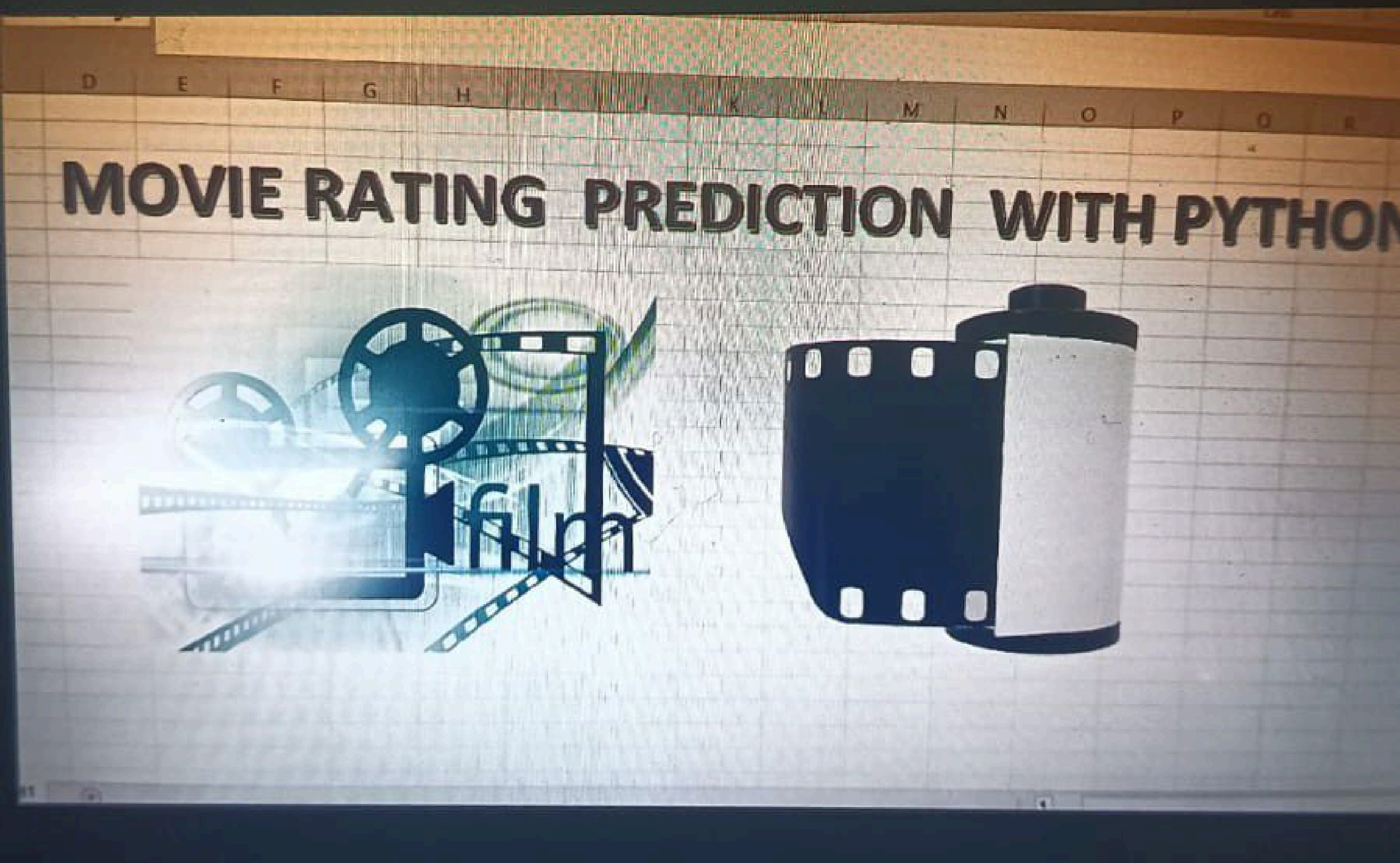
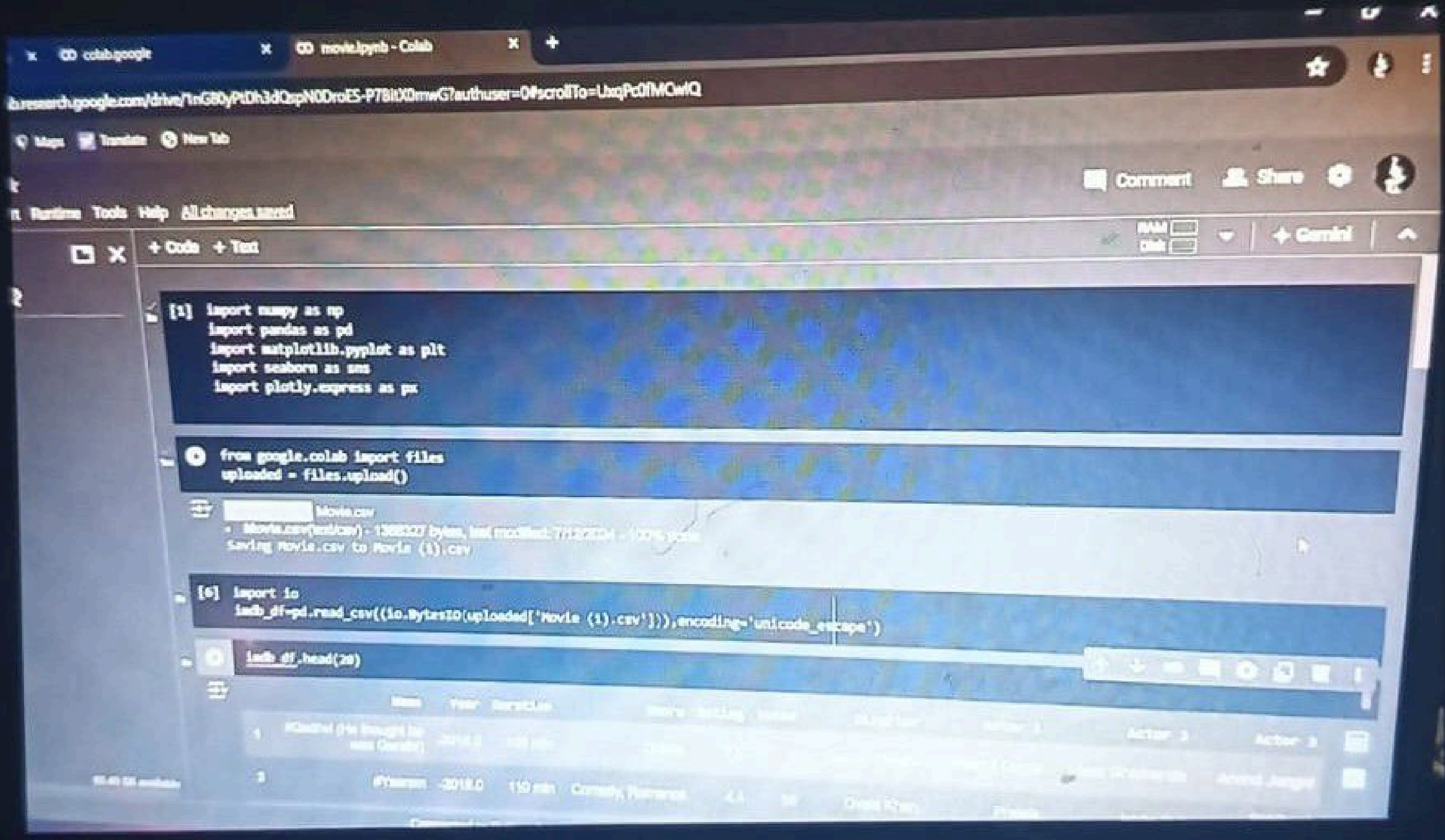


MOVIE RATING PREDICTION WITH PYTHON



```
[1] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
    import plotly.express as px
```

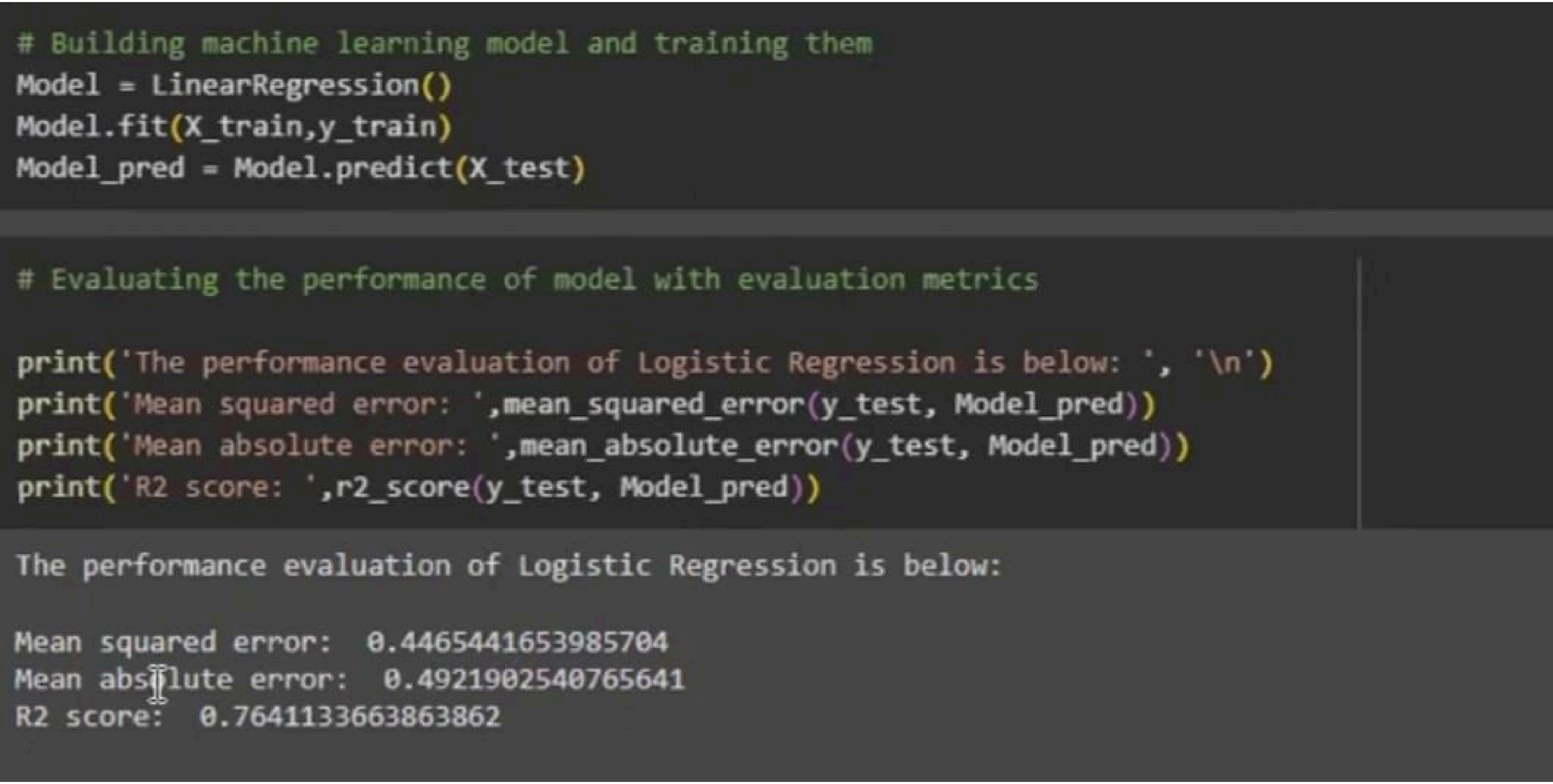
```
from google.colab import files
uploaded = files.upload()
```

```
Movie.csv
• Movie.csv(text/csv) - 1388327 bytes, last modified: 7/12/2024 - 100% done
Saving Movie.csv to Movie (1).csv
```

```
[6] import io
    imdb_df=pd.read_csv((io.BytesIO(uploaded['Movie (1).csv'])),encoding='unicode_escape')
```

```
imdb_df.head(20)
```

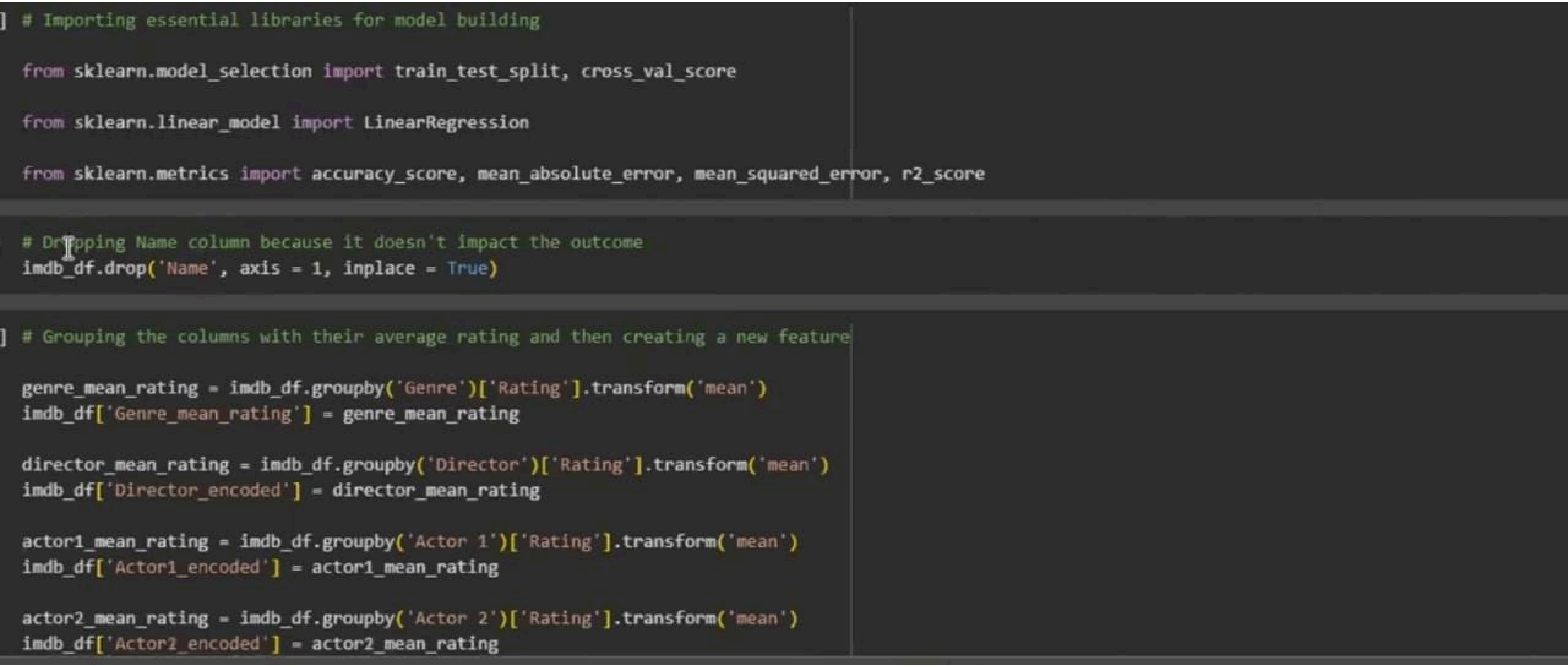
	Name	Year	Duration	Genre	Rating	Votes	Director	Actor 1	Actor 2	Actor 3
1	#Gadhvi (He thought he was Gandhi)	-2019.0	109 min	Drama	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
3	#Yaaram	-2019.0	110 min	Comedy, Romance	4.4	35	Ovais Khan	Prateik	[illegible]	[illegible]



```
# 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))
```

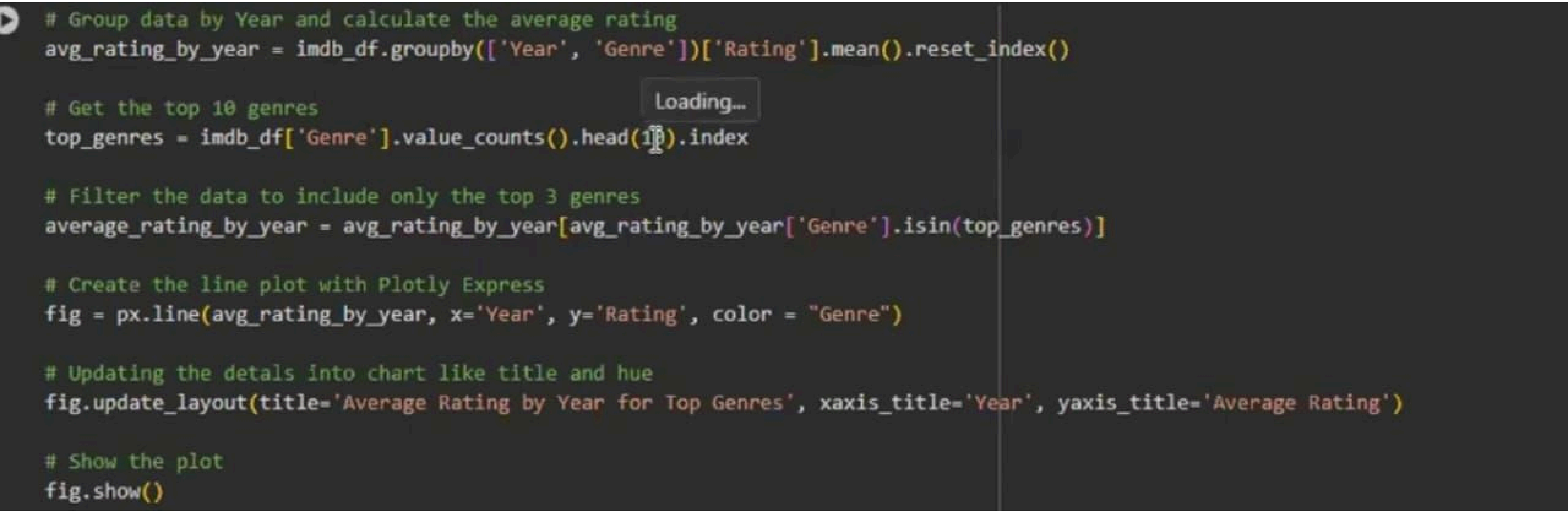
```
The performance evaluation of Logistic Regression is below:

Mean squared error:  0.4465441653985704
Mean absolute error:  0.4921902540765641
R2 score:  0.7641133663863862
```



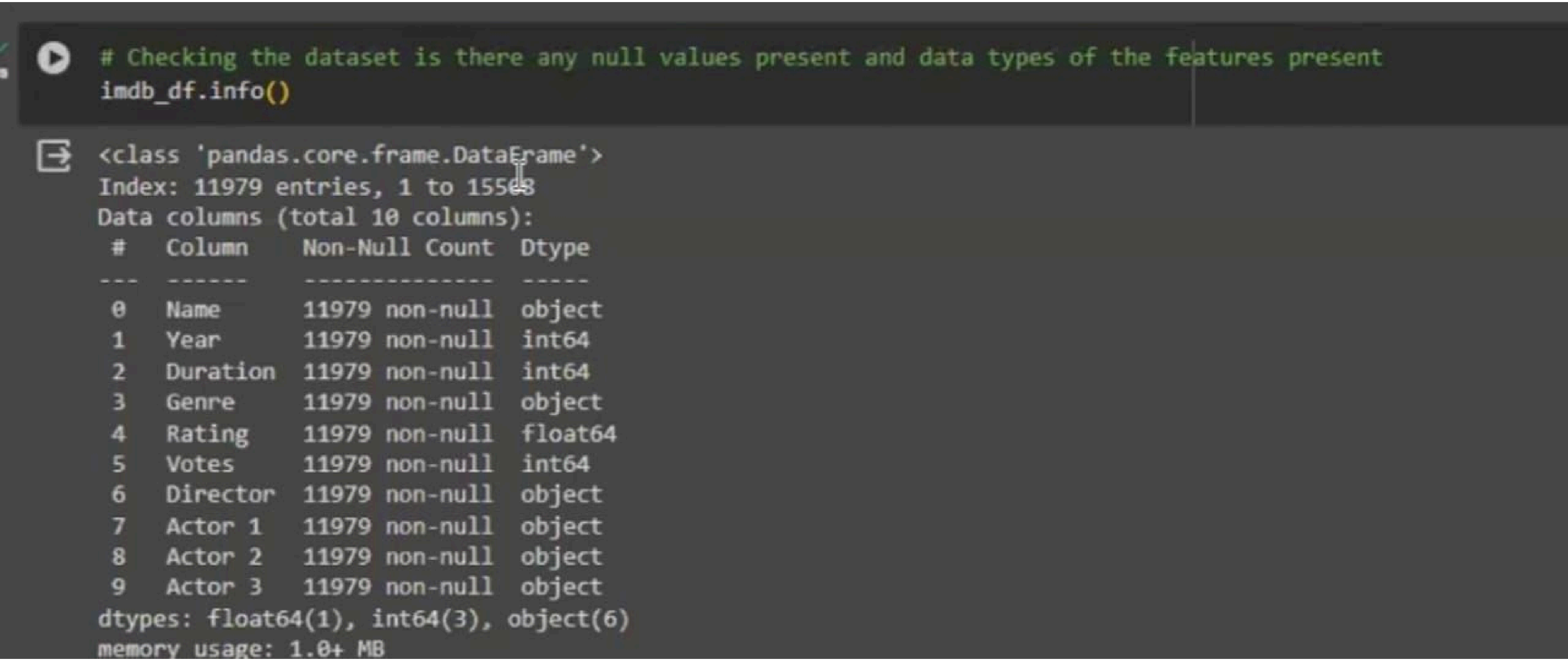
```
# Importing essential libraries for model building

from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.linear_model import LinearRegression

from sklearn.metrics import accuracy_score, mean_absolute_error, mean_squared_error, r2_score
```

```
# Dropping Name column because it doesn't impact the outcome
imdb_df.drop('Name', axis = 1, inplace = True)
```

```
# Grouping the columns with their average rating and then creating a new feature

genre_mean_rating = imdb_df.groupby('Genre')['Rating'].transform('mean')
imdb_df['Genre_mean_rating'] = genre_mean_rating

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

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

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

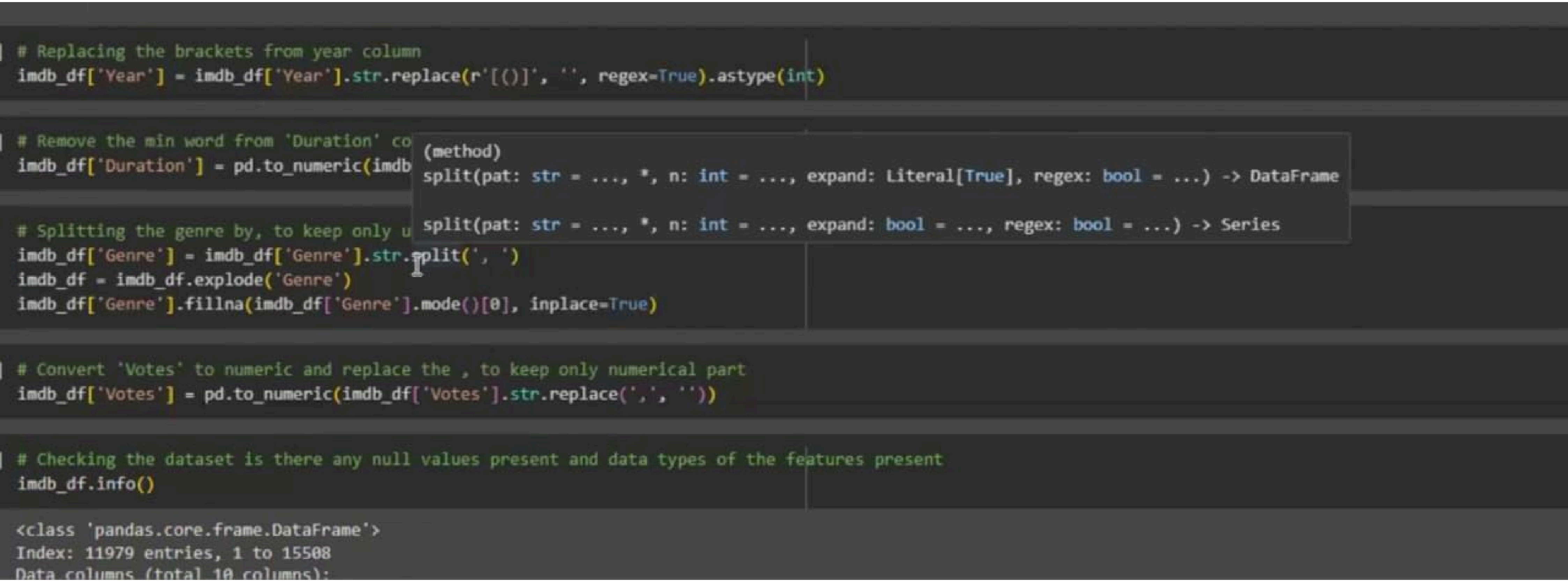


```
# Group data by Year and calculate the average rating
avg_rating_by_year = imdb_df.groupby(['Year', 'Genre'])['Rating'].mean().reset_index()

# Get the top 10 genres
top_genres = imdb_df['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)]

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

# Updating the detals into chart like title 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()
```



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

```
<class 'pandas.core.frame.DataFrame'>
Index: 11979 entries, 1 to 15508
Data columns (total 10 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Name      11979 non-null  object 
 1   Year      11979 non-null  int64  
 2   Duration  11979 non-null  int64  
 3   Genre     11979 non-null  object 
 4   Rating    11979 non-null  float64
 5   Votes     11979 non-null  int64  
 6   Director  11979 non-null  object 
 7   Actor 1   11979 non-null  object 
 8   Actor 2   11979 non-null  object 
 9   Actor 3   11979 non-null  object 
dtypes: float64(1), int64(3), object(6)
memory usage: 1.0+ MB
```



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

```
# Remove the min word from 'Duration' co
imdb_df['Duration'] = pd.to_numeric(imdb
```

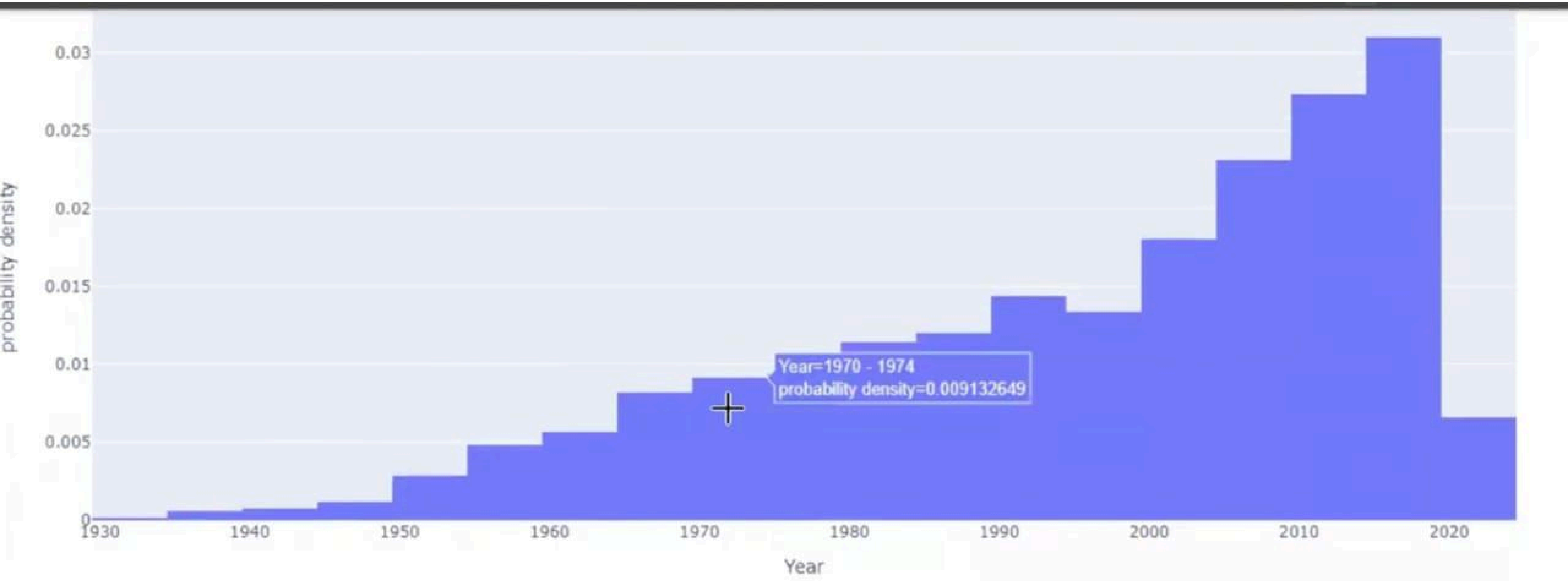
```
(method)
split(pat: str = ..., *, n: int = ..., expand: Literal[True], regex: bool = ...) -> DataFrame

split(pat: str = ..., *, n: int = ..., expand: bool = ..., regex: bool = ...) -> Series
```

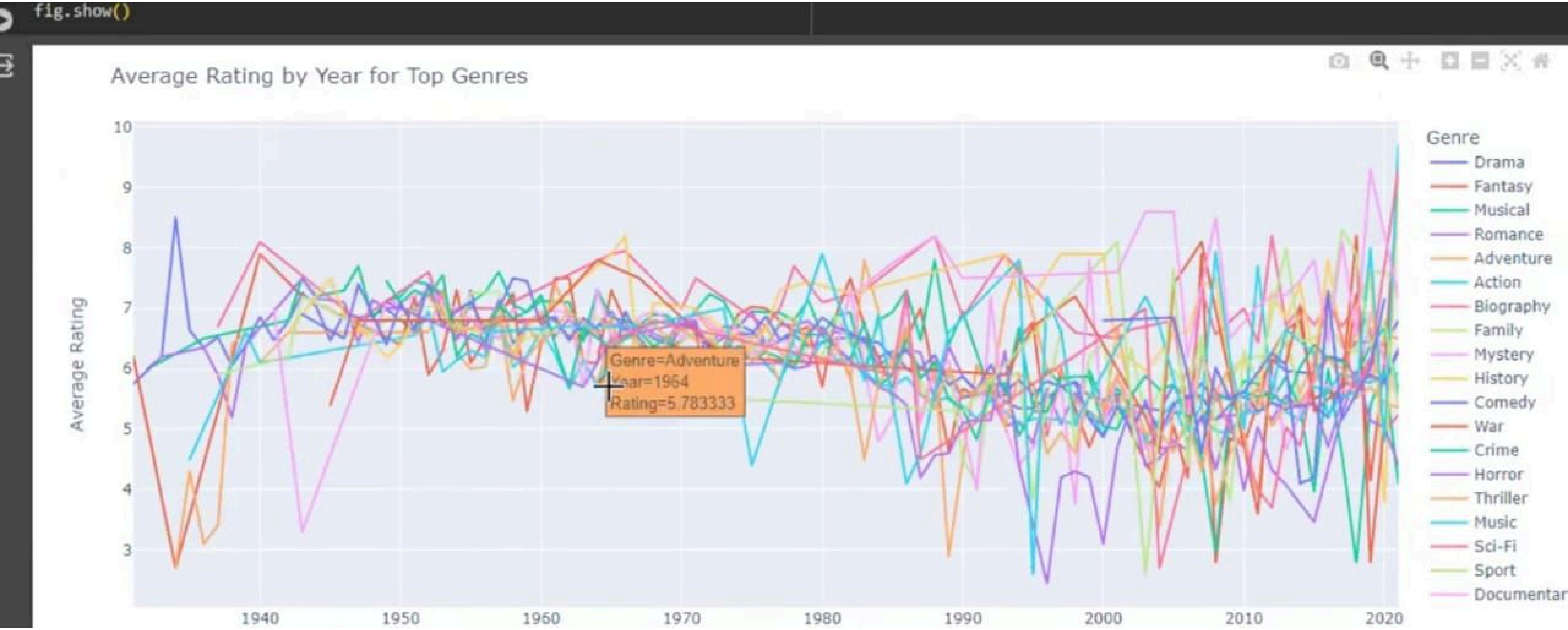
```
# Splitting the genre by, to keep only u
imdb_df['Genre'] = imdb_df['Genre'].str.split(', ')
imdb_df = imdb_df.explode('Genre')
imdb_df['Genre'].fillna(imdb_df['Genre'].mode()[0], inplace=True)
```

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

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

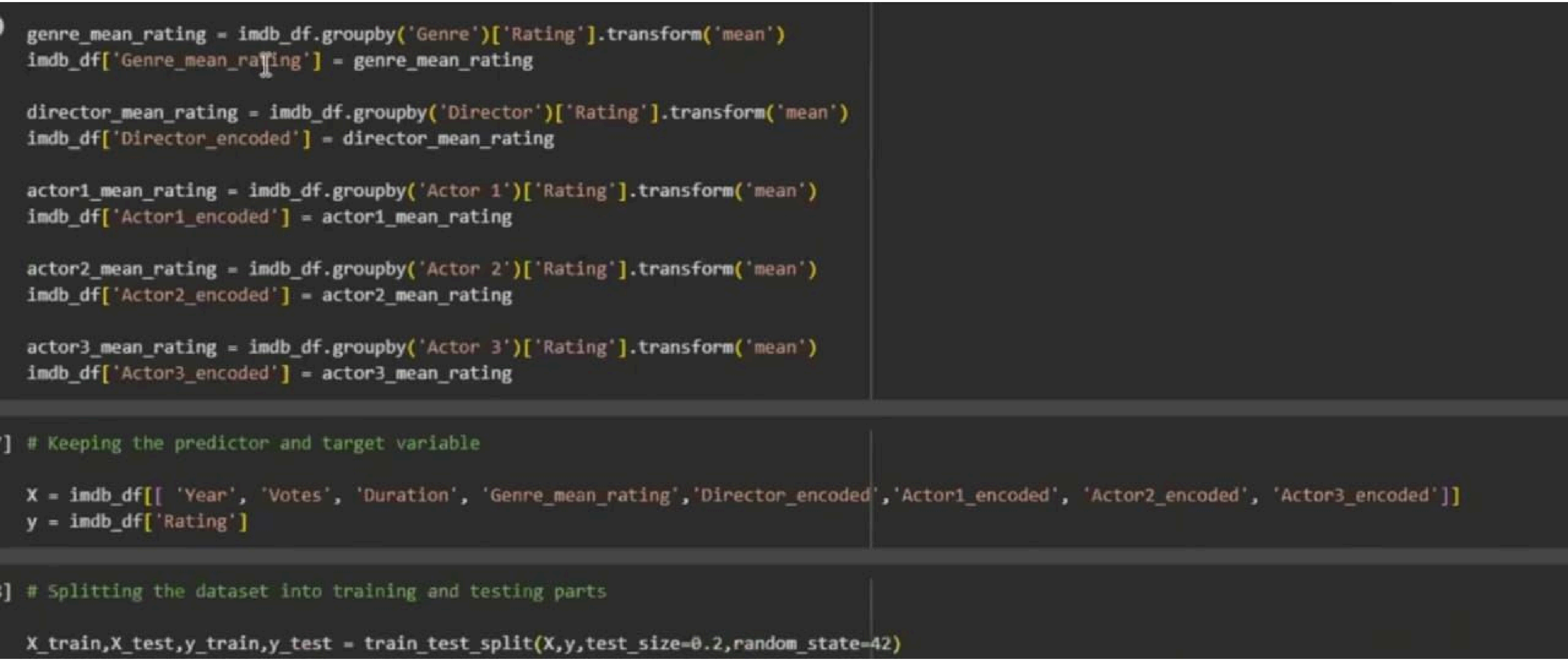
```
<class 'pandas.core.frame.DataFrame'>
Index: 11979 entries, 1 to 15508
Data columns (total 10 columns):
```


0.03
0.025
0.02
0.015
0.01
0.005
0
probability density
1930
1940
1950
1960
1970
1980
1990
2000
2010
2020
Year
Year=1970 - 1974
probability density=0.009132649



```
fig.show()
```

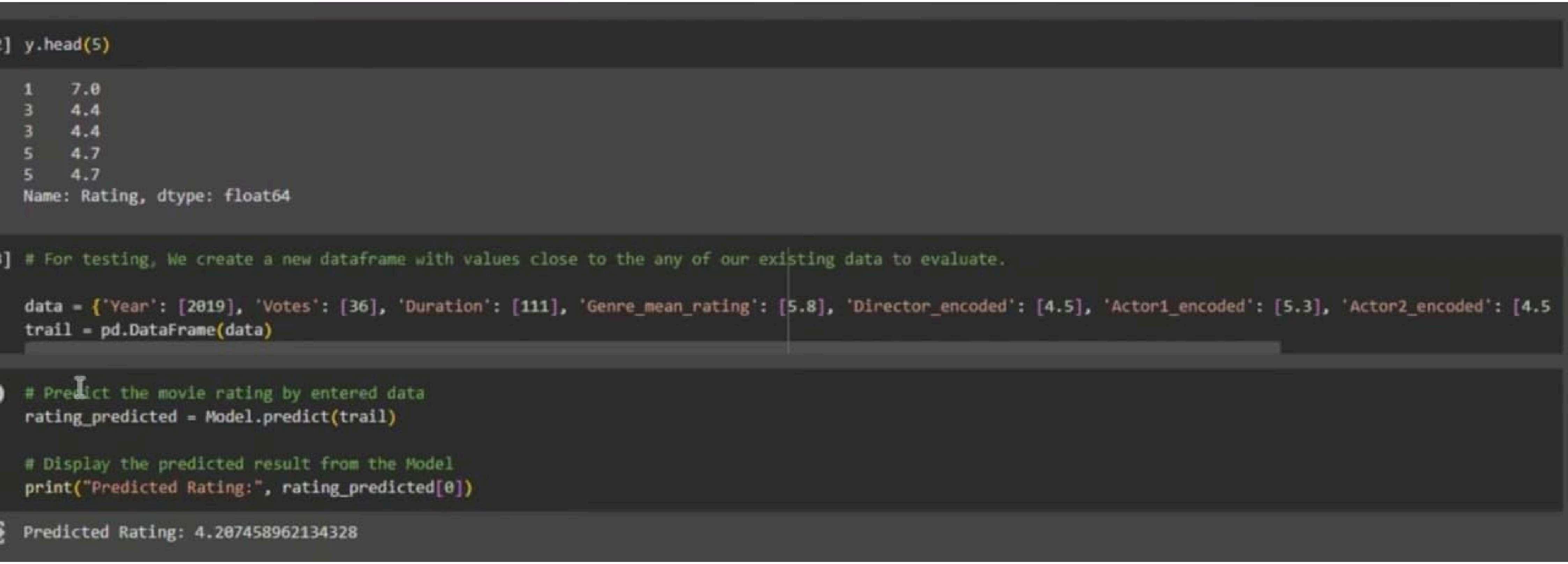
Average Rating by Year for Top Genres
Average Rating
10
9
8
7
6
5
4
3
1940
1950
1960
1970
1980
1990
2000
2010
2020
Genre=Adventure
Year=1964
Rating=5.783333
Genre
Drama
Fantasy
Musical
Romance
Adventure
Action
Biography
Family
Mystery
History
Comedy
War
Crime
Horror
Thriller
Music
Sci-Fi
Sport
Documentar



```
genre_mean_rating = imdb_df.groupby('Genre')['Rating'].transform('mean')
imdb_df['Genre_mean_rating'] = genre_mean_rating

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

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

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

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

```
# Keeping the predictor and target variable

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

```
# 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)
```



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

```
1    7.0
3    4.4
3    4.4
5    4.7
5    4.7
Name: Rating, dtype: float64
```

```
# 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_encoded': [4.5
trail = pd.DataFrame(data)
```

```
# Predict the movie rating by entered data
rating_predicted = Model.predict(trail)

# Display the predicted result from the Model
print("Predicted Rating:", rating_predicted[0])
```

```
Predicted Rating: 4.207458962134328
```