

##PROJECT REPORT

Data set Description

Dataset is of multiple movies and related columns to that movies

It is divided into two sections :

smaller dataset (10k rows)

id,original_language,original_title,release_date,vote_average,vote_count,genre,overview,revenue,runtime
and

larger dataset (1Million rows)

Id, title, genre, userId, rating, gender, age

Columns are self-explanatory and others are :

Avg_voting is out of 10.

Revenue is a movie's collection.

Rating is out of 5*

Users and movies have different IDs.

data quality/integrity/ethics issues

- ❖ The larger dataset has one to many relationship between the columns like :
single movie with its genre and id (1->many) rating, age, gender i.e.one movie was rated by multiple users of any gender and different ratings
- ❖ Incomplete Genre Data: The 'genres' column has only 6020 non-null values out of a total of 1000209 rows. This means that genre information is missing for a large portion of the movies in the dataset. This could limit the usefulness of the dataset for certain types of analysis or recommendations.
- ❖ Biased User Demographics: The dataset only includes information on gender and age of the users, and it is not clear how representative the sample is of the general population. The dataset is not be inclusive of all types of users, which could lead to biased recommendations or conclusions based on the data.
- ❖ Limited Rating Scale: The rating column only ranges from 1 to 5, which could limit the granularity of the analysis and recommendations that can be made from the data.
- ❖ Potential for Fraudulent Ratings: There is no information on how the ratings were obtained from the users, which could leave the dataset open to potential fraudulent ratings or other data quality issues.
- ❖ Privacy Concerns: The dataset contains personal information such as gender and age, which could raise privacy concerns if the data is not handled appropriately.
- ❖ Lack of Contextual Information: The dataset does not provide any contextual information about the movies, such as release date or production budget, which could limit the analysis and insights that can be drawn from the data.
- ❖ While in the smaller data there are missing values: There are missing values in almost all columns, with the number of non-null values ranging from 5336 to 10014. This can be problematic for data analysis since missing data can bias results and decrease the representativeness of the dataset.
- ❖ Inconsistent data types: Some columns contain data of inconsistent data types. For example, The "release_date" column is also of type "object" while it will be more appropriate to represent it as a datetime object.

- ❖ Data integrity issues: There is a data integrity issues in the dataset, such inconsistencies. For example, Unicode formatting utf-8 is not suitable for many different languages hence data is likely to have movie title as garbage column
- ❖ Bias: The dataset may be biased towards certain genres or languages, which can impact the representativeness of the dataset and bias any analysis or modeling based on it.
- ❖ Privacy and ethics: The dataset may contain personal information about individuals involved in the movies, such as the actors or crew, that may need to be protected or anonymized to maintain privacy and ethical considerations.
- ❖ In summary, the dataset has missing values, inconsistent data, data integrity issues, and potential biases and privacy/ethical considerations. These issues need to be addressed and carefully considered in any data analysis or modeling based on this dataset.

Data preparation/wrangling

#Data-wrangling and Data preparation operations :

- ✓ Merging the datasets
- ✓ Groupby() & Merge
- ✓ Simple reading csv files and identifying the types of data
- ✓ Using median and mode in the data columns to fill the null values
- ✓ Categorizing using data slice and dice
- ✓ Dealing with outliers

All outcomes at the end of the report

Data mining fundamentals (25 points)

#Data Mining technique used:

- ✓ Linear Regression
- ✓ Logistic Regression
- ✓ Support Vector Machine to find SVM score.
- ✓ Correlation and Heatmap to relate different parameters of data.
- ✓ KNN for clustering analysis and Silhouette's coefficient

All outcomes at the end of the report

Data Visualization:

Please find all the data visualization techniques performed as per the relevance of the data:

(Loading csv)

```
In [122]: import pandas as pd
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split

desired_width = 40
pd.set_option('display.width', desired_width)
pd.set_option('display.max_columns', 20)
df = pd.read_csv(r'combinedData.csv', low_memory=False)

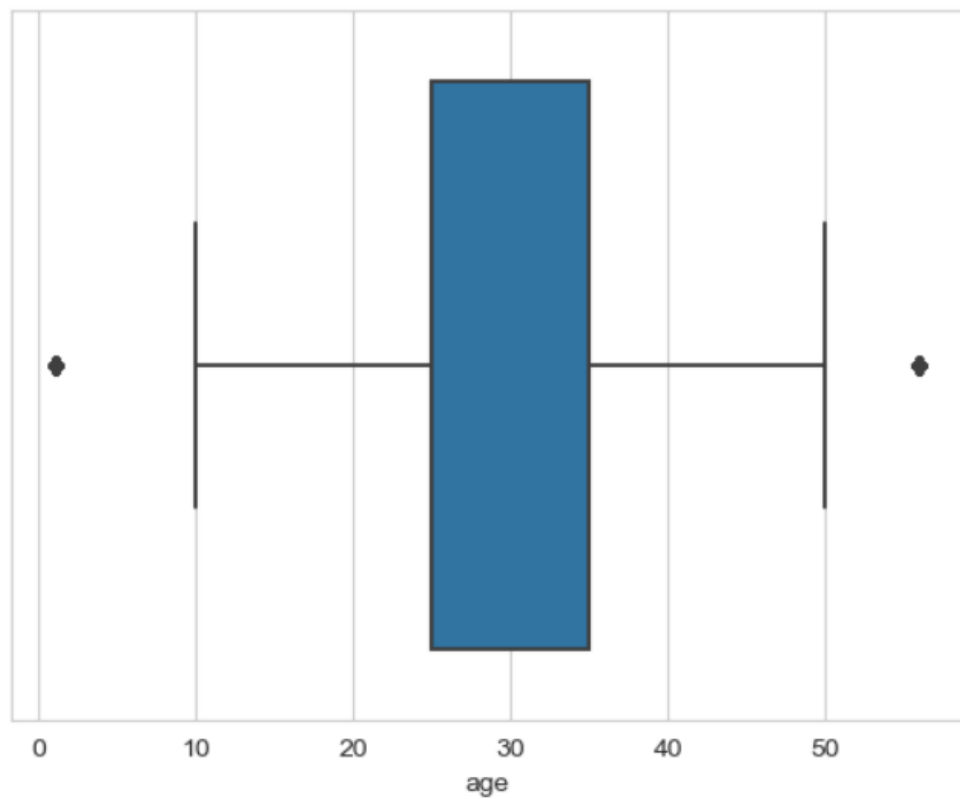
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000209 entries, 0 to 1000208
Data columns (total 7 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   movieId    1000209 non-null   int64
1   title      1000209 non-null   object
2   genres     6020 non-null      object
3   userId     1000209 non-null   int64
4   rating     1000209 non-null   int64
5   gender     1000209 non-null   object
6   age        1000209 non-null   int64
dtypes: int64(4), object(3)
memory usage: 53.4+ MB
```

1. Boxplot

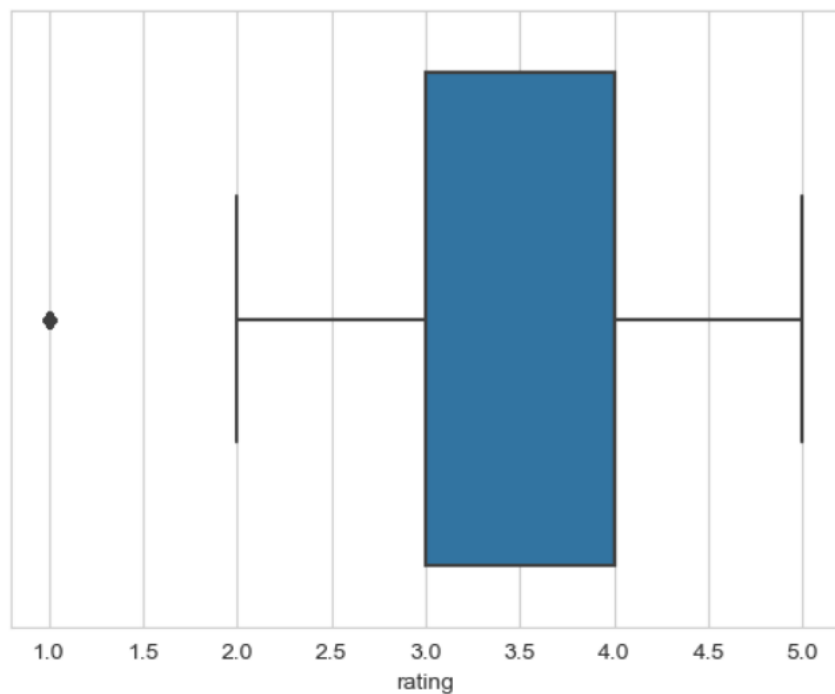
```
In [120]: ▶ sn.boxplot(dataframe.age)  
plt.show()
```

```
C:\Users\adhir\anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: boxplot is deprecated. Use boxplot instead.  
arg: x. From version 0.12, the only valid positional argument will be `data` keyword will result in an error or misinterpretation.  
warnings.warn(
```

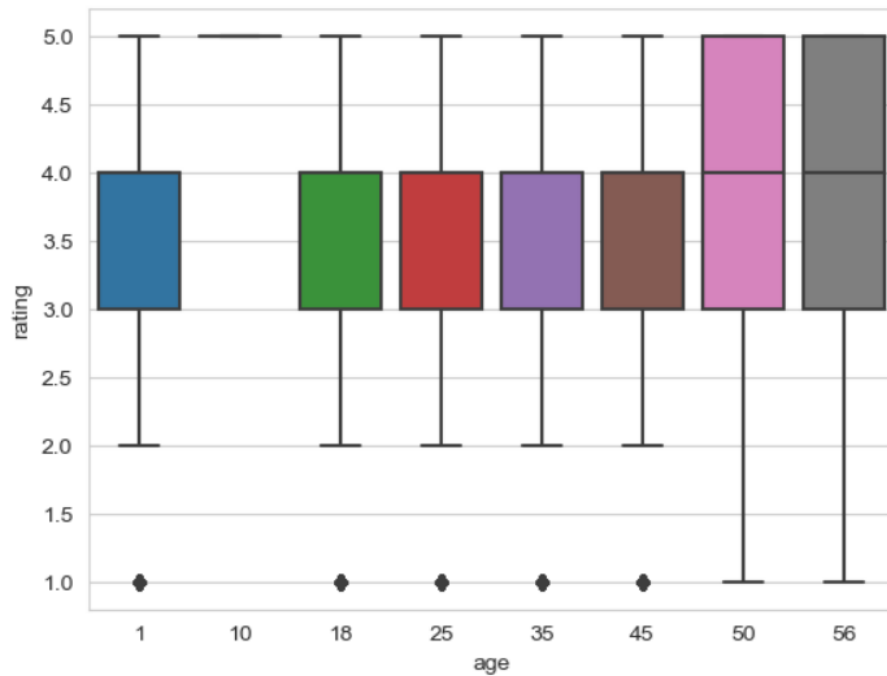


```
In [121]: ▶ sn.boxplot(dataframe.rating)
plt.show()
```

```
C:\Users\adhir\anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
arg: x. From version 0.12, the only valid positional argument will be `data`, and passing
t keyword will result in an error or misinterpretation.
  warnings.warn(
```

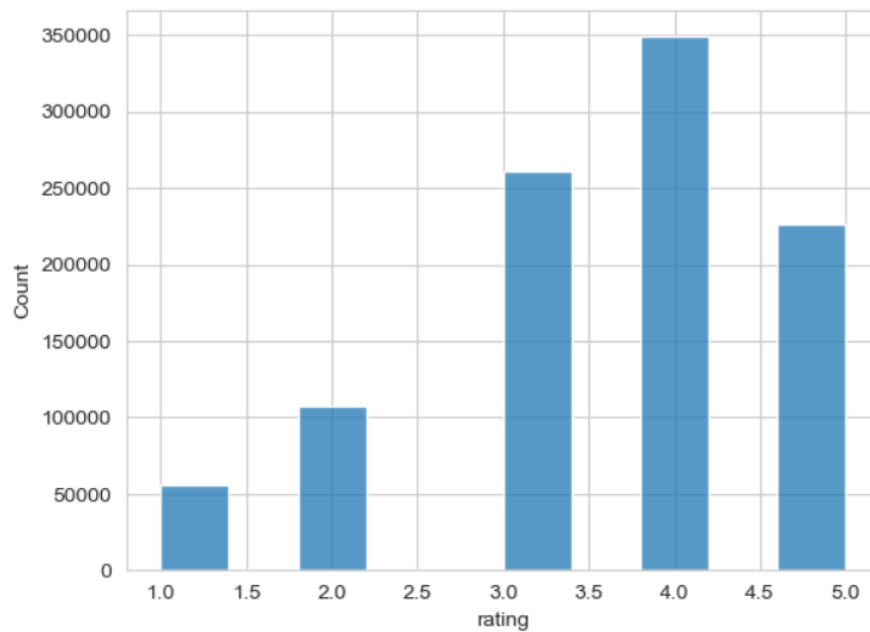


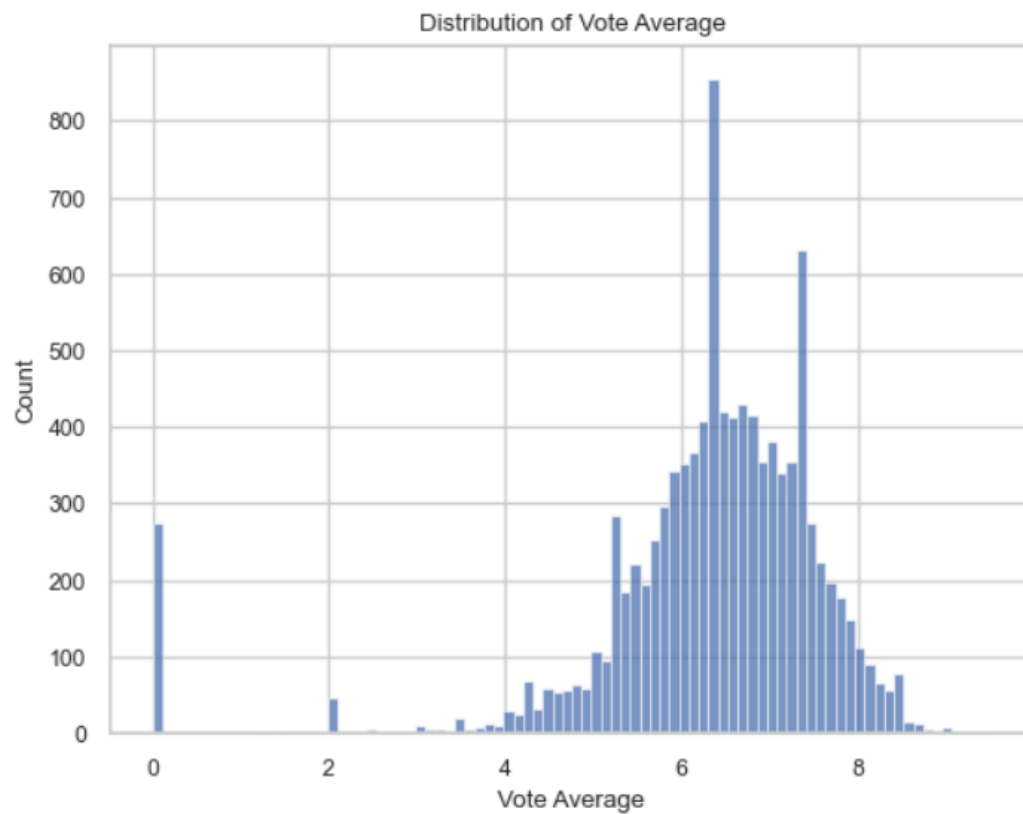
```
In [125]: ▶ sn.boxplot(data=df, x='age', y='rating')
plt.show()
```



2. Histplot

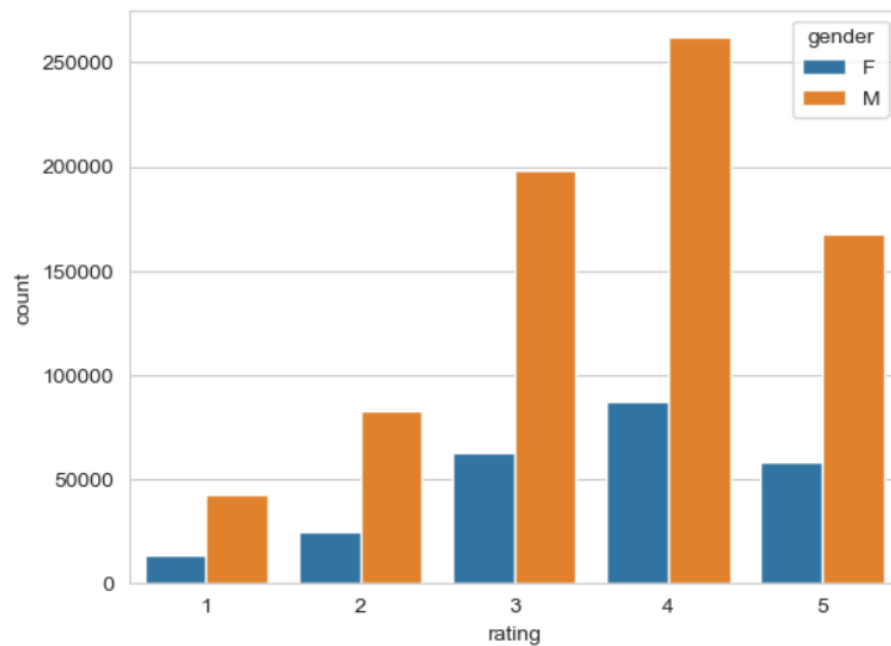
```
In [123]: ▶ sn.histplot(data=df, x='rating', bins=10)
plt.show()
```





3. Countplot

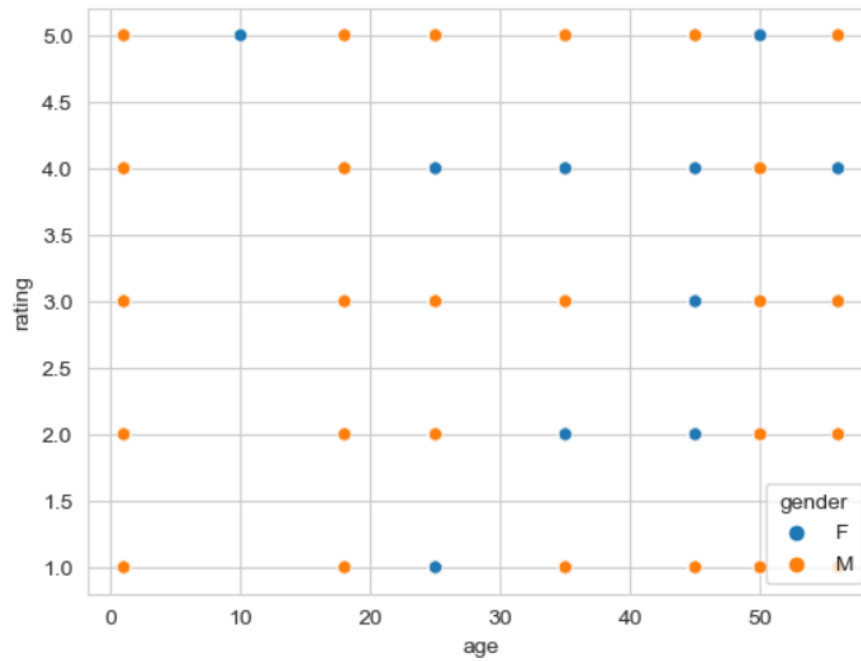
```
In [124]: ▶ sns.countplot(data=df, x='rating', hue='gender')  
plt.show()
```



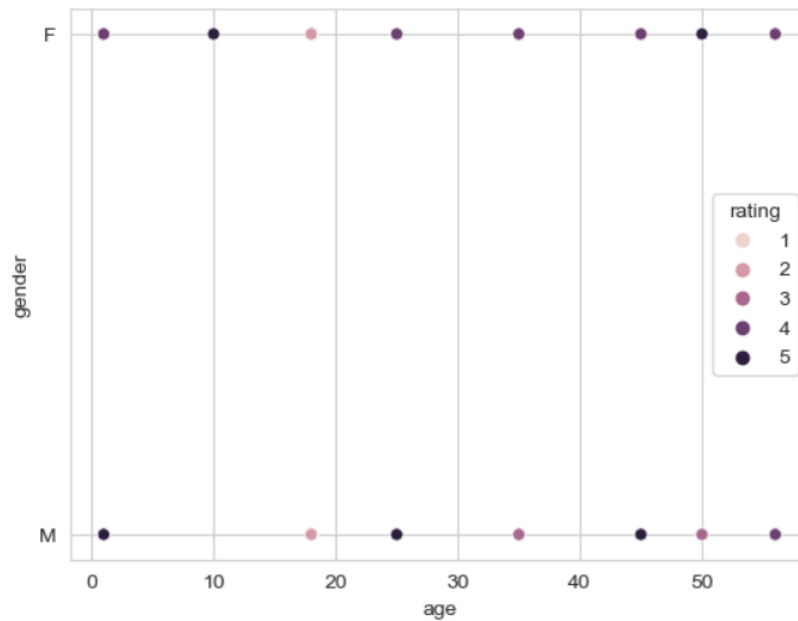
4. Scatterplot

```
In [126]: ▶ sns.scatterplot(data=df, x='age', y='rating', hue='gender')  
plt.show()
```

C:\Users\adhir\anaconda\lib\site-packages\IPython\core\pylabtools.py:151: UserWarning: Creating a plot from a pandas DataFrame is slow with large amounts of data.
fig.canvas.print_figure(bytes_io, **kw)

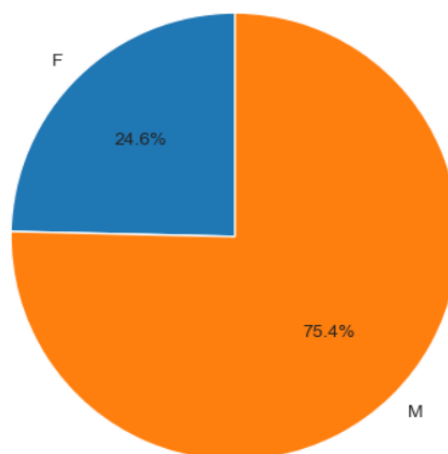



```
In [128]: ▶ sns.scatterplot(data=df, x='age', y='gender', hue='rating')
plt.show()
```



5. Piechart

```
In [127]: ▶ rating_counts = df.groupby('gender')['rating'].count()
sns.set_style("whitegrid")
plt.pie(rating_counts, labels=rating_counts.index, autopct='%1.1f%%', startangle=90)
plt.axis('equal')
plt.show()
```



6. Barchart

```
In [118]: # Filter out rows where runtime is less than 10
df.loc[df['runtime'] >= 10, 'runtime'] = None
df.loc[df['revenue'] < 10000, 'revenue'] = None

# Calculate the mean of the runtime column
mean_runtime = df['runtime'].mean()
mean_rev = df['revenue'].mean()

# Fill any missing values in the runtime column with the mean
df['runtime'] = df['runtime'].fillna(mean_runtime)
df['revenue'] = df['revenue'].fillna(mean_rev)

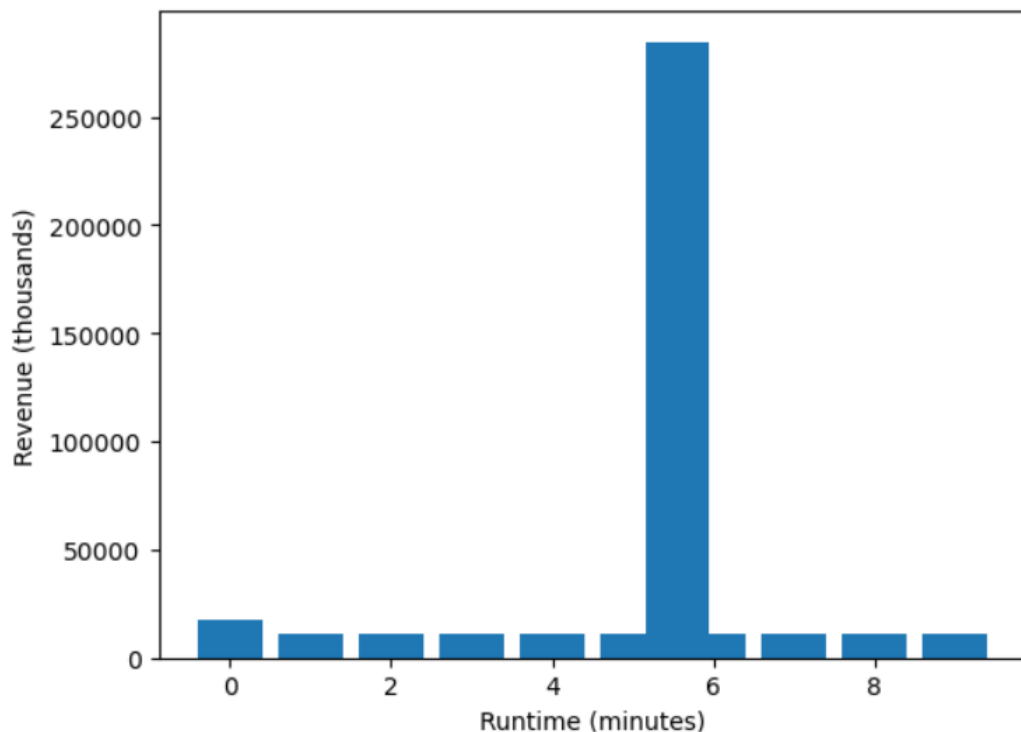
# Convert revenue to thousands
df['revenue'] = df['revenue'] / 10000

# Reset the index of the DataFrame for safer side
# df = df.reset_index(drop=True)

# Create a horizontal bar chart with revenue on the x-axis and runtime on the y-axis
plt.bar(df['runtime'], df['revenue'])

# Set the x-label and y-label of the plot
plt.xlabel('Runtime (minutes)')
plt.ylabel('Revenue (thousands)')

# Show the plot
plt.show()
```



7. Density distribution

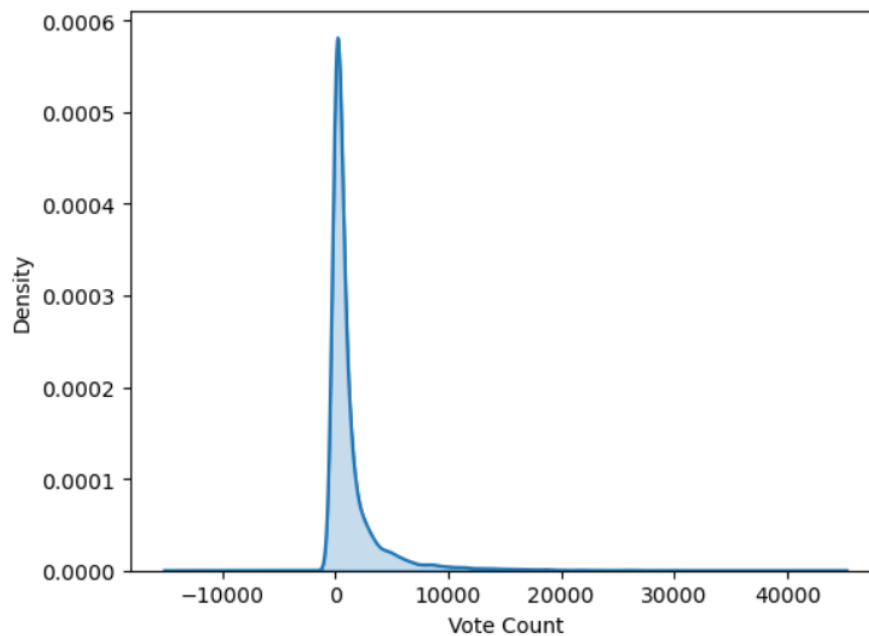
```
In [117]: #density distribution chart of different columns

# Filter out null values from the "vote_count" column
df_filtered = df.dropna(subset=['vote_count'])

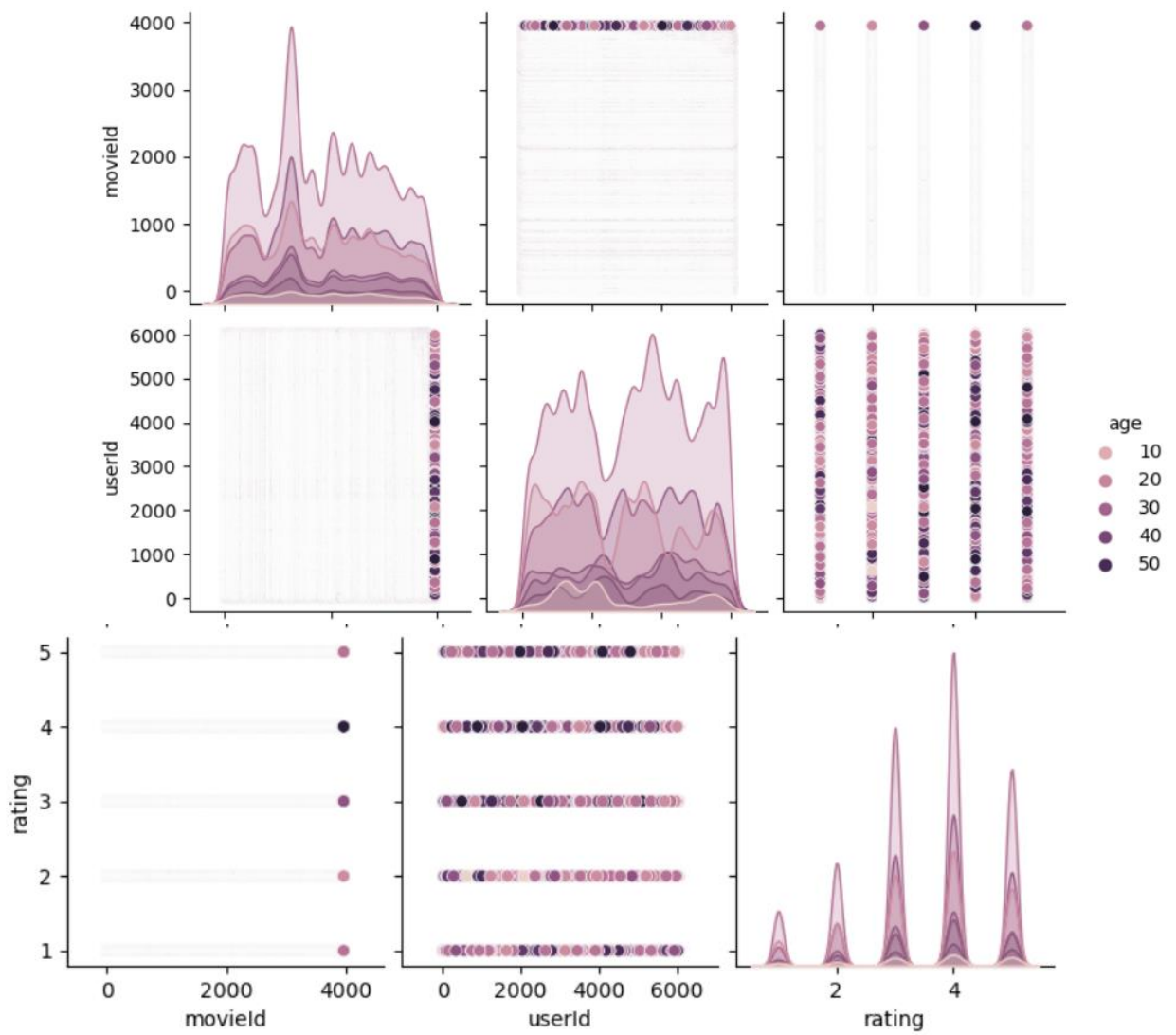
# Create a density distribution chart using Seaborn
sn.kdeplot(df_filtered['vote_count'], shade=True)

# Set the x-label and y-label of the plot
plt.xlabel('Vote Count')
plt.ylabel('Density')

# Show the plot
df.vote_count.plot(kind='kde')
plt.show()
```



8. Pairplots



Outcomes

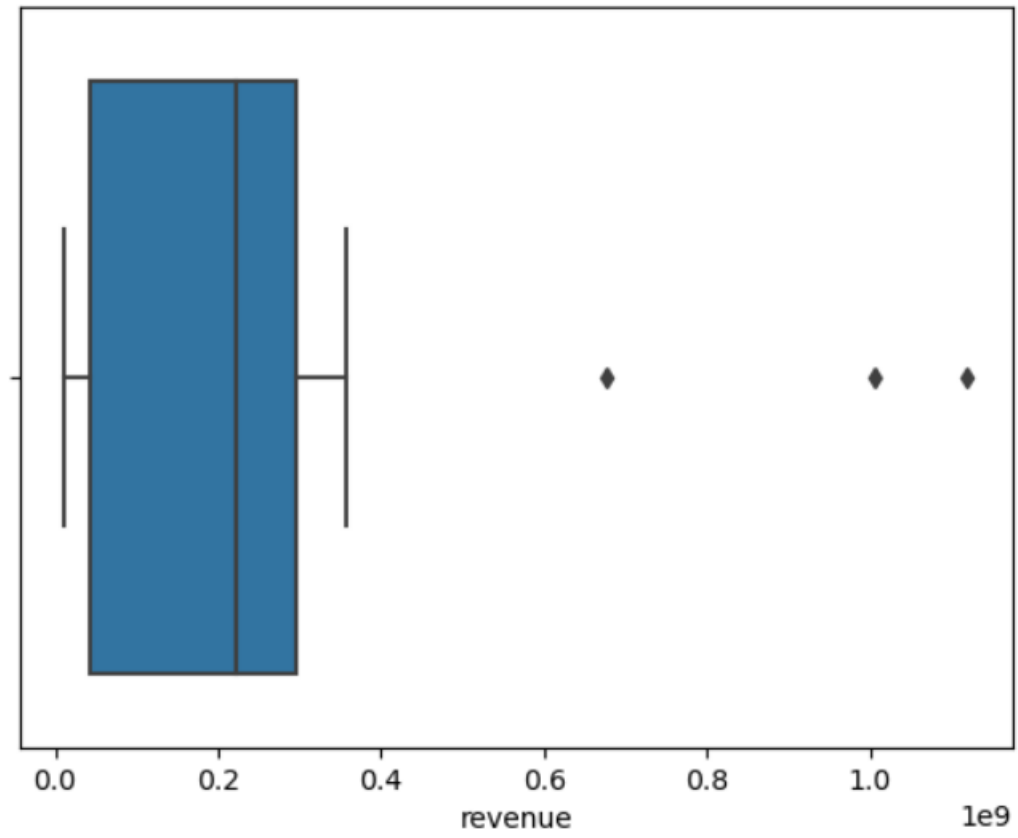
Dealing with outlier :

```
In [126]: ▶ import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import OneHotEncoder, StandardScaler

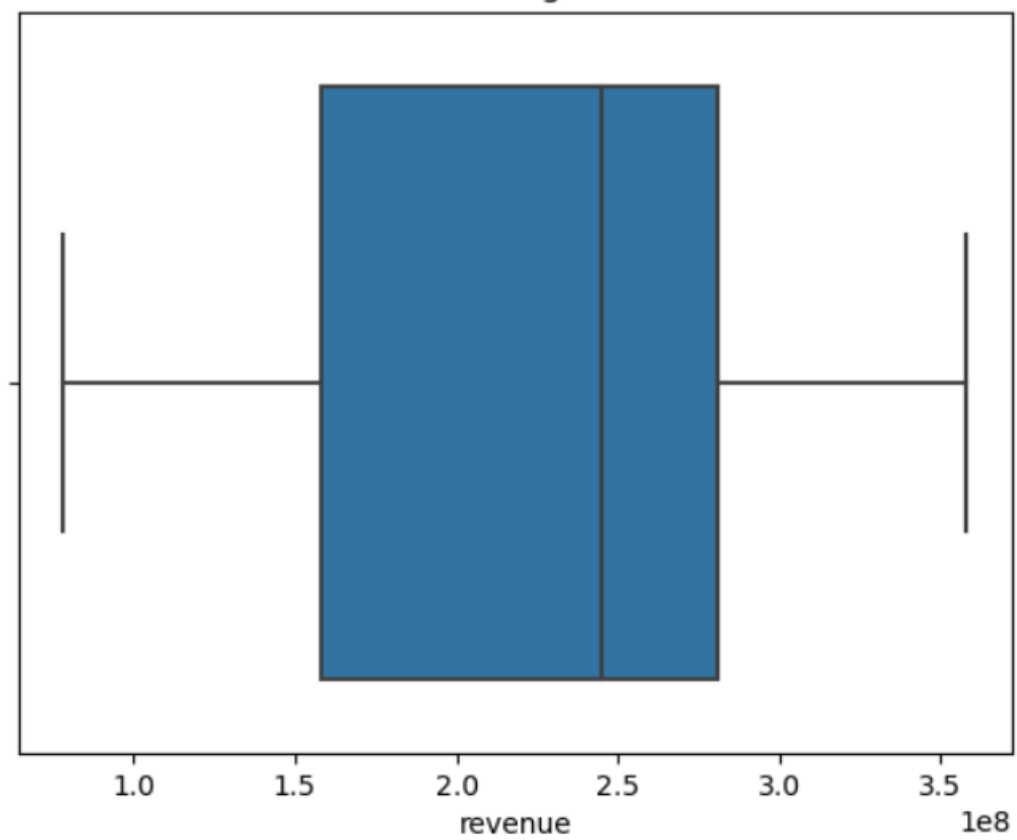
df = pd.read_csv('movies.csv', low_memory=False)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10014 entries, 0 to 10013
Data columns (total 11 columns):
#   Column                Non-Null Count  Dtype
---  -
0   sr_no                 10014 non-null  object
1   id                    10002 non-null  float64
2   original_language     10002 non-null  object
3   original_title        10001 non-null  object
4   release_date          9962 non-null   object
5   vote_average          10000 non-null  float64
6   vote_count            10000 non-null  float64
7   genre                 10000 non-null  object
8   overview              9900 non-null   object
9   revenue               5336 non-null   float64
10  runtime               9762 non-null   float64
dtypes: float64(5), object(6)
memory usage: 860.7+ KB
```

Box Plot1 ~ Dealing with outliers



Box Plot ~ Dealing with outliers



Basic data wrangling operations and their outcomes

```
df['release_date'] = pd.to_datetime(df.release_date) # adds to df as datetime type
print(df.release_date.head(2))
print(df.release_date.dt.year.head(2)) # returns LastSaleDate2 as year format
print(df.release_date.dt.month.head(2)) # returns LastSaleDate2 as month format
print(df.release_date.dt.week.head(2)) # returns LastSaleDate2 as week format
print(df.release_date.dt.day.head(2)) # returns LastSaleDate2 as day format
print(df.release_date.dt.dayofweek.head(2)) # returns as dayofweek format
```

```
0    2021-09-30
1    2021-11-03
Name: release_date, dtype: datetime64[ns]
0    2021.0
1    2021.0
Name: release_date, dtype: float64
0     9.0
1    11.0
Name: release_date, dtype: float64
0    39.0
1    44.0
Name: release_date, dtype: float64
0    30.0
1     3.0
Name: release_date, dtype: float64
0     3.0
1     2.0
Name: release_date, dtype: float64
```



```
In [9]: #-----group by-----
df3 = df.groupby('movieId').age.median().reset_index()
df3.columns = ('movieId', 'movieId_median_value')
df4 = pd.merge(df, df3, on = 'movieId', how = 'left')
print(df4.info())
print(df4.head(2))
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000209 entries, 0 to 1000208
Data columns (total 8 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   movieId                             1000209 non-null  int64
1   title                               1000209 non-null  object
2   genres                              6020 non-null    object
3   userId                              1000209 non-null  int64
4   rating                              1000209 non-null  int64
5   gender                              1000209 non-null  object
6   age                                 1000209 non-null  int64
7   movieId_median_value                1000209 non-null  float64
dtypes: float64(1), int64(4), object(3)
memory usage: 68.7+ MB
None
   movieId      title \
0         1  Toy Story (1995)
1         1  Toy Story (1995)

      genres \
0  Animation|Children's|Comedy
1  Animation|Children's|Comedy

   userId  rating  gender  age \
0         1         5         5  10
```

```
In [14]: ▶ #-----Slicing and Dicing of Data-----
moderateMoviesDF = df[(df.rating>1) & (df.rating<5)]
print('moderateMoviesDF',moderateMoviesDF['movieId'].nunique())

flopMoviesDF = df[(df.rating==1)]
print('flopMoviesDF',flopMoviesDF['movieId'].nunique())

twoStarMoviesDF = df[(df.rating==2)]
print('twoStareMoviesDF',twoStarMoviesDF['movieId'].nunique())

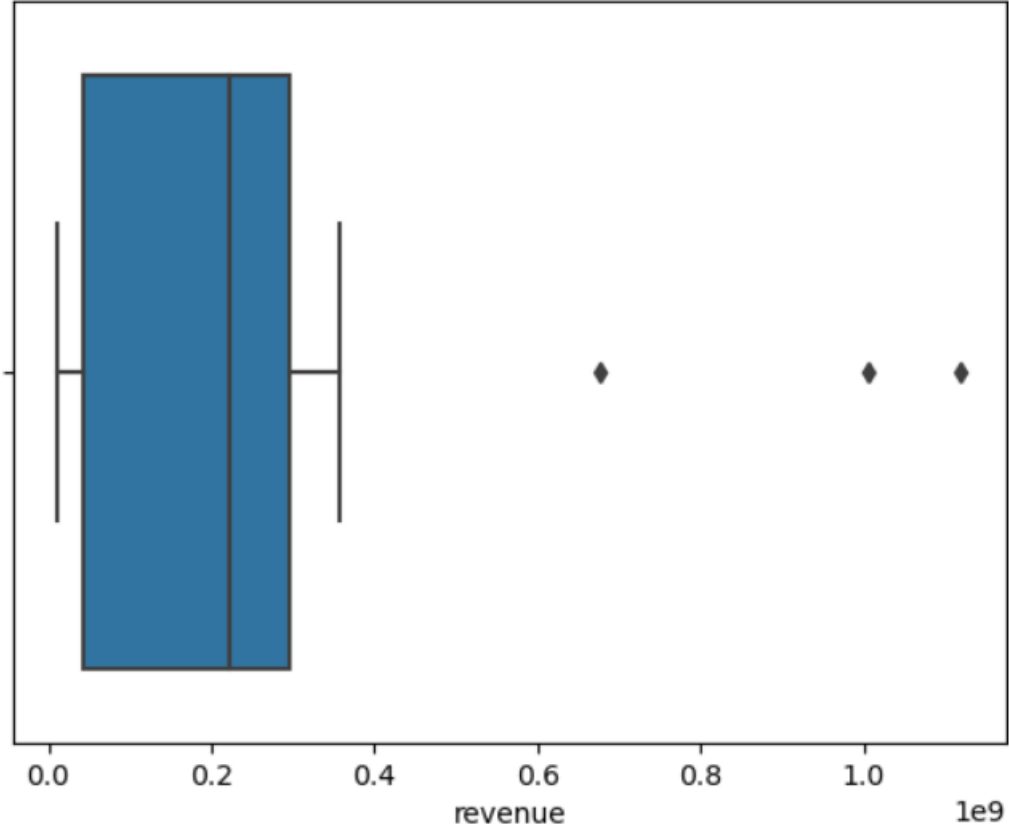
threeStarMoviesDF = df[(df.rating==3)]
print('threeStarMoviesDF',threeStarMoviesDF['movieId'].nunique())

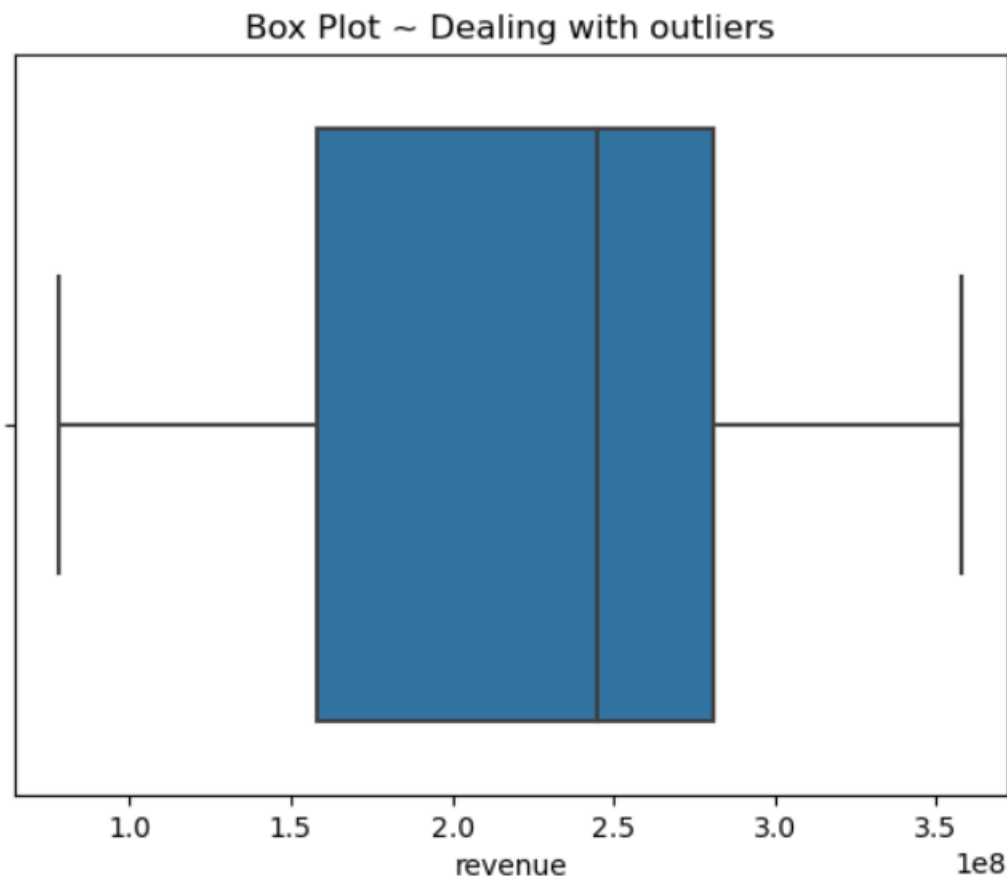
fourStarMoviesDF = df[(df.rating==4)]
print('fourStarMoviesDF',fourStarMoviesDF['movieId'].nunique())

superhitMoviesDF = df[(df.rating==5)]
print('superhitMoviesDF',superhitMoviesDF['movieId'].nunique())

moderateMoviesDF 3663
flopMoviesDF 3274
twoStareMoviesDF 3405
threeStarMoviesDF 3512
fourStarMoviesDF 3489
superhitMoviesDF 3232
```

Box Plot1 ~ Dealing with outliers





Categorizing using slicing

```
In [128]: df['release_date'] = pd.to_datetime(df.release_date) # adds to df as datetime type
print(df.release_date.head(2))
print(df.release_date.dt.year.head(2)) # returns LastSaleDate2 as year format
print(df.release_date.dt.month.head(2)) # returns LastSaleDate2 as month format
print(df.release_date.dt.week.head(2)) # returns LastSaleDate2 as week format
print(df.release_date.dt.day.head(2)) # returns LastSaleDate2 as day format
print(df.release_date.dt.dayofweek.head(2)) # returns as dayofweek format

0    2021-09-30
1    2021-11-03
Name: release_date, dtype: datetime64[ns]
0    2021.0
1    2021.0
Name: release_date, dtype: float64
0     9.0
1    11.0
Name: release_date, dtype: float64
0    39.0
1    44.0
Name: release_date, dtype: float64
0    30.0
1     3.0
Name: release_date, dtype: float64
0     3.0
1     2.0
Name: release_date, dtype: float64
```

Visulization using K-means clustering

```
In [125]: # Drop rows with missing values
data = df.dropna()

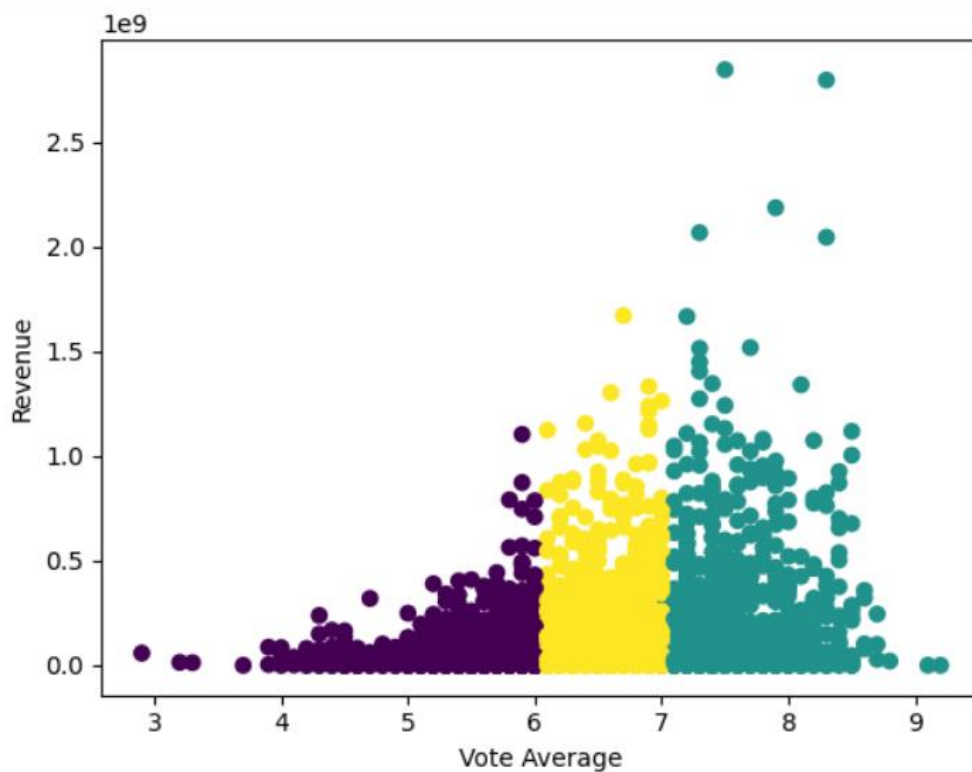
# Select the columns to cluster on
X = data[['genre', 'vote_average']]

# Convert genre column to numerical using one-hot encoding
X = pd.get_dummies(X, columns=['genre'])

# Apply K-Means algorithm with 3 clusters
kmeans = KMeans(n_clusters=3, random_state=42).fit(X)

# Add cluster labels to the data
data['cluster'] = kmeans.labels_

# Visualize the clusters using a scatter plot
plt.scatter(data['vote_average'], data['revenue'], c=data['cluster'])
plt.xlabel('Vote Average')
plt.ylabel('Revenue')
plt.show()
```



Logistic Regression

```
# Data Mining - classification & regression - Logistic Regression
import pandas as pd
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split

desired_width = 400
pd.set_option('display.width', desired_width)          # sets run screen width to 400
pd.set_option('display.max_columns', 20)              # sets run screen column display to 20
df = pd.read_csv(r'movies.csv') # reads zillow file
df.info()

# Drop duplicates
df.drop_duplicates(inplace=True)

# Fill numerical columns with mean
df.fillna(df.mean(), inplace=True)

# Create new column 'gender' with random values 0 or 1
df['gender'] = np.random.randint(0, 2, size=len(df))

# Save the processed data to a new CSV file
df.to_csv('df2.csv', index=False)

df = pd.read_csv('df2.csv', low_memory=False)

# replaces the NaN with 0 to have even 15,000 in all 7 variables
X = df[['vote_average', 'vote_count', 'revenue', 'runtime']] # reduced df as upper case X matrix
y = df.gender
# predictor variable lower case y as array

df['gender'] = df.gender.apply(lambda x: 'male' if x < 1 else 'female')
y2 = df.gender
```

```
log = LogisticRegression()
print(log.fit(X,y2))
print(log.score(X,y2))

X_train, X_test, y2_train, y2_test = train_test_split(X,y2)
print(X_train.shape, y2_train.shape, X_test.shape, y2_test.shape)
print(log.fit(X_train, y2_train))
print(log.score(X_test, y2_test))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10014 entries, 0 to 10013
Data columns (total 11 columns):
#   Column                Non-Null Count  Dtype
---  -
0   sr_no                 10014 non-null  object
1   id                   10002 non-null  float64
2   original_language    10002 non-null  object
3   original_title       10001 non-null  object
4   release_date         9962 non-null   object
5   vote_average         10000 non-null  float64
6   vote_count           10000 non-null  float64
7   genre                10000 non-null  object
8   overview             9900 non-null   object
9   revenue              5336 non-null   float64
10  runtime              9762 non-null   float64
dtypes: float64(5), object(6)
memory usage: 860.7+ KB
```

```
9   revenue              5336 non-null   float64
10  runtime              9762 non-null   float64
dtypes: float64(5), object(6)
memory usage: 860.7+ KB
```

```
C:\Users\adhir\AppData\Local\Temp\ipykernel_19708\100805663.py:16:
reductions (with 'numeric_only=None') is deprecated; in a future v
ns before calling the reduction.
```

```
df.fillna(df.mean(), inplace=True)
```

```
LogisticRegression()
0.49910125823846613
(7510, 4) (7510,) (2504, 4) (2504,)
LogisticRegression()
0.4976038338658147
```

As we can see it has only 49% score so Logistic regression model has an accuracy score of around 50%, which means it is not a good model for predicting gender based on movie ratings, vote counts, revenue, and runtime. It also shows that splitting the dataset into training and testing sets did not improve the accuracy of the model.

Linear Regression

```
import pandas as pd
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import OneHotEncoder, StandardScaler

df = pd.read_csv('movies.csv', low_memory=False)

# Drop duplicates
df.drop_duplicates(inplace=True)

# Fill numerical columns with mean
df.fillna(df.mean(), inplace=True)

# Create new column 'gender' with random values 0 or 1
df['gender'] = np.random.randint(0, 2, size=len(df))

# Save the processed data to a new CSV file
df.to_csv('df2.csv', index=False)

df = pd.read_csv('df2.csv', low_memory=False)

# Drop duplicates
df.drop_duplicates(inplace=True)

# Fill numerical columns with mean
df.fillna(df.mean(), inplace=True)
```



```

# Create new column 'gender' with random values 0 or 1
df['gender'] = np.random.randint(0, 2, size=len(df))

# Save the processed data to a new CSV file
df.to_csv('df2.csv', index=False)

df = pd.read_csv('df2.csv', low_memory=False)

X = df[['revenue', 'vote_count', 'runtime', 'gender']] # reduced
y = df.vote_average #
lg = LinearRegression()
print(lg.fit(X,y))
print(lg.score(X,y))
X_train, X_test, y_train, y_test = train_test_split(X,y)
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
print(lg.fit(X_train, y_train))
print(lg.score(X_test, y_test))

```

```

LinearRegression()
0.10039770471091614
(7510, 4) (7510,) (2504, 4) (2504,)
LinearRegression()
0.0901431961105279

```

As we can see Linear regression has 9% score only and the model may not capture the relationship between the features and the target variable accurately hence its not good method to use for our dataset

Support Vector Machine

```
df = pd.read_csv('df2.csv', low_memory=False)
df2 = df[['vote_average', 'vote_count', 'revenue', 'runtime', 'gender']] # reduced df
df2.fillna(0, inplace=True)
X = df[['vote_average', 'vote_count', 'gender', 'runtime']] # reduced df as upper case X matrix
X1 = X
y = df.revenue
df['revenue'] = df.revenue.apply(lambda x: 'low' if x < 500000 else 'high')
y2 = df.revenue
X_train, X_test, y_train, y_test = train_test_split(X, y)
X1_train, X1_test, y2_train, y2_test = train_test_split(X1, y2)
svr_reg = SVR() # assign svr_reg to the SVR function
svc_class = SVC() # assign svc_class to the SVC function
svr_reg.fit(X_train, y_train) # fit a SVR() model using 11,250 data points
print('svr_score = ', svr_reg.score(X_test, y_test))
svc_class.fit(X1_train, y2_train) # fit a SVC() model using 11,250 data points
print('svc_score = ', svc_class.score(X1_test, y2_test)) #
y2_pred = svc_class.predict(X_test) # assigns y2_pred to SVC prediction of X_test data
print(confusion_matrix(y2_test, y2_pred)) #

C:\Users\adhir\AppData\Local\Temp\ipykernel_19708\4279294242.py:18: FutureWarning: Dropping of nuisance columns in DataFrame
reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError. Select only valid columns
before calling the reduction.
  df.fillna(df.mean(), inplace=True)
C:\Users\adhir\AppData\Local\Temp\ipykernel_19708\4279294242.py:29: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy
  df2.fillna(0, inplace=True)

svr_score = -0.00015045680899761216
svc_score = 0.9672523961661342
[[2422  0]
 [ 82  0]]
```

As we can see the SVR model performs poorly with a negative R-squared score, which indicates that the model does not fit the data well. On the other hand, the SVC model performs well with an accuracy score of approximately 97%. However, when looking at the confusion matrix, we can see that the model is predicting only the low revenue category, with all the high revenue movies being classified as low revenue. Hence not good fit for the dataset

K Means Clustering :

```
df2 = df[['vote_average', 'vote_count', 'revenue', 'runtime', 'gender']] # reduced df
df2.fillna(0, inplace=True)
print(df2.head(2)) # prints top two rows of df3

k_groups = KMeans(n_clusters=5, random_state=0).fit(df2) # separates data set into 5 distinguishable groups
print(k_groups.labels_) # displays k_groups' label (0 to 4) for each row
print(len(k_groups.labels_), df2.shape) # displays rows in k_groups as well as rows, columns in df3
print(k_groups.cluster_centers_) # displays averages of the seven columns for each cluster centroid [0, 1, 2, 3, 4]
print(k_groups.cluster_centers_[0]) # displays averages for each of the seven columns in the cluster centroid [0]

df2['cluster'] = k_groups.labels_ # add a new column to df3 called 'cluster', the k-group #
print(df2.groupby('cluster').min()) # display the means of the seven columns of data frame df3

from sklearn.metrics import silhouette_score # coefficient score where higher is better, 0 = cluster overlap
df3 = df2.drop('cluster', axis = 1) # create a new data frame df4 that dropped the cluster column
# for loop to determine optimum K groups
for i in range(3, 10): # for loop to determine best number of K clusters between 3 and 10
    k_groups = KMeans(n_clusters = i).fit(df3) # K clusters must have atleast 2 clusters
    labels = k_groups.labels_
    print('K Groups = ', i, 'Silhouette Coefficient = ', silhouette_score(df3, labels)) # displays i and coefficient
# End of Data Mining - Cluster Analysis
```

```

    vote_average  vote_count      revenue  runtime  gender
0           6.8      1736.0  424000000.0    97.0      1
1           7.1       622.0  165000000.0   157.0      0
[4 2 4 ... 2 0 2]
10014 (10014, 5)
[[6.55161489e+00 1.02770747e+03 2.08579286e+07 1.05501999e+02
  5.14610889e-01]
 [7.17000000e+00 1.17805133e+04 8.47558199e+08 1.28333333e+02
  4.46666667e-01]
 [6.09958804e+00 8.11218653e+02 1.12109670e+08 9.68070186e+01
  5.09572901e-01]
 [7.52727273e+00 1.72090909e+04 1.92505603e+09 1.44363636e+02
  7.27272727e-01]
 [6.80957230e+00 5.93574134e+03 3.57066574e+08 1.15892057e+02
  4.94908350e-01]]
[6.55161489e+00 1.02770747e+03 2.08579286e+07 1.05501999e+02
  5.14610889e-01]
    vote_average  vote_count      revenue  runtime  gender
cluster
0           2.9         2.0  1.000000e+01     0.0      0
1           5.8        176.0  6.038731e+08    87.0      0
2           0.0         0.0  6.667352e+07     0.0      0
3           6.7       7740.0  1.405404e+09   103.0      0
4           4.3        50.0  2.348019e+08    70.0      0
K Groups = 3 Silhouette Coefficient = 0.7951706220961635
K Groups = 4 Silhouette Coefficient = 0.7612152107989971
K Groups = 5 Silhouette Coefficient = 0.7717650818145939
K Groups = 6 Silhouette Coefficient = 0.7820720131682729
K Groups = 7 Silhouette Coefficient = 0.7897677118640188
K Groups = 8 Silhouette Coefficient = 0.7957006226865337
K Groups = 9 Silhouette Coefficient = 0.7959303423889847

```

Result indicates that the Silhouette Coefficient for 9 clusters is 0.7959, which is quite high and suggests that the clustering can be well-defined and the movies are grouped together in a meaningful way based on their features. This could be useful for further analysis or for making recommendations based on the features of the movies in each cluster.

Here is one example of clustering with $k=9$

