_ids.pop(0)
  return neighbour_ids
```

This method helps us to find the similar movies using KNN

```
movie_titles = dict(zip(movies['movieId'], movies['title']))

movie_id = 3

similar_ids = find_similar_movies(movie_id, X, k=10)
movie_title = movie_titles[movie_id]

print(f"Since you watched {movie_title}")
for i in similar_ids:
  print(movie_titles[i])
```

Now the final step to get recommendations we will find the similar ids and getting the movie title and it will print the recommending movies according to the rating of similar users rated it before and here is the output

```
Since you watched Grumpier Old Men (1995)
Grumpy Old Men (1993)
Striptease (1996)
Nutty Professor, The (1996)
Twister (1996)
Father of the Bride Part II (1995)
Broken Arrow (1996)
Bio-Dome (1996)
Truth About Cats & Dogs, The (1996)
Sabrina (1995)
Birdcage, The (1996)
```

The machine recommended a list of movies for me according to the highest rating that similar users rated it before .