Question 1

In [1]:
```
#pip install pyppeteer
```

In [2]:
```
#Import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [3]:
```
df=pd.read_csv('dataset/imdb_top_1000.csv')
```

In [4]:
```
df.head()
```

Out[4]:

	Poster_Link	Series_Title	Released_Year	Certificate	Runtime	Genre	IMDB_Ratir
0	https://m.media-amazon.com/images/M/MV5BMDFkYT...	The Shawshank Redemption	1994	A	142 min	Drama	9
1	https://m.media-amazon.com/images/M/MV5BM2MyNj...	The Godfather	1972	A	175 min	Crime, Drama	9
2	https://m.media-amazon.com/images/M/MV5BMTMxNT...	The Dark Knight	2008	UA	152 min	Action, Crime, Drama	9
3	https://m.media-amazon.com/images/M/MV5BMWMwMG...	The Godfather: Part II	1974	A	202 min	Crime, Drama	9
4	https://m.media-amazon.com/images/M/MV5BMWU4N2...	12 Angry Men	1957	U	96 min	Crime, Drama	9

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 16 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Poster_Link    1000 non-null   object
 1   Series_Title   1000 non-null   object
 2   Released_Year  1000 non-null   object
 3   Certificate    899 non-null    object
 4   Runtime        1000 non-null   object
 5   Genre          1000 non-null   object
 6   IMDB_Rating    1000 non-null   float64
 7   Overview       1000 non-null   object
 8   Meta_score     843 non-null    float64
 9   Director       1000 non-null   object
 10  Star1          1000 non-null   object
 11  Star2          1000 non-null   object
 12  Star3          1000 non-null   object
 13  Star4          1000 non-null   object
 14  No_of_Votes    1000 non-null   int64
 15  Gross          831 non-null    object
dtypes: float64(2), int64(1), object(13)
memory usage: 125.1+ KB
```

In [6]:
```
#lets see what is the percentage of null values in total dataset
df.isnull().sum()/len(df)
```

Out[6]:
```
Poster_Link      0.000
Series_Title     0.000
Released_Year    0.000
Certificate      0.101
Runtime          0.000
Genre            0.000
IMDB_Rating      0.000
Overview         0.000
Meta_score       0.157
Director         0.000
Star1            0.000
Star2            0.000
Star3            0.000
Star4            0.000
No_of_Votes      0.000
Gross            0.169
dtype: float64
```

As the null values are max 17% in total I will replace with other values instead of dropping them

In [7]:
```
#fill the null values with the Meta_score mean
df['Meta_score']=df['Meta_score'].fillna(df['Meta_score'].mean())
```

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 16 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Poster_Link    1000 non-null   object
 1   Series_Title   1000 non-null   object
 2   Released_Year  1000 non-null   object
 3   Certificate    899 non-null    object
 4   Runtime        1000 non-null   object
 5   Genre          1000 non-null   object
 6   IMDB_Rating    1000 non-null   float64
 7   Overview       1000 non-null   object
 8   Meta_score     1000 non-null   float64
 9   Director       1000 non-null   object
 10  Star1          1000 non-null   object
 11  Star2          1000 non-null   object
 12  Star3          1000 non-null   object
 13  Star4          1000 non-null   object
 14  No_of_Votes    1000 non-null   int64
 15  Gross          831 non-null    object
dtypes: float64(2), int64(1), object(13)
memory usage: 125.1+ KB
```

In [9]:
```
#change Gross from object to numeric numbers
df['Gross']=df['Gross'].str.replace(',','')
df["Gross"] = df["Gross"].apply(pd.to_numeric)
```

In [10]:
```
#fill the missing values with mean of the Gross column
df['Gross'].fillna(df['Gross'].mean(),inplace=True)
```

In [11]:
```
df['Gross'].head()
```

Out[11]:
```
0     28341469.0
1    134966411.0
2    534858444.0
3     57300000.0
4      4360000.0
Name: Gross, dtype: float64
```

In [12]:
```
df.describe()
```

Out[12]:

	IMDB_Rating	Meta_score	No_of_Votes	Gross
count	1000.000000	1000.00000	1.000000e+03	1.000000e+03
mean	7.949300	77.97153	2.736929e+05	6.803475e+07
std	0.275491	11.36206	3.273727e+05	1.000371e+08
min	7.600000	28.00000	2.508800e+04	1.305000e+03
25%	7.700000	72.00000	5.552625e+04	5.012919e+06
50%	7.900000	77.97153	1.385485e+05	4.238945e+07
75%	8.100000	85.25000	3.741612e+05	6.803475e+07
max	9.300000	100.00000	2.343110e+06	9.366622e+08

In [13]:
```
df['Certificate'].is_unique
```

Out[13]: False

In [14]:
```
df['Certificate'].nunique()
```

Out[14]: 16

In [15]:
```
#Drop unnecessary columns
df.drop(columns=['Poster_Link','Series_Title','Certificate','Overview'],inplace=True)
```

In [16]:
```
#Remaining columns
df.columns
```

Out[16]:
```
Index(['Released_Year', 'Runtime', 'Genre', 'IMDB_Rating', 'Meta_score',
       'Director', 'Star1', 'Star2', 'Star3', 'Star4', 'No_of_Votes', 'Gross'],
      dtype='object')
```

In [17]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Released_Year  1000 non-null   object
 1   Runtime        1000 non-null   object
 2   Genre          1000 non-null   object
 3   IMDB_Rating    1000 non-null   float64
 4   Meta_score     1000 non-null   float64
 5   Director       1000 non-null   object
 6   Star1          1000 non-null   object
 7   Star2          1000 non-null   object
 8   Star3          1000 non-null   object
 9   Star4          1000 non-null   object
 10  No_of_Votes    1000 non-null   int64
 11  Gross          1000 non-null   float64
dtypes: float64(3), int64(1), object(8)
memory usage: 93.9+ KB
```

In [18]:
```
#Drop PG value from Released_Year
df = df.drop(df[df['Released_Year'] == 'PG'].index)
```

In [19]:
```
#Change Released_Year datatype to integer
df['Released_Year']=df['Released_Year'].astype('int')
```

In [20]:
```
#Remove commas and change Runtime feature datatype to integer
df['Runtime']=df['Runtime'].str.replace('min','')
df['Runtime']=df['Runtime'].astype('int')
```

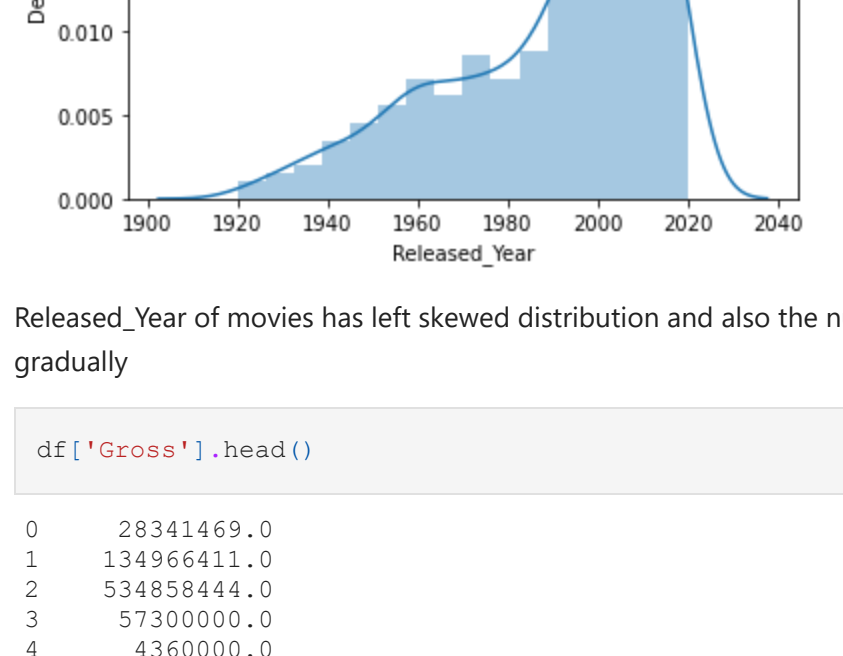
In [21]:
```
#After cleaning dataset there are no null values left
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 999 entries, 0 to 999
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Released_Year  999 non-null    int32
 1   Runtime        999 non-null    int32
 2   Genre          999 non-null    object
 3   IMDB_Rating    999 non-null    float64
 4   Meta_score     999 non-null    float64
 5   Director       999 non-null    object
 6   Star1          999 non-null    object
 7   Star2          999 non-null    object
 8   Star3          999 non-null    object
 9   Star4          999 non-null    object
 10  No_of_Votes    999 non-null    int64
 11  Gross          999 non-null    float64
dtypes: float64(3), int32(2), int64(1), object(6)
memory usage: 93.7+ KB
```

In [22]:
```
#some visualization techniques- distribution of Runtime
sns.distplot(df['Runtime'])
```

```
D:\Emrullah\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[22]: <AxesSubplot:xlabel='Runtime', ylabel='Density'>



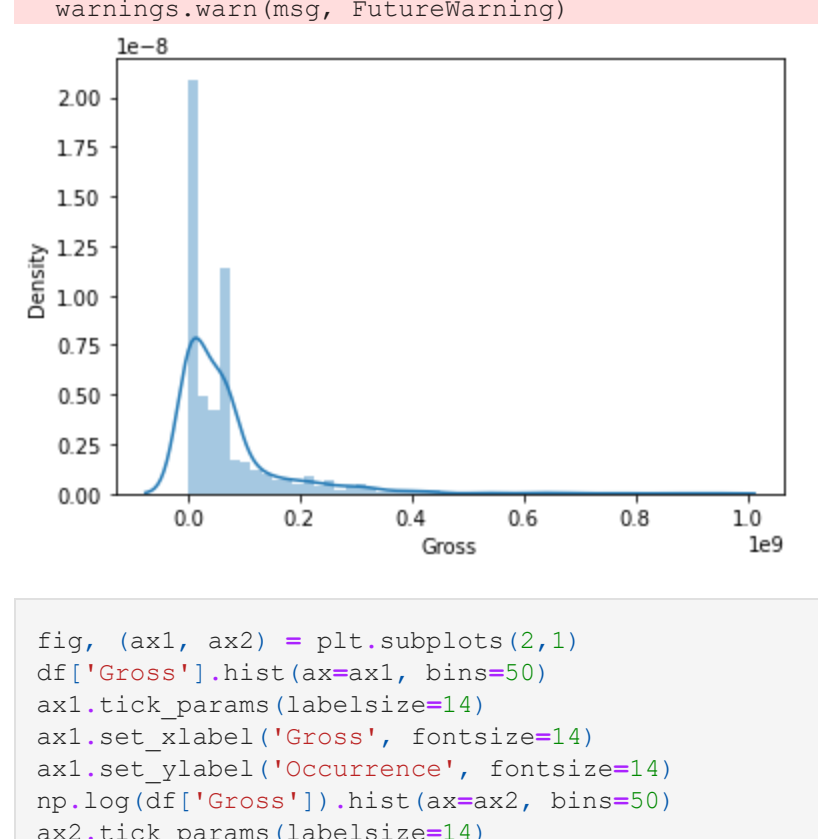
To be able to see the distribution of Runtime of movies distplot is used. The Runtime has Gaussian distribution and the films between 100m and 150m have the highest Density.

In [23]:
```
# to see when earlier and latest movies are produced
df['Released_Year'].min(),df['Released_Year'].max()
```

Out[23]: (1920, 2020)

In [24]:
```
sns.distplot(df['Released_Year']);
```

```
D:\Emrullah\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



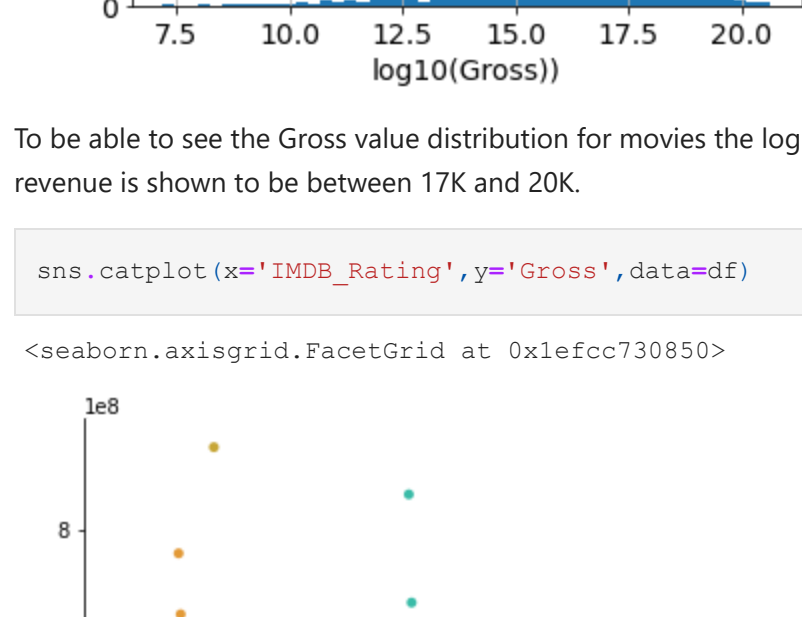
Released_Year of movies has left skewed distribution and also the number of movies are increasing gradually

In [25]:
```
df['Gross'].head()
```

Out[25]:
```
0     28341469.0
1    134966411.0
2    534858444.0
3     57300000.0
4      4360000.0
Name: Gross, dtype: float64
```

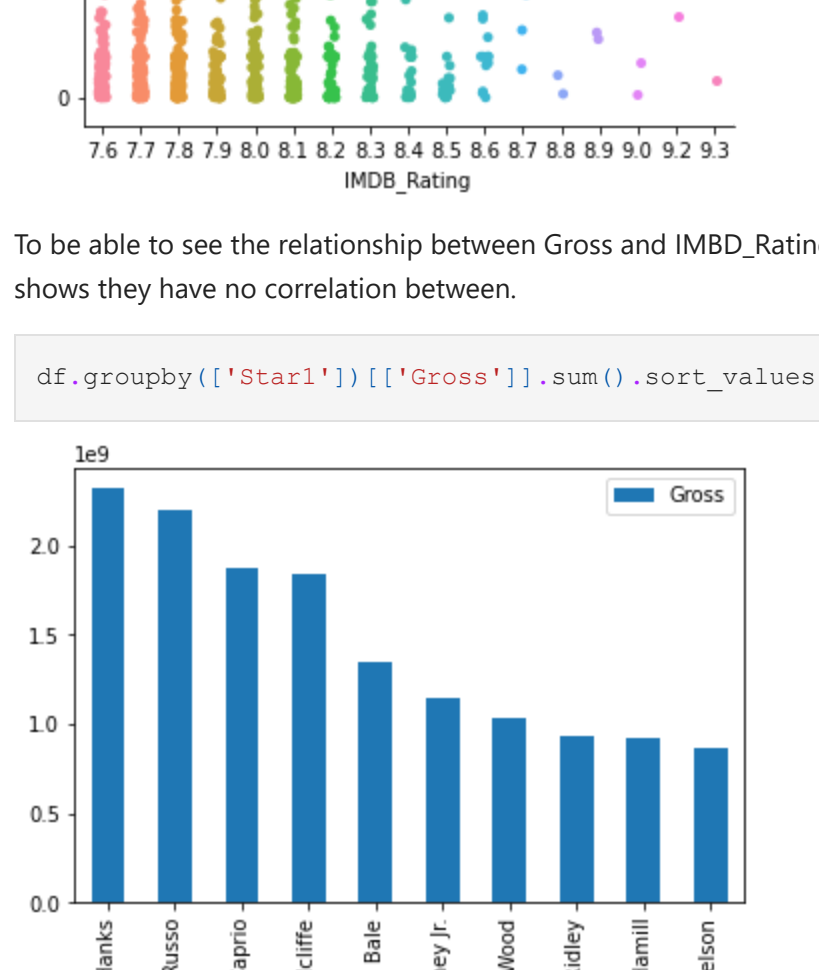
In [26]:
```
sns.distplot(df['Gross']);
```

```
D:\Emrullah\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [27]:
```
fig, (ax1, ax2) = plt.subplots(2,1)
df['Gross'].hist(ax=ax1, bins=50)
ax1.tick_params(labelsize=14)
ax1.set_xlabel('Gross', fontsize=14)
ax1.set_ylabel('Occurrence', fontsize=14)
np.log(df['Gross']).hist(ax=ax2, bins=50)
ax2.tick_params(labelsize=14)
ax2.set_xlabel('log10(Gross))', fontsize=14)
ax2.set_ylabel('Occurrence', fontsize=14)
```

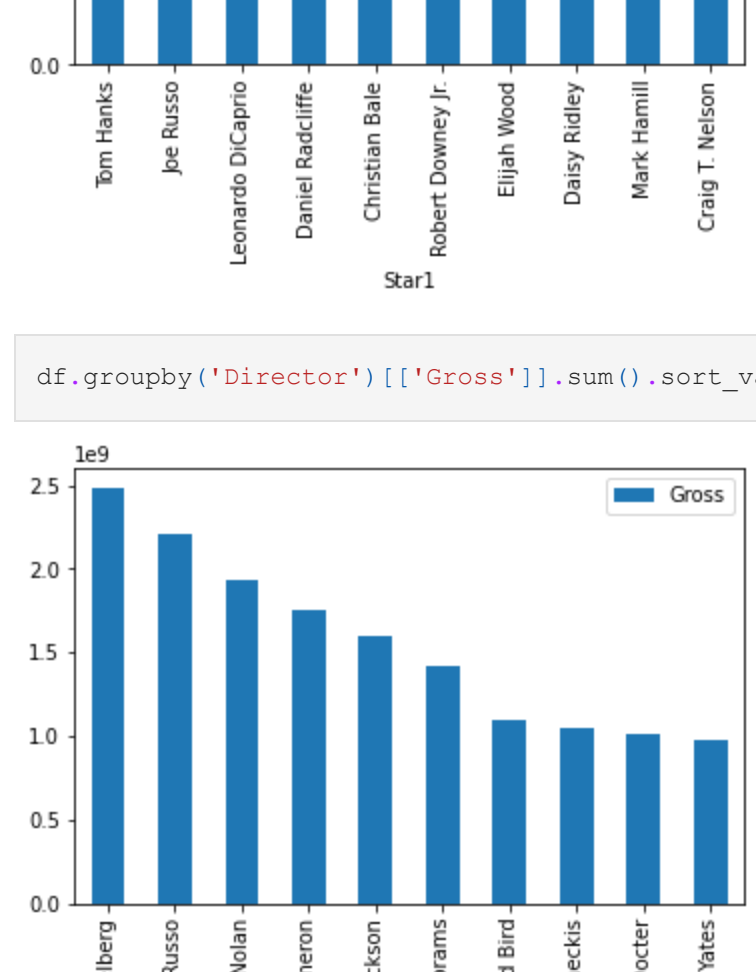
Out[27]: Text(0, 0.5, 'Occurrence')



To be able to see the Gross value distribution for movies the log transformation is applied and the highest revenue is shown to be between 17K and 20K.

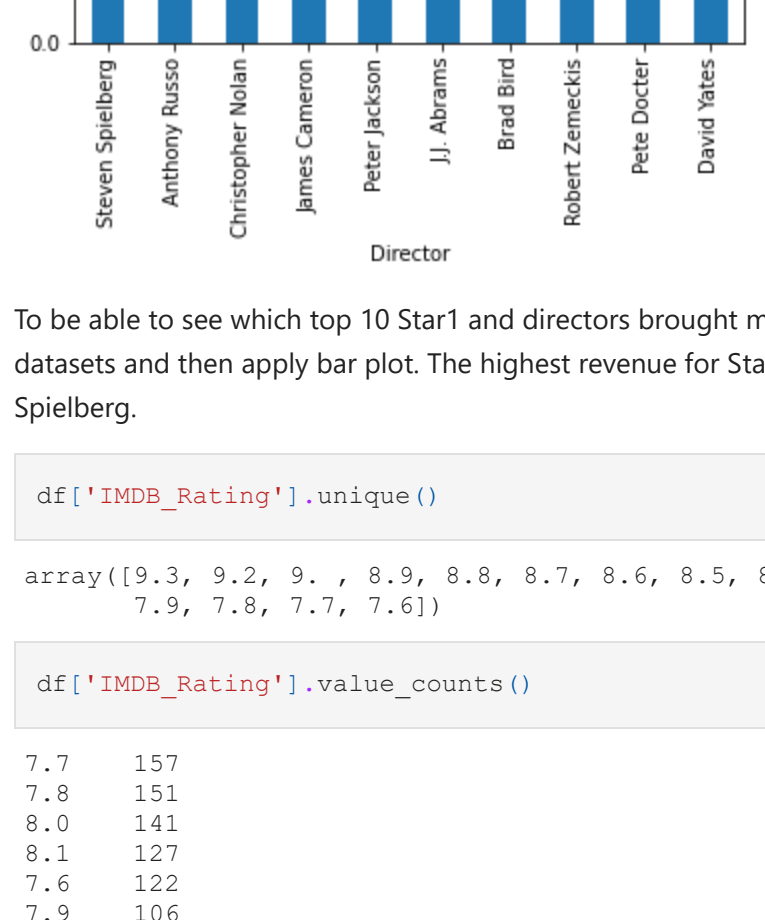
In [28]:
```
sns.catplot(x='IMDB_Rating',y='Gross',data=df)
```

Out[28]: <seaborn.axisgrid.FacetGrid at 0x1efcc730850>

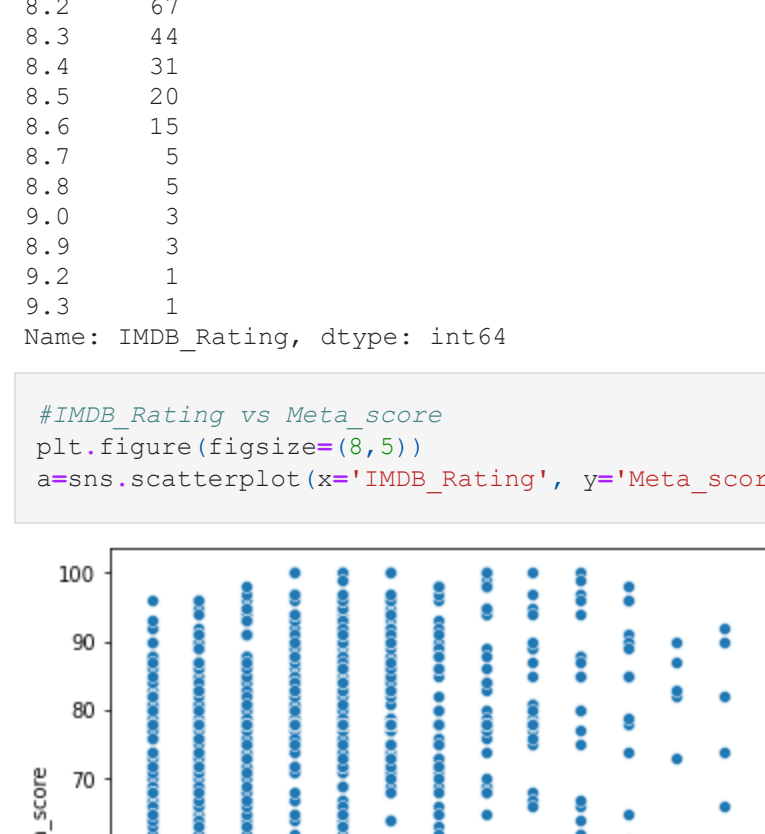


To be able to see the relationship between Gross and IMBD_Rating features the catplot is applied, the plot shows they have no correlation between.

In [29]:
```
df.groupby(['Star1'])[['Gross']].sum().sort_values(by='Gross',ascending=False)[:10].p
```



In [30]:
```
df.groupby('Director')[['Gross']].sum().sort_values(by='Gross',ascending=False)[:10].p
```



To be able to see which top 10 Star1 and directors brought most revenue I use groupby function to create datasets and then apply bar plot. The highest revenue for Star1 is Tom Hanks and for director is Steven Spielberg.

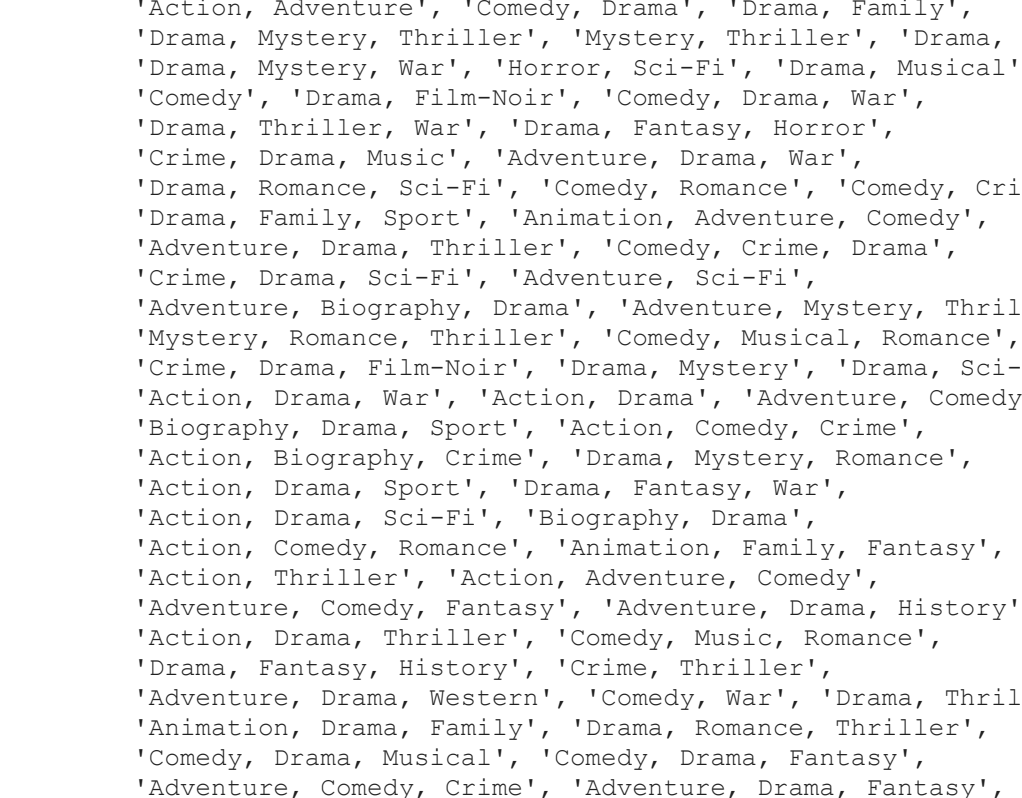
In [31]:
```
df['IMDB_Rating'].unique()
```

Out[31]:
```
array([9.3, 9.2, 9. , 8.9, 8.8, 8.7, 8.6, 8.5, 8.4, 8.3, 8.2, 8.1, 8. ,
       7.9, 7.8, 7.7, 7.6])
```

In [32]:
```
df['IMDB_Rating'].value_counts()
```

Out[32]:
```
7.7    157
7.8    151
8.0    141
8.1    127
7.6    122
7.9    106
8.2     67
8.3     44
8.4     31
8.5     20
8.6     15
8.7      5
8.8      5
9.0      3
8.9      3
9.2      1
9.3      1
Name: IMDB_Rating, dtype: int64
```

In [33]:
```
#IMDB_Rating vs Meta_score
plt.figure(figsize=(8,5))
a=sns.scatterplot(x='IMDB_Rating', y='Meta_score',data=df, ci= None)
```

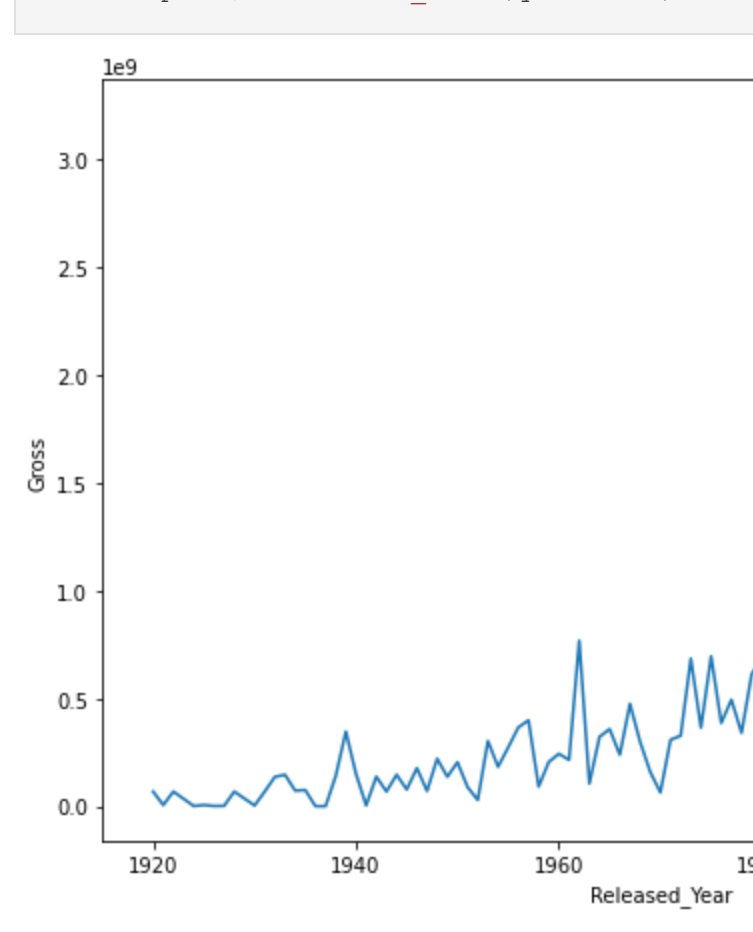


Also there is no relationship between IMDB_Rating and Meta_score

In [34]:
```
#See Genre column unique values
df['Genre'].unique()
```

Out[34]:
```
array(['Drama', 'Crime, Drama', 'Action, Crime, Drama',
       'Action, Adventure, Drama', 'Biography, Drama, History',
       'Action, Adventure, Sci-Fi', 'Drama, Romance', 'Western',
       'Action, Sci-Fi', 'Biography, Crime, Drama',
       'Action, Adventure, Fantasy', 'Comedy, Drama, Thriller',
       'Adventure, Drama, Sci-Fi', 'Animation, Adventure, Family',
       'Drama, War', 'Crime, Drama, Fantasy', 'Comedy, Drama, Romance',
       'Crime, Drama, Mystery', 'Crime, Drama, Thriller',
       'Action, Drama, Mystery', 'Drama, Family, Fantasy', 'Drama, Music',
       'Biography, Comedy, Drama', 'Drama, Mystery, Sci-Fi',
       'Biography, Drama, Music', 'Crime, Mystery, Thriller',
       'Animation, Adventure, Drama', 'Animation, Drama, War',
       'Adventure, Comedy, Sci-Fi', 'Horror, Mystery, Thriller',
       'Drama, Romance, War', 'Comedy, Drama, Family',
       'Animation, Drama, Fantasy', 'Action, Biography, Drama',
       'Animation, Action, Adventure', 'Drama, Western',
       'Action, Adventure', 'Comedy, Drama', 'Drama, Family',
       'Drama, Mystery, Thriller', 'Mystery, Thriller', 'Drama, Horror',
       'Drama, Mystery, War', 'Horror, Sci-Fi', 'Drama, Musical',
       'Comedy', 'Drama, Film-Noir', 'Comedy, Drama, War',
       'Drama, Thriller, War', 'Drama, Fantasy, Horror',
       'Crime, Drama, Music', 'Adventure, Drama, War',
       'Drama, Romance, Sci-Fi', 'Comedy, Romance', 'Comedy, Crime',
       'Drama, Family, Sport', 'Animation, Adventure, Comedy',
       'Adventure, Drama, Thriller', 'Comedy, Crime, Drama',
       'Crime, Drama, Sci-Fi', 'Adventure, Sci-Fi',
       'Adventure, Biography, Drama', 'Adventure, Mystery, Thriller',
       'Mystery, Romance, Thriller', 'Comedy, Musical, Romance',
       'Crime, Drama, Film-Noir', 'Drama, Mystery', 'Drama, Sci-Fi',
       'Action, Drama, War', 'Action, Drama', 'Adventure, Comedy, Drama',
       'Biography, Drama, Sport', 'Action, Comedy, Crime',
       'Action, Biography, Crime', 'Drama, Mystery, Romance',
       'Action, Drama, Sport', 'Drama, Fantasy, War',
       'Action, Drama, Sci-Fi', 'Biography, Drama',
       'Action, Comedy, Romance', 'Animation, Family, Fantasy',
       'Action, Thriller', 'Action, Adventure, Comedy',
       'Adventure, Comedy, Fantasy', 'Adventure, Drama, History',
       'Action, Drama, Thriller', 'Comedy, Music, Romance',
       'Drama, Fantasy, History', 'Crime, Thriller',
       'Adventure, Drama, Western', 'Comedy, War', 'Drama, Thriller',
       'Animation, Drama, Family', 'Drama, Romance, Thriller',
       'Comedy, Drama, Musical', 'Comedy, Drama, Fantasy',
       'Adventure, Comedy, Crime', 'Adventure, Drama, Fantasy',
       'Biography, Drama, Family', 'Animation, Comedy, Drama',
       'Drama, Sport', 'Animation, Action, Drama',
       'Adventure, Drama, Musical', 'Drama, Music, Romance',
       'Comedy, Crime, Romance', 'Comedy, Crime, Sport',
       'Drama, History, Romance', 'Adventure, Drama',
       'Animation, Adventure, Fantasy', 'Horror, Mystery, Sci-Fi',
       'Drama, Fantasy, Music', 'Action, Sci-Fi, Thriller',
       'Drama, Fantasy', 'Drama, Horror, Thriller', 'Drama, History',
       'Film-Noir, Mystery, Thriller', 'Fantasy, Horror, Mystery',
       'Action, Crime, Thriller', 'Comedy, Drama, Music',
       'Biography, Drama, Thriller', 'Animation, Biography, Drama',
       'Action, Mystery, Thriller', 'Crime, Drama, Romance',
       'Action, Adventure, Thriller', 'Crime, Drama, Musical',
       'Animation, Crime, Mystery', 'Action, Crime, Comedy',
       'Mystery, Sci-Fi, Thriller', 'Animation, Action, Crime',
       'Comedy, Fantasy, Romance', 'Drama, History, Thriller',
       'Animation, Action, Sci-Fi', 'Adventure, Family, Fantasy',
       'Drama, Fantasy, Romance', 'Drama, History, War',
       'Adventure, Thriller', 'Horror', 'Drama, Family, Musical',
       'Action, Drama, Western', 'Crime, Drama, Horror',
       'Drama, Film-Noir, Mystery', 'Comedy, Crime, Thriller',
       'Film-Noir, Mystery', 'Comedy, Crime, Mystery',
       'Drama, Fantasy, Mystery', 'Comedy, Horror',
       'Action, Adventure, History', 'Drama, Music, Mystery',
       'Comedy, Music', 'Comedy, Family', 'Drama, Music, Musical',
       'Action, Adventure, Horror', 'Action, Adventure, Biography',
       'Biography, Drama, War', 'Action, Adventure, Western',
       'Horror, Thriller', 'Comedy, Mystery, Romance',
       'Drama, Thriller, Western', 'Crime, Film-Noir, Thriller',
       'Drama, Film-Noir, Romance', 'Crime, Film-Noir, Mystery',
       'Action, Adventure, Romance', 'Comedy, Music, Musical',
       'Adventure, Horror, Sci-Fi', 'Fantasy, Horror',
       'Action, Drama, History', 'Adventure, Comedy, Family',
       'Animation, Biography, Crime', 'Adventure, Biography, Crime',
       'Adventure, Fantasy', 'Drama, History, Mystery',
       'Action, Comedy, Mystery', 'Adventure, Drama, Romance',
       'Drama, Sci-Fi, Thriller', 'Crime, Drama, History',
       'Action, Comedy, Fantasy', 'Family, Sci-Fi',
       'Adventure, History, War', 'Animation, Sci-Fi',
       'Family, Fantasy, Musical', 'Thriller', 'Comedy, Family, Fantasy',
       'Adventure, Comedy, Film-Noir', 'Film-Noir, Thriller',
       'Comedy, Family, Romance', 'Drama, Horror, Sci-Fi',
       'Comedy, Musical, War', 'Biography, Drama, Romance',
       'Drama, History, Music', 'Animation, Action, Fantasy',
       'Animation, Comedy, Fantasy', 'Comedy, Western',
       'Action, Adventure, War', 'Drama, Horror, Mystery',
       'Animation, Comedy, Crime', 'Action, Adventure, Crime',
       'Action, Adventure, Mystery', 'Action, Adventure, Family',
       'Action, Crime, Mystery', 'Animation, Drama, Romance',
       'Drama, War, Western', 'Adventure, Comedy, War'], dtype=object)
```

In [35]:
```
#Dsitobution of top 10 Genres in barplot
df['Genre'].value_counts()[:10].plot.bar();
```



In [36]:
```
#Find unique values for Released_Year with Gross and append to the lists
x =[]
y =[]

for i in df['Released_Year'].unique():
    y.append(df[df['Released_Year']==i]['Gross'].sum())
    x.append(i)
```

In [37]:
```
#Put 2 lists into DataFrame with zip function
df2=pd.DataFrame(list(zip(x,y)),columns=['Released_Year','Gross']).sort_values('Gross
```

In [38]:
```
plt.figure(figsize=(10,7))
sns.lineplot(x='Released_Year',y='Gross',data=df2);
```

For the yearly increase in Gross, I use lineplot function. It shows Revenue is increasing gradually despite some fluctuations.

In [39]:
```
df['Gross'].head()
```

Out[39]:
```
0     28341469.0
1    134966411.0
2    534858444.0
3     57300000.0
4      4360000.0
Name: Gross, dtype: float64
```

In [40]:
```
min_value = df['Gross'].min()
max_value = df['Gross'].max()
print(min_value)
print(max_value)
```

```
1305.0
936662225.0
```

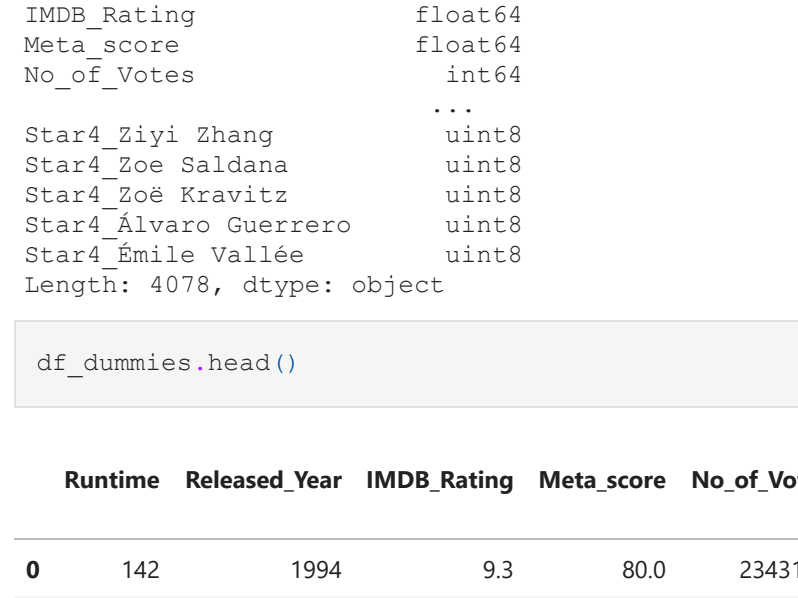
In [41]:
```
import numpy as np
bins = np.linspace(min_value,max_value,4)
bins
```

Out[41]:
```
array([1.30500000e+03, 3.12221612e+08, 6.24441918e+08, 9.36662225e+08])
```

In [42]:
```
labels = ['Average', 'High', 'Very High']
```

In [43]:
```
df['Gross_bin'] = pd.cut(df['Gross'], bins=bins, labels=labels, include_lowest=True)
```

In [44]:
```
plt.hist(df['Gross_bin'], bins=3)
plt.show()
```



I use Gross column to convert continuous data to ordinary categorical features for classification model. I use Min() and Max() of the data and 'linspace' function. Then after that plot the new categories and it shows that most of the movies' Gross fall into Average category.

In [45]:
```
df.columns
```

Out[45]:
```
Index(['Released_Year', 'Runtime', 'Genre', 'IMDB_Rating', 'Meta_score',
       'Director', 'Star1', 'Star2', 'Star3', 'Star4', 'No_of_Votes', 'Gross',
       'Gross_bin'],
      dtype='object')
```

In [46]:
```
obj=['Genre','Director','Star1', 'Star2', 'Star3', 'Star4']
```

In [47]:
```
cols_to_std = ['Runtime','Released_Year','IMDB_Rating','Meta_score','No_of_Votes']
```

In [48]:
```
dummies = pd.get_dummies(df[obj],drop_first=True)
df_dummies = pd.concat([df[cols_to_std],dummies],axis=1)
df_dummies.dtypes
```

Out[48]:
```
Runtime                     int32
Released_Year               int32
IMDB_Rating               float64
Meta_score                float64
No_of_Votes                 int64
                            ...
Star4_Ziyi Zhang            uint8
Star4_Zoe Saldana           uint8
Star4_Zoë Kravitz           uint8
Star4_Álvaro Guerrero       uint8
Star4_Émile Vallée          uint8
Length: 4078, dtype: object
```

In [49]:
```
df_dummies.head()
```

Out[49]:

	Runtime	Released_Year	IMDB_Rating	Meta_score	No_of_Votes	Genre_Action, Adventure, Biography	Genre_Action, Adventure, Comedy	Genre_Action Adventure Crime
0	142	1994	9.3	80.0	2343110	0	0	(
1	175	1972	9.2	100.0	1620367	0	0	(
2	152	2008	9.0	84.0	2303232	0	0	(
3	202	1974	9.0	90.0	1129952	0	0	(
4	96	1957	9.0	96.0	689845	0	0	(

5 rows × 4078 columns

In [50]:
```
df_dummies.columns
```

Out[50]:
```
Index(['Runtime', 'Released_Year', 'IMDB_Rating', 'Meta_score', 'No_of_Votes',
       'Genre_Action, Adventure, Biography', 'Genre_Action, Adventure, Comedy',
       'Genre_Action, Adventure, Crime', 'Genre_Action, Adventure, Drama',
       'Genre_Action, Adventure, Family',
       ...
       'Star4_Yûnosuke Itô', 'Star4_Zac Mattoon O'Brien', 'Star4_Zach Grenier',
       'Star4_Zarah Jane McKenzie', 'Star4_Zeppo Marx', 'Star4_Ziyi Zhang',
       'Star4_Zoe Saldana', 'Star4_Zoë Kravitz', 'Star4_Álvaro Guerrero',
       'Star4_Émile Vallée'],
      dtype='object', length=4078)
```

I create obj list with non numeric columns and use Dummies function to convert them to numeric values and then concatenate with cols_to_std list which already have numeric values and after that I have 4078 columns.

In [51]:
```
from sklearn.preprocessing import OrdinalEncoder

ord_enc = OrdinalEncoder()
df["Gross_bin"] = ord_enc.fit_transform(df[["Gross_bin"]])
df['Gross_bin'].head(11)
```

Out[51]:
```
0     0.0
1     0.0
2     1.0
3     0.0
4     0.0
5     1.0
6     0.0
7     0.0
8     0.0
9     0.0
10    1.0
Name: Gross_bin, dtype: float64
```

Used sklearn OrdinalEncoder function to convert target variable into categorical data

In [52]:
```
#split data in X and y variable
X=df_dummies
y=df['Gross_bin']
```

In [53]:
```
## import the library for train, test and splitting of teh data set
from sklearn.model_selection import train_test_split
# split X and y into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.20)
```

In [54]:
```
# check the shape of X_train, X_test,y_train and y_test
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[54]: ((799, 4078), (200, 4078), (799,), (200,))

In [55]:
```
#import library to scale dataset
from sklearn.preprocessing import StandardScaler

# Create an initialise an object sc by calling a method StandardScaler()
sc = StandardScaler()

# Train the model by calling a method fit_transform
X_train = sc.fit_transform(X_train)

# Store the transformed data into X_test
X_test = sc.transform(X_test)
```

In [56]:
```
# Fitting Naive Bayes Classification to the Training set with linear kernel
from sklearn.naive_bayes import GaussianNB


# Create an initialise an object 'nvclassifier' by calling a method 'GaussianNB()'
nvclassifier = GaussianNB()

# Call fit() method for training the dataset
nvclassifier.fit(X_train, y_train)
```

Out[56]: GaussianNB()

In [57]:

```python
# Predicting the Test set results
y_pred = nvclassifier.predict(X_test)

# Display the predicted values
print(y_pred)
```

```
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 2. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 2. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 2. 0. 0. 2. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 1. 0. 0. 1. 0. 0. 2. 0. 0. 0. 0. 2. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0.
 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 2. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 2. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0.]
```

In [58]:

```python
# Check the actual and predicted value side by side
y_compare = np.vstack((y_test,y_pred)).T
# actual value on the left side and predicted value on the right hand side
# printing the top 10 values
y_compare[:10,:]
```

Out[58]:

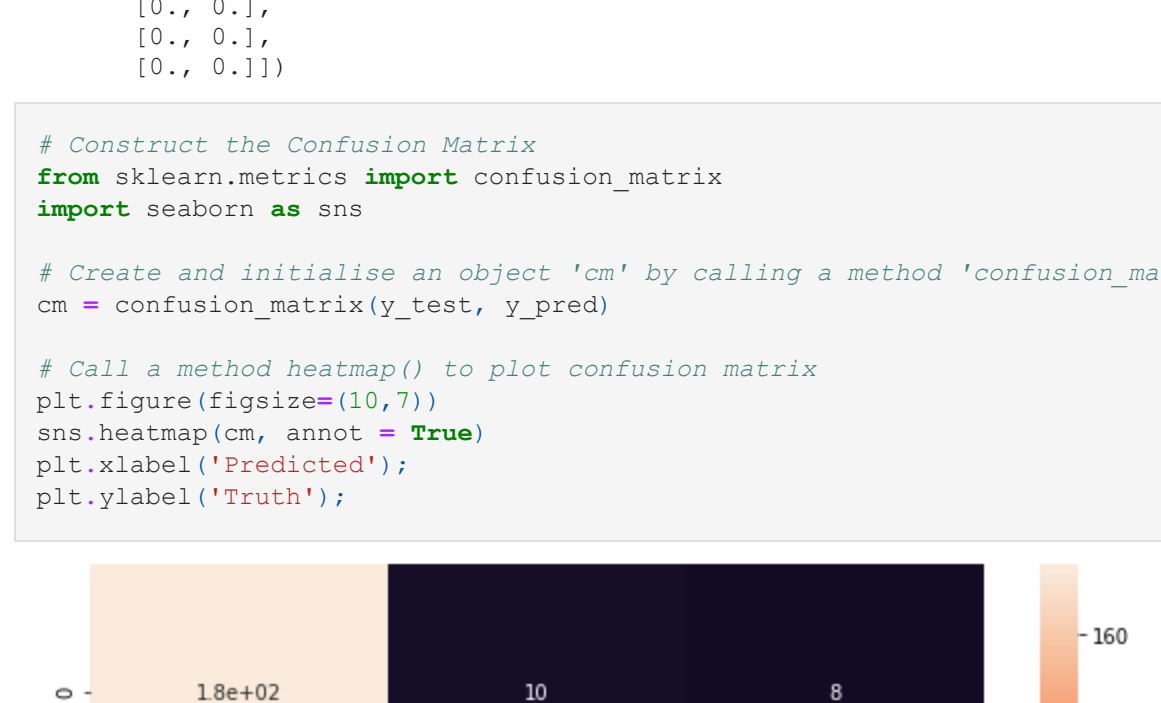
```
array([[0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.],
       [1., 1.],
       [1., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.]])
```

In [59]:

```python
# Construct the Confusion Matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Create and initialise an object 'cm' by calling a method 'confusion_matrix()'
cm = confusion_matrix(y_test, y_pred)

# Call a method heatmap() to plot confusion matrix
plt.figure(figsize=(10,7))
sns.heatmap(cm, annot = True)
plt.xlabel('Predicted');
plt.ylabel('Truth');
```



In [60]:

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

	precision	recall	f1-score	support
0.0	0.98	0.91	0.94	194
1.0	0.17	0.33	0.22	6
2.0	0.00	0.00	0.00	0
accuracy			0.89	200
macro avg	0.38	0.41	0.39	200
weighted avg	0.95	0.89	0.92	200

```
D:\Emrullah\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
D:\Emrullah\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
D:\Emrullah\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [61]:

```python
from sklearn.model_selection import cross_val_score

# Create an initialise an object 'nvclassifier' by calling a method 'GaussianNB()'
nvclassifier = GaussianNB()

# Calculate the score using cross validation method
cross_val_score(nvclassifier, X, y, scoring = 'accuracy', cv = 20).mean()
```

```
D:\Emrullah\lib\site-packages\sklearn\model_selection\_split.py:666: UserWarning: The least populated class in y has only 5 members, which is less than n_splits=20.
  warnings.warn(("The least populated class in y has only %d"
```

Out[61]: 0.9419795918367349

After setting testing and training dataset I use standard scaler to convert dataset on the same scales for entire columns. After scaling dataset I apply Naive Bayes classifier for machine learning and the accuracy for that is 91% which is not too bad and to improve the accuracy I use Cross Validation to put more data mixture into training and the accuracy goes up to 94%.

Insights or Takeaways:

- Drama is the most popular genre with the majority of the audience, this is then followed by Comedy and Romance. Therefore Producers can focus on these genres in future.
- Tom Hanks, Russo Brothers and Leonardo DiCaprio produced most revenue in the Actor section.
- Steven Spielberg, Anthony Russo and Christopher Nolan are the directors that bring the highest revenue.
- Movies that are between 120 to 150 minutes are leading in the movie sector.
- There are movies with higher metrics of IMBD rating and metascore but they have not done well in terms of revenue.
- The films with Average Gross are more popular.

Question 2

In [62]:

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [63]:

```python
fruit = pd.read_csv('dataset/fruit_data_with_colours.csv')
```

In [64]:

```python
fruit.head()
```

Out[64]:

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60
3	2	mandarin	mandarin	86	6.2	4.7	0.80
4	2	mandarin	mandarin	84	6.0	4.6	0.79

In [65]:

```python
X= fruit.iloc[:,3:6].values
y=fruit.iloc[:,0].values
```

In [66]:

```python
print(X,y)
```

```
[[192.    8.4   7.3]
 [180.    8.    6.8]
 [176.    7.4   7.2]
 [ 86.    6.2   4.7]
 [ 84.    6.    4.6]
 [ 80.    5.8   4.3]
 [ 80.    5.9   4.3]
 [ 76.    5.8   4. ]
 [178.    7.1   7.8]
 [172.    7.4   7. ]
 [166.    6.9   7.3]
 [172.    7.1   7.6]
 [154.    7.    7.1]
 [164.    7.3   7.7]
 [152.    7.6   7.3]
 [156.    7.7   7.1]
 [156.    7.6   7.5]
 [168.    7.5   7.6]
 [162.    7.5   7.1]
 [162.    7.4   7.2]
 [160.    7.5   7.5]
 [156.    7.4   7.4]
 [140.    7.3   7.1]
 [170.    7.6   7.9]
 [342.    9.    9.4]
 [356.    9.2   9.2]
 [362.    9.6   9.2]
 [204.    7.5   9.2]
 [140.    6.7   7.1]
 [160.    7.    7.4]
 [158.    7.1   7.5]
 [210.    7.8   8. ]
 [164.    7.2   7. ]
 [190.    7.5   8.1]
 [142.    7.6   7.8]
 [150.    7.1   7.9]
 [160.    7.1   7.6]
 [154.    7.3   7.3]
 [158.    7.2   7.8]
 [144.    6.8   7.4]
 [154.    7.1   7.5]
 [180.    7.6   8.2]
 [154.    7.2   7.2]
 [194.    7.2  10.3]
 [200.    7.3  10.5]
 [186.    7.2   9.2]
 [216.    7.3  10.2]
 [196.    7.3   9.7]
 [174.    7.3  10.1]
 [132.    5.8   8.7]
 [130.    6.    8.2]
 [116.    6.    7.5]
 [118.    5.9   8. ]
 [120.    6.    8.4]
 [116.    6.1   8.5]
 [116.    6.3   7.7]
 [116.    5.9   8.1]
 [152.    6.5   8.5]
 [118.    6.1   8.1]] [1 1 1 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
```

In [67]:

```python
fruit['fruit_name'].value_counts()
```

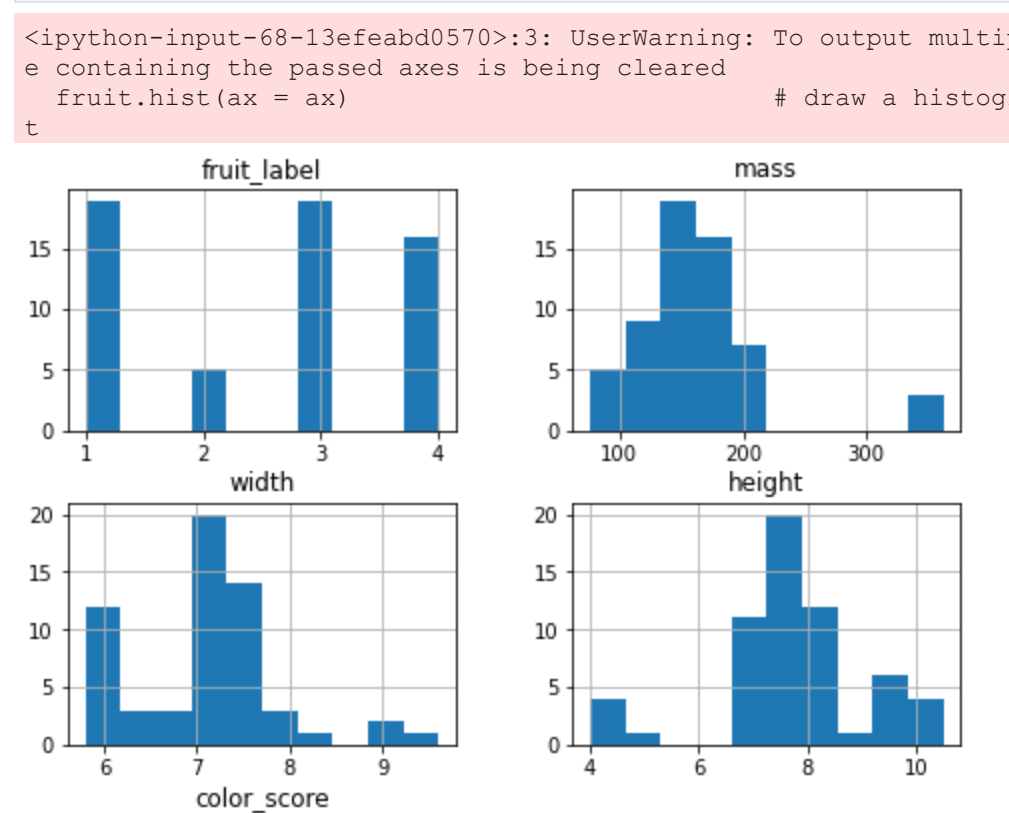
Out[67]:

```
apple       19
orange      19
lemon       16
mandarin     5
Name: fruit_name, dtype: int64
```

In [68]:

```python
fig = plt.figure(figsize = (8, 8))                # Size of the figure
ax = fig.gca()
fruit.hist(ax = ax)                               # draw a histogram based on the dataset
plt.show()
```

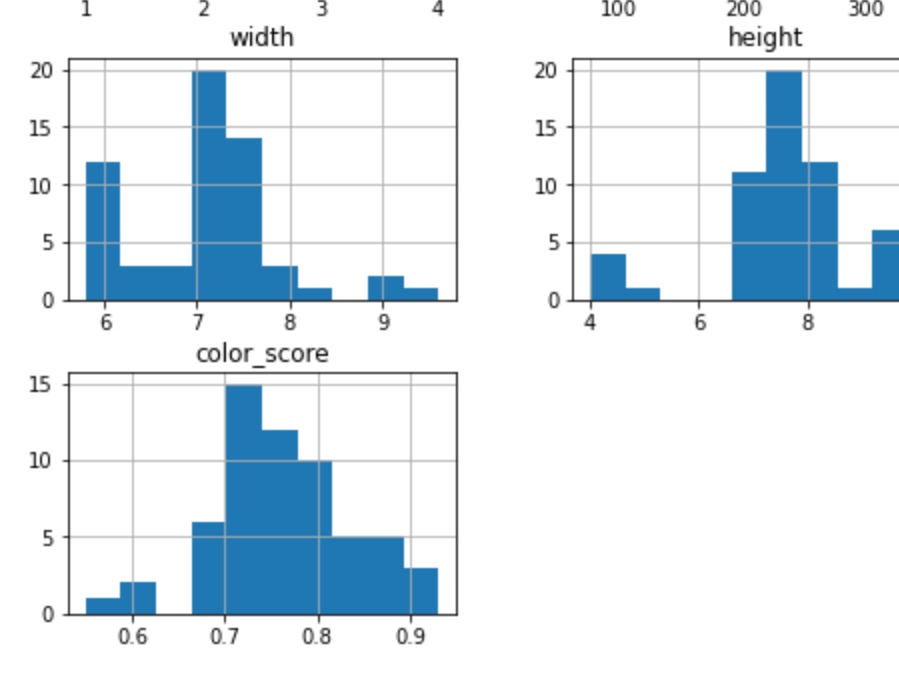
```
<ipython-input-68-13efeabd0570>:3: UserWarning: To output multiple subplots, the figure containing the passed axes is being cleared
  fruit.hist(ax = ax)                              # draw a histogram based on the dataset
```



To be able to see the distribution of each featurs I use histogram

In [69]:

```python
#compare 2 features together
sns.relplot(x='width', y='height',hue='mass',palette='BrBG',data=fruit);
```



In [ ]:

```python
# Import Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
 # Import train_test_split function
from sklearn.model_selection import train_test_split
# Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
from sklearn import tree
```

In [ ]:

```python
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_sta
```

In [ ]:

```python
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

In [ ]:

```python
# Create Decision Tree classifer object
clf = tree.DecisionTreeClassifier( min_samples_split=2)

# Train Decision Tree Classifer
clf = clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

In [ ]:

```python
print(y_pred)
```

In [ ]:

```python
pred_train=clf.predict(X_train)
pred_train
```

In [ ]:

```python
from sklearn.metrics import confusion_matrix,precision_score,recall_score,accuracy_sco
```

In [ ]:

```python
# Model Accuracy, how often is the classifier correct?
print("Accuracy on train score :",metrics.accuracy_score(y_train,pred_train))
print("Accuracy on test score:",metrics.accuracy_score(y_test, y_pred))
```

In [ ]:

```python
# Import knearest neighbors Classifier model
from sklearn.neighbors import KNeighborsClassifier

# Create KNN Classifier
kNN = KNeighborsClassifier(n_neighbors = 3)

# Train the model using the training sets
kNN.fit(X_train, y_train)

# Predict the response for test dataset
y_pred2 = kNN.predict(X_test)

X_pred=kNN.predict(X_train)
X_pred
```

In [ ]:

```python
# Model Accuracy, how often is the classifier correct?
print("Accuracy on train score :",metrics.accuracy_score(y_train,X_pred))
print("Accuracy on test score:",metrics.accuracy_score(y_test, y_pred2))
```

In the fruit datset I apply 2 machine learning to be able to see data accuracy. For KNN model the accuracy is underfitting which is showing that accuracy for training and testing data is low. In other words, the model has high bias and high variance. However, for decision tree model the Training accuracy is high and testing accuracy is low. This means the model has low bias, high variance.

HOW TO PREVENT OVERFITTING AND UNDERFITTING(1)

1-Cross-validation We can use initial training data to generate multiple mini train-test splits. Use these splits to tune the model.

2-Train with more data Training with more data can help algorithms detect the signal better, for example if we increase mandarin and lemon we might have better results.

3-Remove features Basically remove irrelevant features to improve model.

4-Early stopping Especially if using itterations, after certain points itteration can improve model.

5-Regularization Regularization refers to a broad range of techniques for artificially forcing your model to be simpler.

Question 3

Decision Tree algorithm belongs to the family of supervised learning algorithms. The decision tree algorithm can be used for solving regression and classification problems too.

The primary challenge in the decision tree implementation is to identify which attributes we need to be considered as the root node on each level, this is called attributes selection.

Decision trees use multiple algorithms to decide to split a node into two or more sub-nodes. The algorithm selection is also based on the type of target variables.

Some of the Algorithms:

ID3 → (extension of D3) C4.5 → (successor of ID3) CART → (Classification And Regression Tree) CHAID → (Chi-square automatic interaction detection Performs multi-level splits when computing classification trees)

The Attribute Selection Measures

If the dataset consists of N attributes then deciding which attribute to place at the root or at different levels of the tree as internal nodes is a complicated step as just randomly selecting any node to be the root can't solve the issue. If we follow a random approach it may give us bad results with low accuracy.

For solving this attribute selection problem, here are some solutions for using some criteria like:

Entropy, Information gain, Gini index, Gain Ratio, Reduction in Variance Chi-Square

These criteria will calculate values for every attribute. The values are sorted and attributes are placed in the tree by following the order, i.e the attribute with a high value (in case of information gain) is placed at the root. While using Information Gain as a criterion we assume attributes to be categorical and for the Gini index attributes are assumed to be continuous.

After choosing Attribute Selection another max_depth means how deep the decison tree will go.

Also another challenge is the decison tree result might have an overfitting issue.

For over fitting there are 2 solutions:

1- Pruning the Tree 2- Using Random Forest

Entropy is an information theory metric that measures the impurity or uncertainty in a group of observations and helps us to build an appropriate decision tree for selecting the best splitter. Entropy can be between 0 and 1, the higher the entropy the harder it is to draw any conclusions from that information. A branch with an entropy of zero is a leaf node and a brach with entropy more than zero needs further splitting.

The internal working of Gini impurity is also somewhat similar to the working of entropy in the Decision Tree. In the Decision Tree algorithm, both are used for building the tree by splitting as per the appropriate features but there is quite a difference in the computation of both the methods.

The range of Entropy lies in between 0 to 1 and the range of Gini Impurity lies in between 0 to 0.5. For computing power Gini impurity is more efficient.

The dataset contains details of the purchase of a product based on social network advertisements based on gender, age and estimated salary.

Target class belongs to two discrete categories of purchased and not purchased.

In [ ]:

```python
sn_df=pd.read_csv('dataset/Social_Network_Ads.csv')
```

In [ ]:

```python
sn_df.head()
```

In [ ]:

```python
sn_df.info()
```

In [ ]:

```python
sn_df['Purchased'].value_counts()
```

In [ ]:

```python
sn_df['Gender'].value_counts()
```

In [ ]:

```python
sn_df['User ID'].is_unique
```

In [ ]:

```python
sns.scatterplot(x='Age',y='EstimatedSalary',hue='Purchased', data=sn_df)
```

To be able to see the relationship between Age and Estimated Salary I used scatterplot and also to see target category for these two features I used hue attribute.

In [ ]:

```python
plt.figure(figsize=(10, 3))
sns.catplot(x='Purchased', col='Gender', kind='count', data=sn_df,palette="Set1");
plt.show()
```

To see Purchased target in male and female categories I used histogram and Female gender group purchasing is slightly higher than men

In [ ]:

```python
plt.figure(figsize=(10,5)
sns.histplot(sn_df['Age'],kde=True,color='b',bins=70);
```

The distribution of age group showing in this histogram between 30 and 45 are dominant

In [ ]:

```python
plt.figure(figsize=(10,7))
sns.histplot(sn_df['EstimatedSalary'],kde=True,palette="set2",bins=50);
```

Estimated salary skew is towards 70k to 90k

In [ ]:

```python
from sklearn.preprocessing import OrdinalEncoder
ord_enc = OrdinalEncoder()
sn_df['Gender'] = ord_enc.fit_transform(sn_df[['Gender']])
```

In [ ]:

```python
sn_df.head()
```

In [ ]:

```python
# Import Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
 # Import train_test_split function
from sklearn.model_selection import train_test_split
# Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
from sklearn import tree
```

In [ ]:

```python
X=sn_df.iloc[:,1:4]
y=sn_df.iloc[:,4]
```

In [ ]:

```python
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_sta
```

In [ ]:

```python
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

In [ ]:

```python
# data normalization with sklearn
from sklearn.preprocessing import MinMaxScaler

# fit scaler on training data
norm = MinMaxScaler().fit(X)

# transform training data
X_train = norm.transform(X_train)

# transform testing dataabs
X_test = norm.transform(X_test)
```

In [ ]:

```python
# instantiate the DecisionTreeClassifier model with criterion gini index
clf_gini = DecisionTreeClassifier(criterion = 'gini', max_depth = 3, random_state = 0)

# fit the model
clf_gini.fit(X_train, y_train)
```

In [ ]:

```python
# Predict the Test set results with criterion gini index
y_pred_gini = clf_gini.predict(X_test)

# Compare the train-set and test-set accuracy

y_pred_train_gini = clf_gini.predict(X_train)
```

In [ ]:

```python
from sklearn.metrics import accuracy_score
```

In [ ]:

```python
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_t
```

In [ ]:

```python
# print the scores on training and test set

print('Training set score: {:.4f}'.format(clf_gini.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(clf_gini.score(X_test, y_test)))
```

In [ ]:

```python
# instantiate the DecisionTreeClassifier model with criterion entropy index
clf_ent = DecisionTreeClassifier(criterion = 'entropy', max_depth = 3, random_state =

# fit the model
clf_ent.fit(X_train, y_train)
```

In [ ]:

```python
# Predict the Test set results with criterion entropy index
y_pred_ent = clf_ent.predict(X_test)

# Compare the train-set and test-set accuracy

y_pred_train_ent = clf_ent.predict(X_train)
```

In [ ]:

```python
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_t
```

In [ ]:

```python
# print the scores on training and test set

print('Training set score: {:.4f}'.format(clf_ent.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(clf_ent.score(X_test, y_test)))
```

The result with Gini index and Entropy are quite close.

Question 4

Text analytics (also known as text mining or text data mining) is the process of extracting information and uncovering actionable insights from unstructured text. Giving an example, analysis of customer reviews of products or services helps companies understand user sentiment or common issues customers are talking about.

Natural Language Processing is not bound to text only. Consider for example speech recognition and processing of speech, or even sign language, which is visually communicated eg. Google Alexa and Apple Siri.

In [ ]:

```python
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
import os
import seaborn as sns
import matplotlib.pyplot as plt
```

In [ ]:

```python
email_df=pd.read_csv('dataset/SMS.csv', encoding='unicode_escape')
```

In [ ]:

```python
email_df.head()
```

In [ ]:

```python
email_df.shape
```

In [ ]:

```python
email_df=email_df.drop(columns=['S. No.'])
```

In [ ]:

```python
email_df.head()
```

In [ ]:

```python
#check if any null values
email_df.isnull().sum()
```

In [ ]:

```python
email_df.describe(include=object)
```

In [ ]:

```python
#Lets see the distribution of labels on barchart
email_df['Label'].value_counts().plot(kind='bar',color=['blue','red']);
```

In [ ]:

```python
# Import libraries
import re
import nltk
from nltk.tokenize import sent_tokenize
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
```

In [ ]:

```python
# label encoding for Target column
from sklearn import preprocessing

label_encoder = preprocessing.LabelEncoder()
email_df['Label'] = label_encoder.fit_transform(email_df['Label'])
email_df
```

In [ ]:

```python
#install wordcloud
#pip install wordcloud
```

In [ ]:

```python
from wordcloud import WordCloud
#Plotting the spam words
plt.figure(figsize = (20,20))
wc = WordCloud(max_words = 200 , width = 1600 , height = 800).generate(" ".join(email_
plt.imshow(wc , interpolation = 'bilinear')
plt.title("Spam Word Cloud")
```

In [ ]:

```python
##Plotting the Non-spam words
plt.figure(figsize = (20,20))
wc = WordCloud(max_words = 200 , width = 1600 , height = 800).generate(" ".join(email_
plt.imshow(wc , interpolation = 'bilinear')
plt.title("Spam Word Cloud")
```

In [ ]:

```python
# Create stemmer
porter = PorterStemmer()
```

In [ ]:

```python
CleanedEmail = []
for i in range(957):
    email = re.sub('[^a-zA-Z]',' ', email_df["Message_body"][i])
    # Cleaning url
    email = re.sub(r'http\S+', ' ', email_df["Message_body"][i])

    # remove html tags
    email = re.sub('r<.*?>',' ', email_df["Message_body"][i])

     # remove hastags
    email = re.sub(r'#\w+',' ', email_df["Message_body"][i])

     # remove mentions
    email = re.sub(r'@\w+',' ', email_df["Message_body"][i])

    # remove digits
    email = re.sub(r'\d+', ' ', email_df["Message_body"][i])

    email = email.lower()
    email = email.split()
    email = [porter.stem(word) for word in email if not word in set(stopwords.words("e
    email = ''.join(email)
    CleanedEmail.append(email)
```

In [ ]:

```python
CleanedEmail
```

In [ ]:

```python
#lets take first 1000 features
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features = 1000, analyzer='word', lowercase=False)
```

In [ ]:

```python
X =cv.fit_transform(CleanedEmail).toarray()
```

In [ ]:

```python
X.shape
```

In [ ]:

```python
y=email_df.iloc[:,1]
```

In [ ]:

```python
y.shape
```

In [ ]:

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_stat
```

In [ ]:

```python
from sklearn.naive_bayes import MultinomialNB

# Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

# Model Generation Using Multinomial Naive Bayes
clf = MultinomialNB().fit(X_train, y_train)
predicted_1= clf.predict(X_test)

# Display the predicted score
print("MultinomialNB Accuracy:",metrics.accuracy_score(y_test, predicted_1))
```

In [ ]:

```python
from sklearn.feature_extraction.text import TfidfVectorizer

# Create an object 'tf' by calling a method TfidfVectorizer()
tf = TfidfVectorizer()

# Train the dataset by calling a method fit_tranform()
text_tf = tf.fit_transform(CleanedEmail)
```

In [ ]:

```python
from sklearn.model_selection import train_test_split

# Divide the dataset into training and testing
X_train, X_test, y_train, y_test = train_test_split( text_tf, y , test_size = 0.2, ran
```

In [ ]:

```python
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
# Model Generation Using Multinomial Naive Bayes
clf = MultinomialNB().fit(X_train, y_train)
predicted_2= clf.predict(X_test)
print("MultinomialNB Accuracy:",metrics.accuracy_score(y_test, predicted_2))
```

In [ ]:

In [ ]:

Qustion 5

In passive Reinforcement Learning, the agent's policy is fixed which means with this policy(π) the agent tries to learn how good the policy is by observing the environment and going with this policy. For example, parking can be achieved by learning automatic parking policies.

In active Reinforcement Learning, an agent needs to decide what to do as there is no fixed policy that it can act on. The agent attempts to find an optimal (or at least good) policy by exploring different actions in the environment. For example, when playing chess the agent does not have a fixed policy to follow.

In [ ]:

References: In the reference the numbers in brackets indicate which topics/question it belongs to

(1) https://elitedatascience.com/overfitting-in-machine-learning
(5)https://towardsdatascience.com/applications-of-reinforcement-learning-in-real-world-1a94955bcd12
(5)https://neptune.ai/blog/reinforcement-learning-applications
(5)https://www.simplilearn.com/tutorials/machine-learning-tutorial/what-is-q-learning