Double-click (or enter) to edit

TASK-2 CODSOFT(DS) MOVIE RATING PREDICTION WITH PYTHON

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

```
X=df_movie[['Year','Votes','Duration','Genre_mean_rating','Director_encoded','Actor1_encoded','Actor2_encoded','Actor3_encoded']]
y=df_movie['Rating']
```

```
# Try reading the file with a different encoding, like 'latin-1'
df_movie= pd.read_csv("/content/drive/MyDrive/codsodt1/IMDb Movies India.csv",sep=",",engine='python',encoding='latin-1')
df_movie.dropna(inplace=True)
df_movie.head(10)
```

	Name	Year	Duration	Genre	Rating	Votes	Director	Actor 1	Actor 2	Actor 3
1	#Gadhvi (He thought he was Gandhi)	(2019)	109 min	Drama	7.0	8	Gaurav Bakshi	Rasika Dugal	Vivek Ghamande	Arvind Jangid
3	#Yaaram	(2019)	110 min	Comedy, Romance	4.4	35	Ovais Khan	Prateik	Ishita Raj	Siddhant Kapoor
5	...Aur Pyaar Ho Gaya	(1997)	147 min	Comedy, Drama, Musical	4.7	827	Rahul Rawail	Bobby Deol	Aishwarya Rai Bachchan	Shammi Kapoor
6	...Yahaan	(2005)	142 min	Drama, Romance, War	7.4	1,086	Shoojit Sircar	Jimmy Sheirgill	Minissha Lamba	Yashpal Sharma
8	?: A Question Mark	(2012)	82 min	Horror, Mystery, Thriller	5.6	326	Allyson Patel	Yash Dave	Muntazir Ahmad	Kiran Bhatia
9	@Andheri	(2014)	116 min	Action, Crime, Thriller	4.0	11	Biju Bhaskar Nair	Augustine	Fathima Babu	Byon
10	1:1.6 An Ode to Lost Love	(2004)	96 min	Drama	6.2	17	Madhu Ambat	Rati Agnihotri	Gulshan Grover	Atul Kulkarni
11	1:13:7 Ek Tera Saath	(2016)	120 min	Horror	5.9	59	Arshad Siddiqui	Pankaj Berry	Anubhav Dhir	Hritu Dudani
12	100 Days	(1991)	161 min	Horror, Romance, Thriller	6.5	983	Partho Ghosh	Jackie Shroff	Madhuri Dixit	Javed Jaffrey

Next steps: Generate code with df_movie View recommended plots New interactive sheet

```
df_movie.shape
```

```
(5659, 10)
```

```
df_movie.describe()
```

	Rating
count	5659.000000
mean	5.898533
std	1.381165
min	1.100000
25%	5.000000
50%	6.100000
75%	6.900000
max	10.000000

```
df_movie.isna().sum()
```

	0
Name	0
Year	0
Duration	0
Genre	0
Rating	0
Votes	0
Director	0
Actor 1	0
Actor 2	0
Actor 3	0

```
df_movie.drop_duplicates(inplace=True)
```

```
df_movie.shape
```

```
(5659, 10)
```

```
df_movie.columns
```

```
Index(['Name', 'Year', 'Duration', 'Genre', 'Rating', 'Votes', 'Director',
       'Actor 1', 'Actor 2', 'Actor 3'],
      dtype='object')
```

DATA PRE-PROCESSING

```
#Replacing the brackets from year column.
df_movie['Year'] = df_movie['Year'].astype(str).str.replace('[()]','',regex=True).astype(int)
```

```
#Remove the min word from 'Duration' column and convert all values to numeric.
df_movie['Duration'] = df_movie['Duration'].astype(str).str.replace(' min','').astype(float)
```

```
#Splitting the genre by,to keep only unique genres and replacing the null values with mode.
df_movie['Genre'] = df_movie['Genre'].str.split(',').str[0]
df_movie=df_movie.explode('Genre')
df_movie['Genre'].fillna(df_movie['Genre'].mode()[0], inplace=True)
```

```
#Convert 'Votes' to numeric and replace the, to keep only numerical part.
df_movie['Votes']=pd.to_numeric(df_movie['Votes'].str.replace(',',''))
```

```
#Checking the dataset is there any null values present and data types of the features present.
df_movie.info()
```

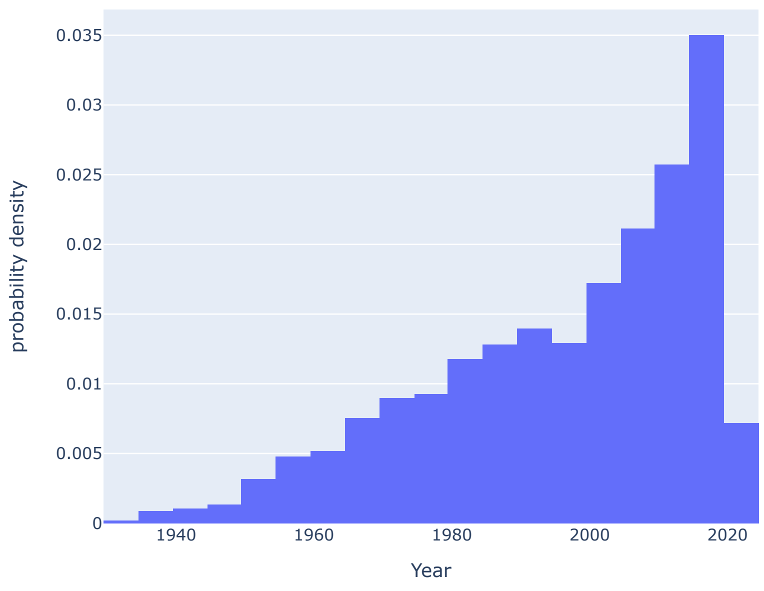
```
<class 'pandas.core.frame.DataFrame'>
Index: 5659 entries, 1 to 15508
Data columns (total 10 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Name      5659 non-null   object 
 1   Year      5659 non-null   int64  
 2   Duration  5659 non-null   float64
 3   Genre     5659 non-null   object 
 4   Rating    5659 non-null   float64
 5   Votes     5659 non-null   int64  
 6   Director  5659 non-null   object 
 7   Actor 1   5659 non-null   object 
 8   Actor 2   5659 non-null   object 
 9   Actor 3   5659 non-null   object 
dtypes: float64(2), int64(2), object(6)
memory usage: 486.3+ KB
```

DATA VISUALIZING

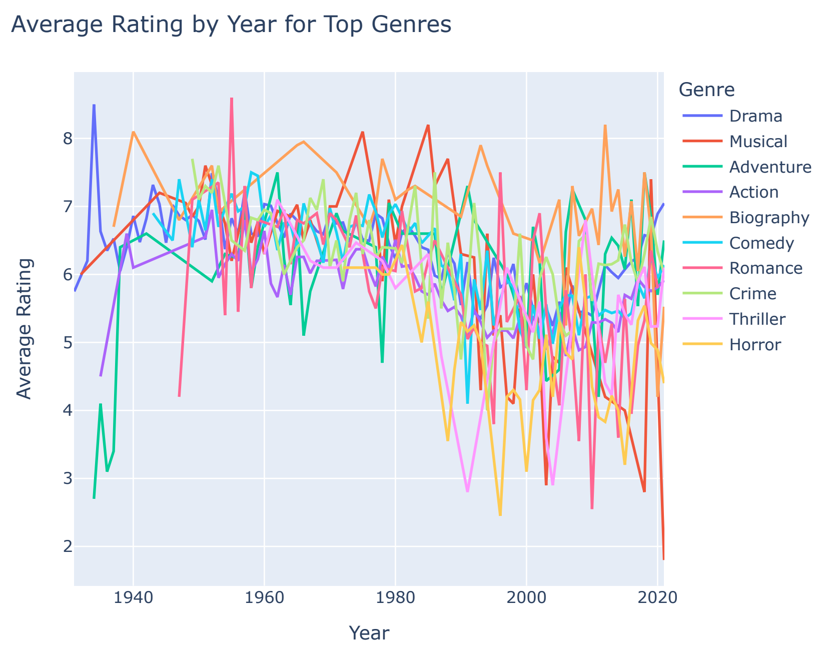
```
#Import the plotly.express module
!pip install plotly
import plotly.express as px
```

```
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (5.15.0)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly) (9.0.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from plotly) (24.1)
```

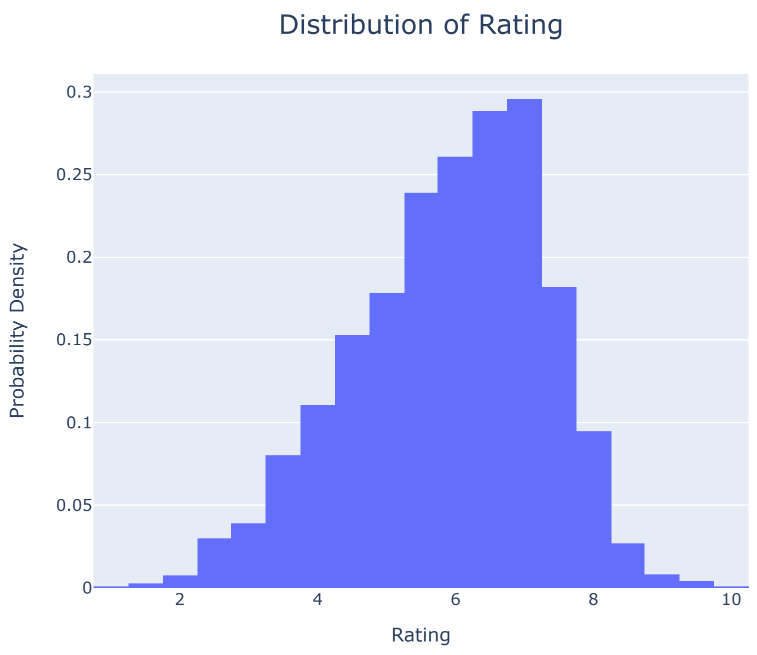
```
#Here we have created a histogram over the years in the data.
Year=px.histogram(df_movie,x='Year',histnorm='probability density',nbins=30)
Year.show()
```



```
#Group date by Year and calculate the average rating.
avg_rating_by_year=df_movie.groupby(['Year','Genre'])['Rating'].mean().reset_index()

#Get the top 10 genres.
top_genres=df_movie['Genre'].value_counts().head(10).index

#Filter the data to include only the top 3 genres
average_rating_by_year=avg_rating_by_year[avg_rating_by_year['Genre'].isin(top_genres)] # Fixed a typo here, it should be isin not ,isir

#Create the line plot with  plotly Express
fig=px.line(average_rating_by_year,x='Year',y='Rating',color='Genre')

#Updating the details into chart like tittle and hue
fig.update_layout(title='Average Rating by Year for Top Genres',xaxis_title='Year',yaxis_title='Average Rating')

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



```
#This histogram shows the distribution of ratings and its probable density.
rating_fig=px.histogram(df_movie,x='Rating',histnorm='probability density',nbins=40)
rating_fig.update_layout(title='Distribution of Rating',title_x=0.5,title_pad=dict(t=20),title_font=dict(size=20),xaxis_title='Rating',y
rating_fig.show()
```



FEATURE ENGINEERING

```
#Importing essential libraries for model building
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.linear_model import LinearRegression # Import LinearRegression from the correct module
from sklearn.metrics import accuracy_score,mean_absolute_error,mean_squared_error,r2_score
```

```
#Dropping Name column because it does'nt impact the outcome.
df_movie.drop('Name',axis=1,inplace=True)
```

```
#Grouping the columns with their average rating and then creating a new feature.
genre_mean_rating=df_movie.groupby('Genre')['Rating'].transform('mean')
df_movie['Genre_mean_rating']=genre_mean_rating
```

```
director_mean_rating=df_movie.groupby('Director')['Rating'].transform('mean')
df_movie['Director_encoded']=director_mean_rating

actor1_mean_rating=df_movie.groupby('Actor 1')['Rating'].transform('mean')
df_movie['Actor1_encoded']=actor1_mean_rating

actor2_mean_rating=df_movie.groupby('Actor 2')['Rating'].transform('mean')
df_movie['Actor2_encoded']=actor2_mean_rating

actor3_mean_rating=df_movie.groupby('Actor 3')['Rating'].transform('mean')
df_movie['Actor3_encoded']=actor3_mean_rating



# Assuming 'Rating' is your target variable and the rest are features
X = df_movie.drop('Rating', axis=1)  # Features
y = df_movie['Rating']  # Target variable

# Splitting the dataset into training and testing parts.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

MODEL BUILDING

```
# Assuming 'Rating' is your target variable and the rest are features
X = df_movie.drop('Rating', axis=1)  # Features
y = df_movie['Rating']  # Target variable

# Use one-hot encoding to convert categorical features to numerical
X = pd.get_dummies(X)

# Splitting the dataset into training and testing parts.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#Building machine Learning model and training them.
Model=LinearRegression()
Model.fit(X_train,y_train)
Model_pred=Model.predict(X_test)



#Evaluating the performance of model with evaluation metrics.
print('The performance evaluation of Logistic Regression is below:','\n')
print('Mean squared error:',mean_squared_error(y_test,Model_pred))
print('Mean absolute error:',mean_absolute_error(y_test,Model_pred))
print('R2 score:',r2_score(y_test,Model_pred))
```

```
The performance evaluation of Logistic Regression is below:

Mean squared error: 16464979728102.89
Mean absolute error: 658393.281555279
R2 score: -8891608599549.445
```

MODEL TESTING

```
X.head(5)
```

	Year	Duration	Votes	Genre_mean_rating	Director_encoded	Actor1_encoded	Actor2_encoded	Actor3_encoded	Genre_Action	Genre_A
1	2019	109.0	8	6.248697	7.000000	6.850000	7.000000	7.000000	False	
3	2019	110.0	35	5.838423	4.400000	5.420000	4.400000	4.450000	False	
5	1997	147.0	827	5.838423	5.313333	4.788889	5.786667	5.872727	False	
6	2005	142.0	1086	6.248697	7.383333	5.435000	6.933333	6.500000	False	
8	2012	82.0	326	4.687500	5.600000	5.600000	5.883333	5.600000	False	

5 rows × 9296 columns

```
y.head(5)
```

	Rating
1	7.0
3	4.4
5	4.7
6	7.4
8	5.6

```
#For testing ,we create a new dataframe with values close to the any of our existing data to evaluate.
data={'Year':[2019],'Votes':[36],'Duration':[111],'Genre_mean_rating':[5.8],'Director_encoded':[4.5],'Actor1_encoded':[5.3],'Actor2_enc
test_data=pd.DataFrame(data)


# Assuming 'X' is the DataFrame used for training the model.

# Get missing columns from the training data
missing_cols = set(X.columns) - set(test_data.columns)

# Add missing columns to test_data and fill them with 0
for col in missing_cols:
    test_data[col] = 0

# Ensure the order of columns in test_data is the same as in X
test_data = test_data[X.columns]

# Now you can predict the rating
rating_prediction = Model.predict(test_data)

# Display the predicted result from the model.
print('The predicted rating for the movie is:', rating_prediction[0])
```

```
  DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  C
```