codetechit

February 25, 2024

1 TASK2 MOVIE RATING PREDICTION WITH PYTHON

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

[2]:
```
df = pd.read_csv('imdb_top_1000.csv')
df.head(2)
```

[2]:
```
                                        Poster_Link  \
0  https://m.media-amazon.com/images/M/MV5BMDFkYT…
1  https://m.media-amazon.com/images/M/MV5BM2MyNj…

              Series_Title Released_Year Certificate  Runtime         Genre  \
0  The Shawshank Redemption          1994           A  142 min         Drama
1             The Godfather          1972           A  175 min  Crime, Drama

   IMDB_Rating                                           Overview  Meta_score  \
0          9.3  Two imprisoned men bond over a number of years…        80.0
1          9.2  An organized crime dynasty's aging patriarch t…       100.0

               Director          Star1           Star2       Star3  \
0       Frank Darabont    Tim Robbins  Morgan Freeman  Bob Gunton
1  Francis Ford Coppola  Marlon Brando       Al Pacino  James Caan

            Star4  No_of_Votes        Gross
0  William Sadler      2343110   28,341,469
1    Diane Keaton      1620367  134,966,411
```

[3]:
```
df.columns
```

[3]:
```
Index(['Poster_Link', 'Series_Title', 'Released_Year', 'Certificate',
       'Runtime', 'Genre', 'IMDB_Rating', 'Overview', 'Meta_score', 'Director',
       'Star1', 'Star2', 'Star3', 'Star4', 'No_of_Votes', 'Gross'],
      dtype='object')
```

[4]:
```
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
```

[5]:
```
null = df.isnull().sum()
percentage = 100*null/len(df)
percentage
```

[5]:
```
Poster_Link       0.0
Series_Title      0.0
Released_Year     0.0
Certificate      10.1
Runtime           0.0
Genre             0.0
IMDB_Rating       0.0
Overview          0.0
Meta_score       15.7
Director          0.0
Star1             0.0
Star2             0.0
Star3             0.0
Star4             0.0
No_of_Votes       0.0
Gross            16.9
dtype: float64
```

[6]:
```
df = df.dropna()
```

[7]:
```
df.describe()
```

[7]:

	IMDB_Rating	Meta_score	No_of_Votes
count	714.000000	714.000000	7.140000e+02
mean	7.937115	77.158263	3.561348e+05
std	0.293278	12.401144	3.539011e+05
min	7.600000	28.000000	2.522900e+04
25%	7.700000	70.000000	9.600975e+04
50%	7.900000	78.000000	2.366025e+05
75%	8.100000	86.000000	5.077922e+05
max	9.300000	100.000000	2.343110e+06

[8]:
```
df.isnull().sum()
```

[8]:
```
Poster_Link      0
Series_Title     0
Released_Year    0
Certificate      0
Runtime          0
Genre            0
IMDB_Rating      0
Overview         0
Meta_score       0
Director         0
Star1            0
Star2            0
Star3            0
Star4            0
No_of_Votes      0
Gross            0
dtype: int64
```

[9]:
```
df = df[df['Released_Year'] != 'PG']
df['Released_Year'] = df['Released_Year'].astype(int)
```

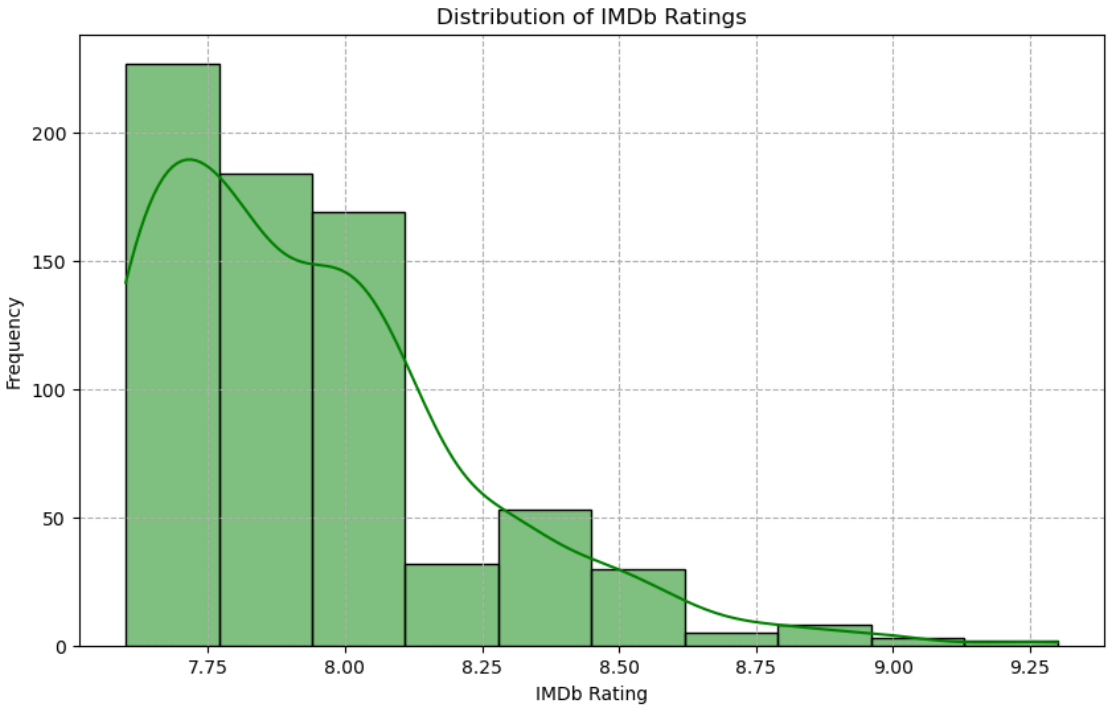
[10]:
```
df.shape
```

[10]:
```
(713, 16)
```

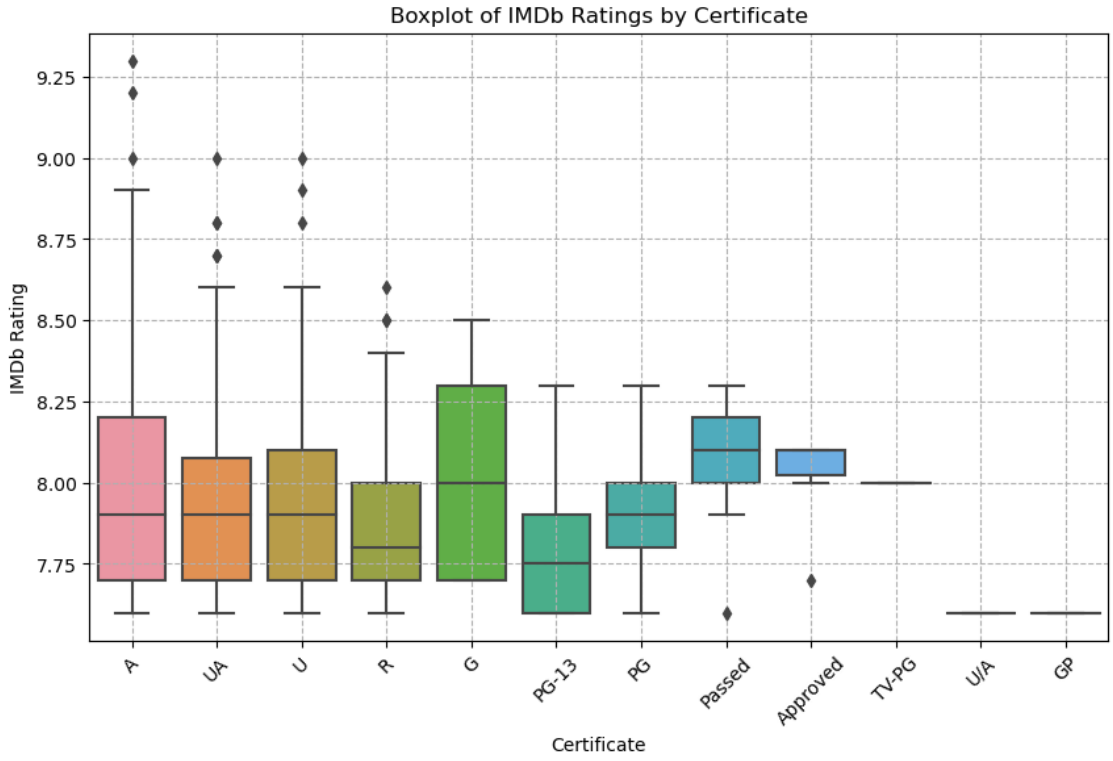
[11]:
```
df['Gross'] = df['Gross'].str.replace(',','').astype(float)
df['Runtime'] = df['Runtime'].str.replace('min','').astype(float)
```

[12]:
```
plt.figure(figsize=(10, 6))
sns.histplot(df['IMDB_Rating'], bins=10, kde=True, color='green')
plt.title('Distribution of IMDb Ratings')
```

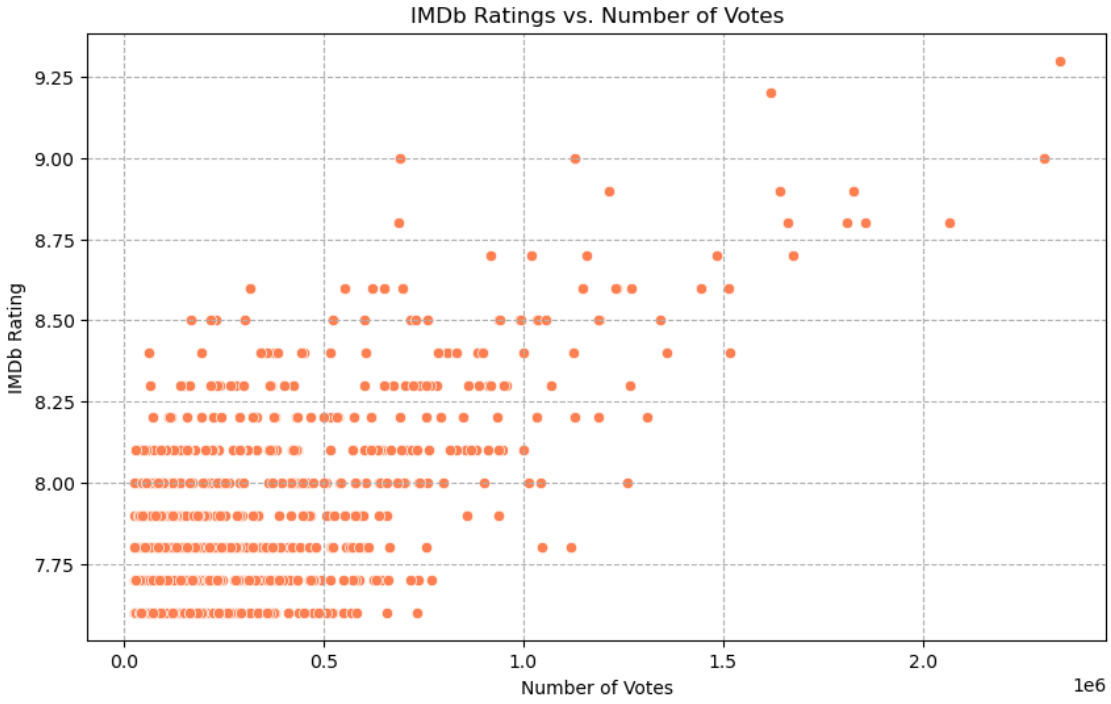
```
plt.xlabel('IMDb Rating')
plt.ylabel('Frequency')
plt.grid(linestyle='--')
plt.show()
```



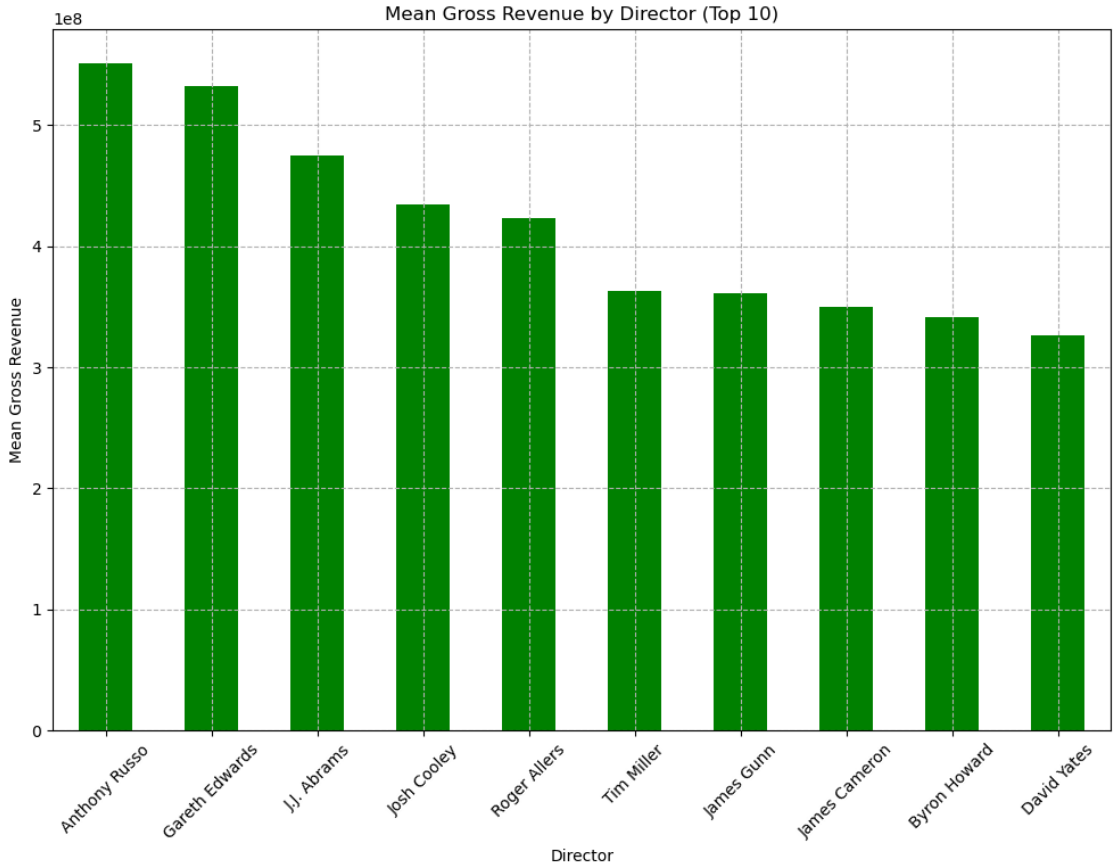
[13]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x='Certificate', y='IMDB_Rating')
plt.title('Boxplot of IMDb Ratings by Certificate')
plt.xlabel('Certificate')
plt.ylabel('IMDb Rating')
plt.grid(linestyle='--')
plt.xticks(rotation=45)
plt.show()
```



[14]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='No_of_Votes', y='IMDB_Rating', color='coral')
plt.title('IMDb Ratings vs. Number of Votes')
plt.xlabel('Number of Votes')
plt.ylabel('IMDb Rating')
plt.grid(linestyle='--')
plt.show()
```

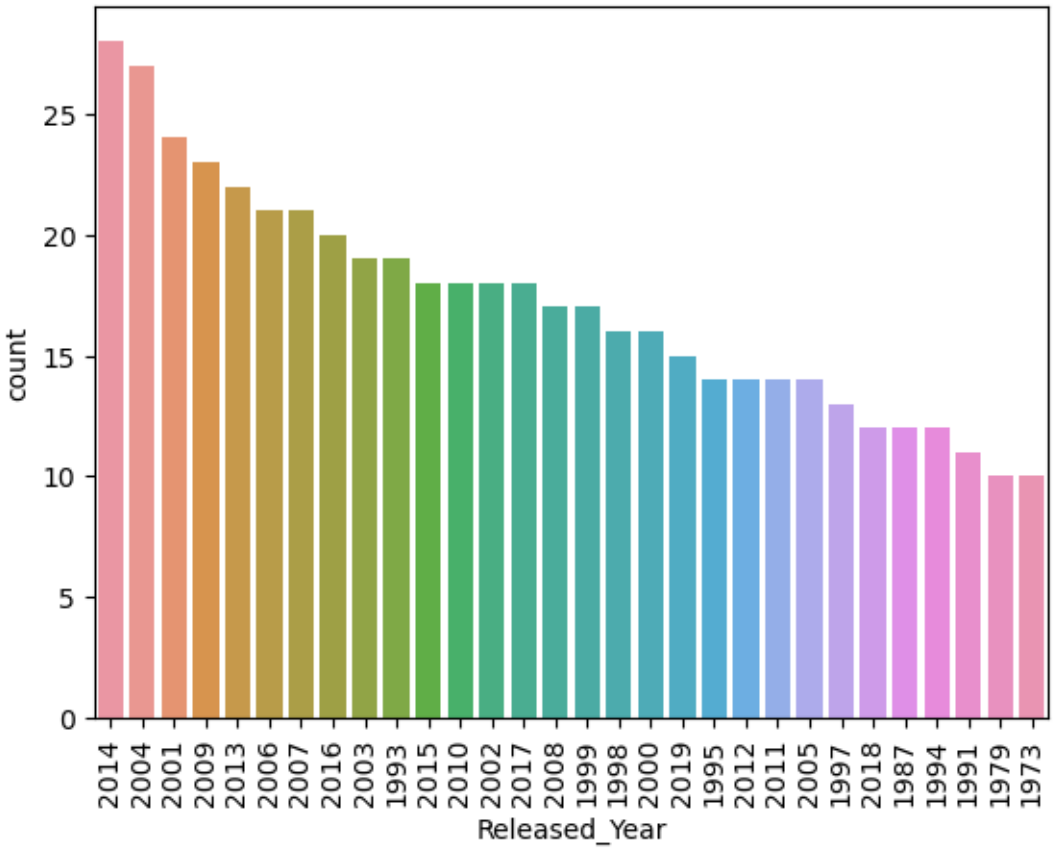


[15]:
```
top_directors = df.groupby('Director')['Gross'].mean().nlargest(10)
plt.figure(figsize=(12, 8))
top_directors.plot(kind='bar', color='green')
plt.title('Mean Gross Revenue by Director (Top 10)')
plt.xlabel('Director')
plt.ylabel('Mean Gross Revenue')
plt.xticks(rotation=45)
plt.grid(linestyle='--')
plt.show()
```

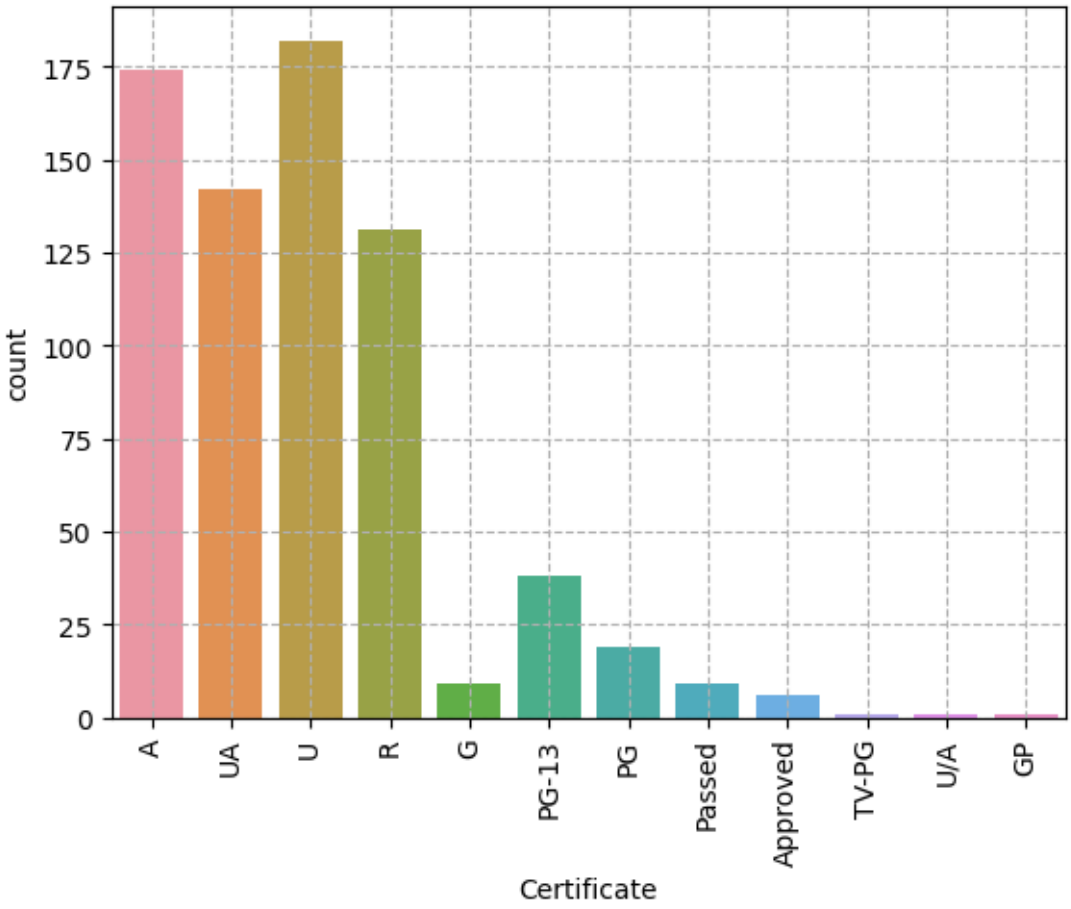


[16]:
```
year = df['Released_Year'].value_counts().head(30)
```

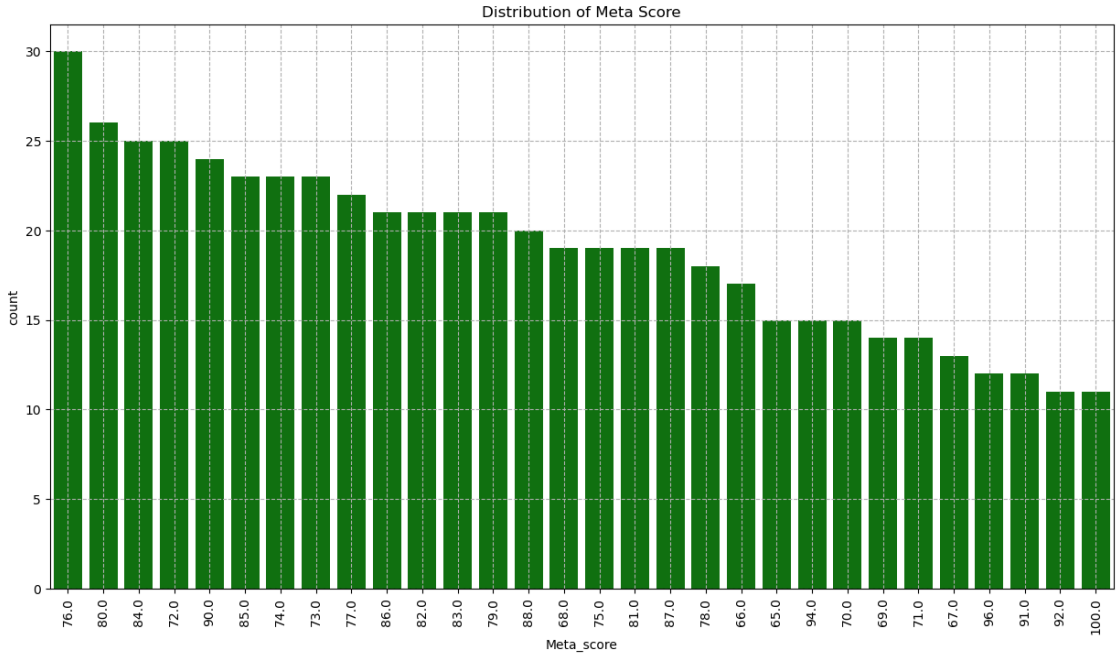
[17]:
```
year_counts = df['Released_Year'].value_counts().head(30)

sns.countplot(data=df, x='Released_Year', order=year_counts.index)
plt.xticks(rotation=90)
plt.show()
```



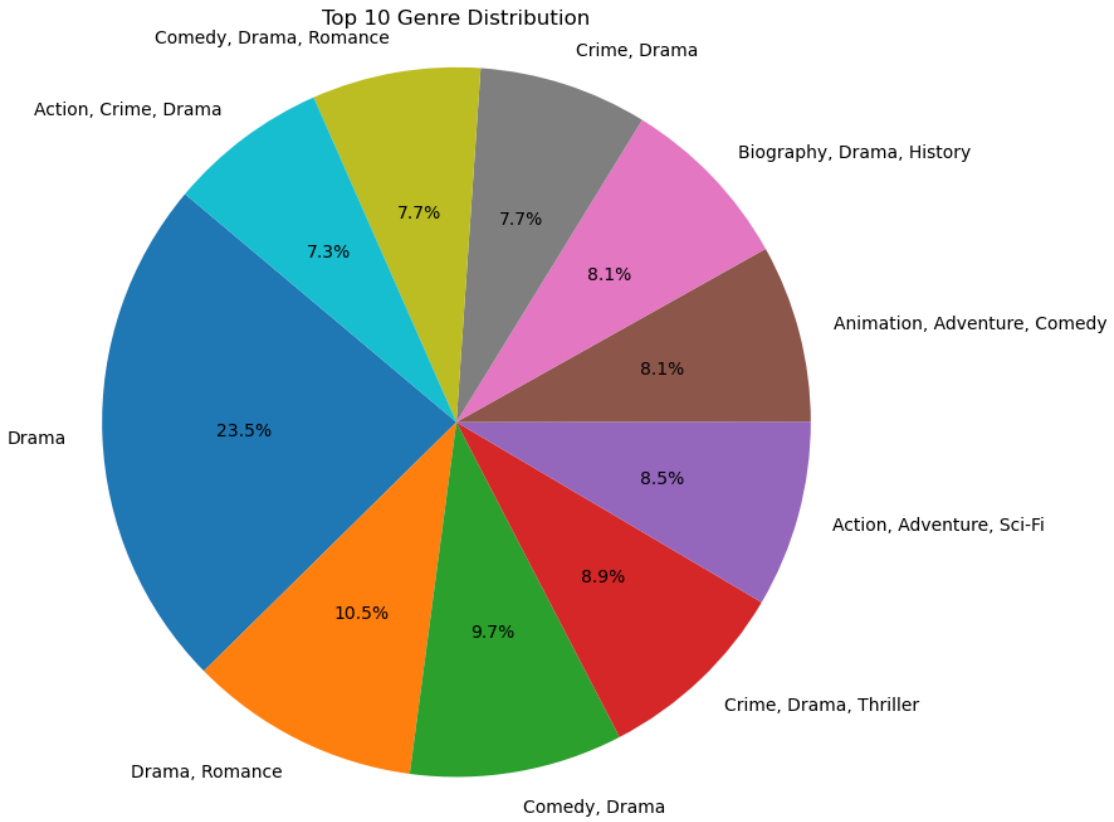
[18]:
```
sns.countplot(data=df, x='Certificate')
plt.xticks(rotation=90)
plt.grid(linestyle='--')
plt.show()
```



[19]:
```
meta = df['Meta_score'].value_counts().head(30)
plt.figure(figsize=(15,8))
sns.countplot(data=df, x='Meta_score',order = meta.index, color = 'green')
plt.xticks(rotation=90)
plt.grid(linestyle='--')
plt.title('Distribution of Meta Score')
plt.show()
```



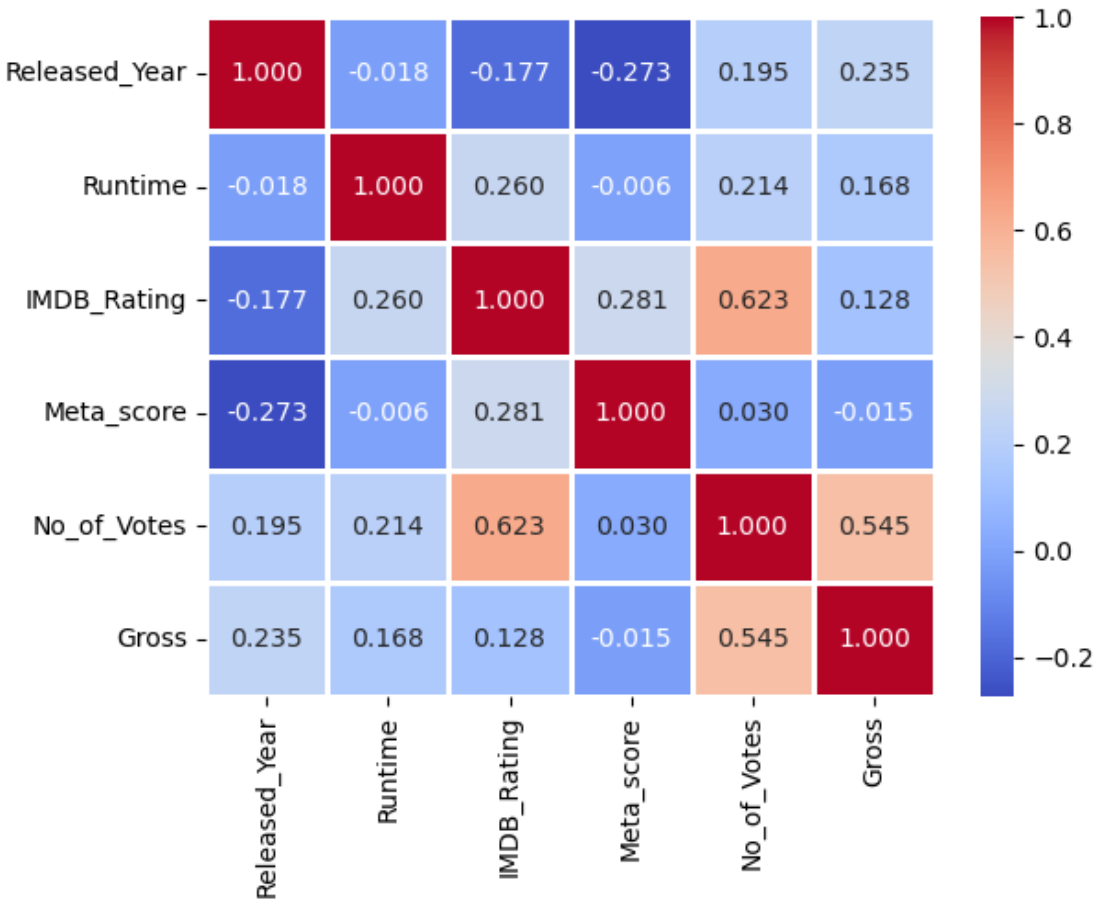
[20]:
```
genre_counts = df['Genre'].value_counts().head(10)
plt.figure(figsize=(8, 8))
plt.pie(genre_counts, labels=genre_counts.index, autopct='%1.1f%%',␣
 ↪startangle=140)
plt.title('Top 10 Genre Distribution')
plt.axis('equal')
plt.show()
```



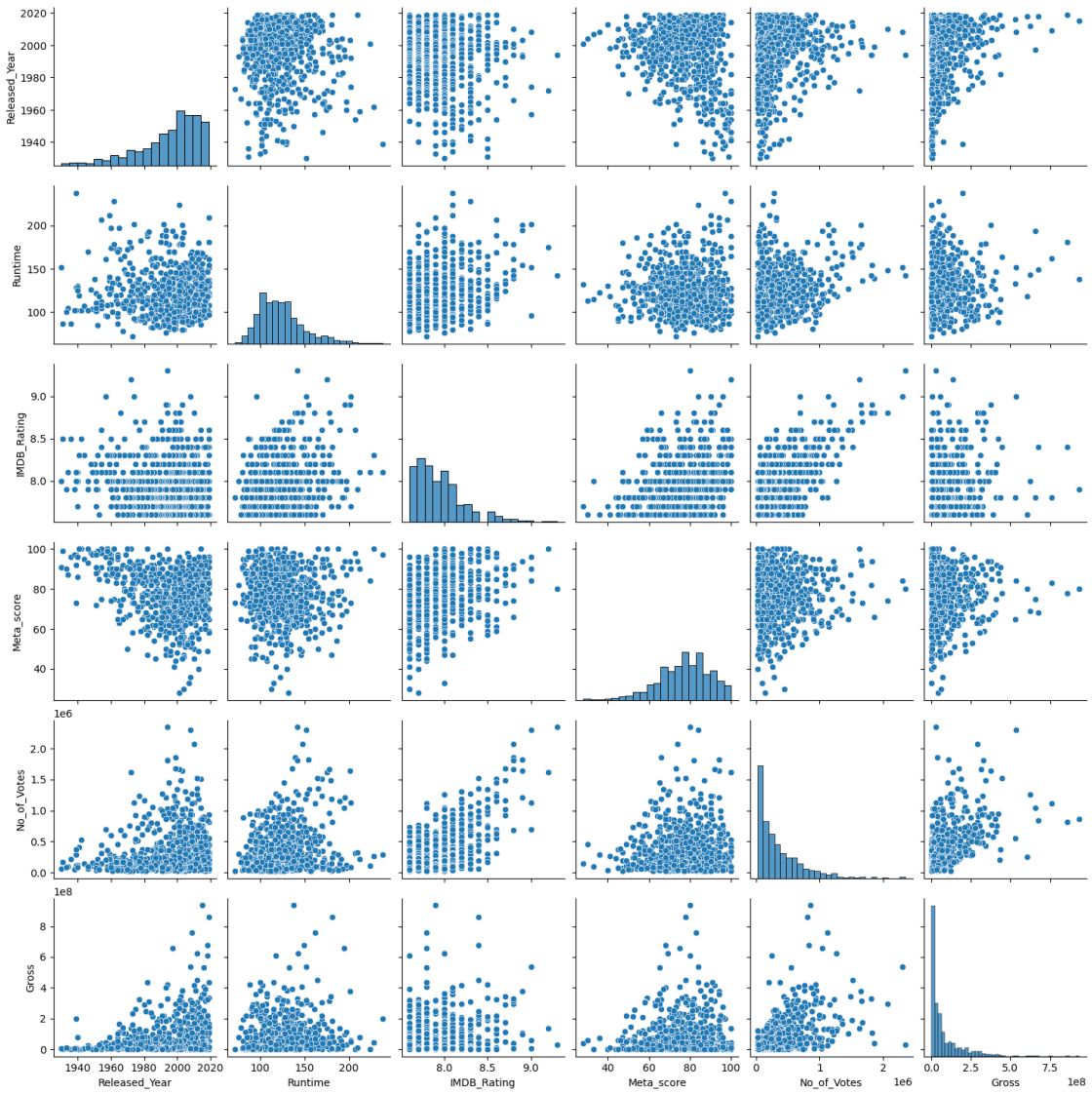
[21]:
```
sns.heatmap(df.corr(), annot=True, fmt='.3f', cmap='coolwarm', linewidths=1)
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_3428\3033028183.py:1: FutureWarning:
The default value of numeric_only in DataFrame.corr is deprecated. In a future
version, it will default to False. Select only valid columns or specify the
value of numeric_only to silence this warning.
  sns.heatmap(df.corr(), annot=True, fmt='.3f', cmap='coolwarm', linewidths=1)
```

[21]: <Axes: >



[22]:
```
sns.pairplot(df)
plt.grid()
```



2 TASK-3

[23]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
%matplotlib inline
```

[24]:
```
df = pd.read_csv('Iris.csv')
```

[25]:
```
df.head()
```

[25]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

[26]:
```
df.describe()
```

[26]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

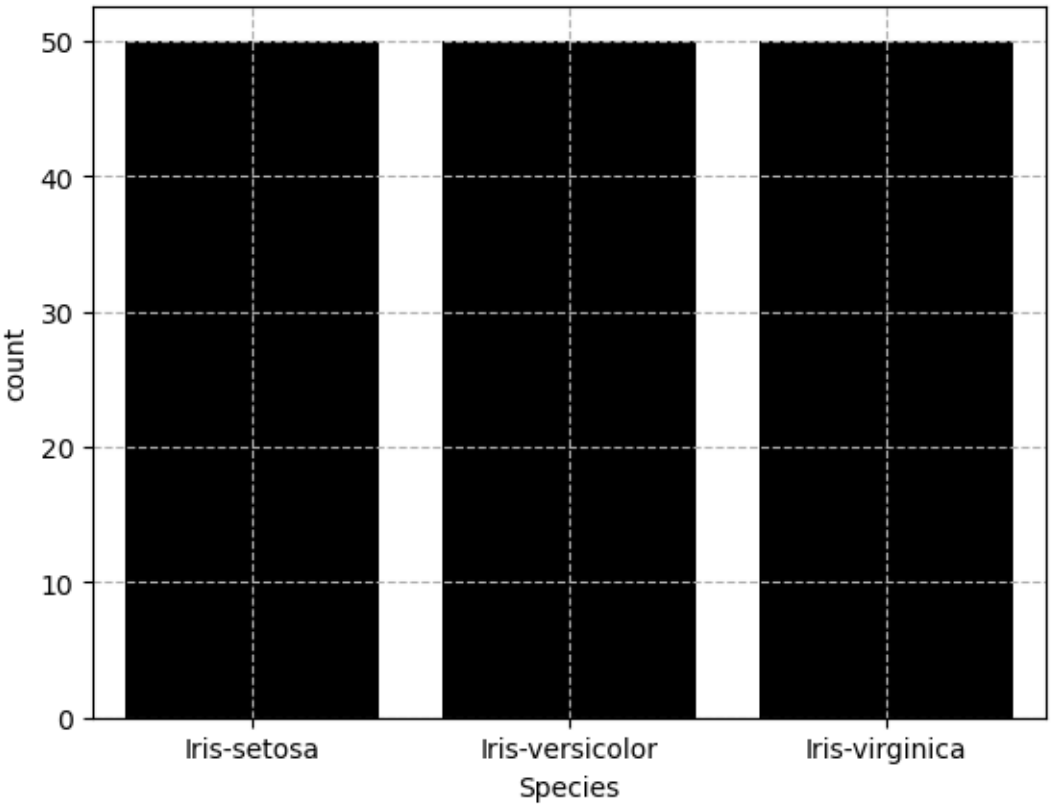
[27]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

[28]:
```
df['Species'].unique()
```

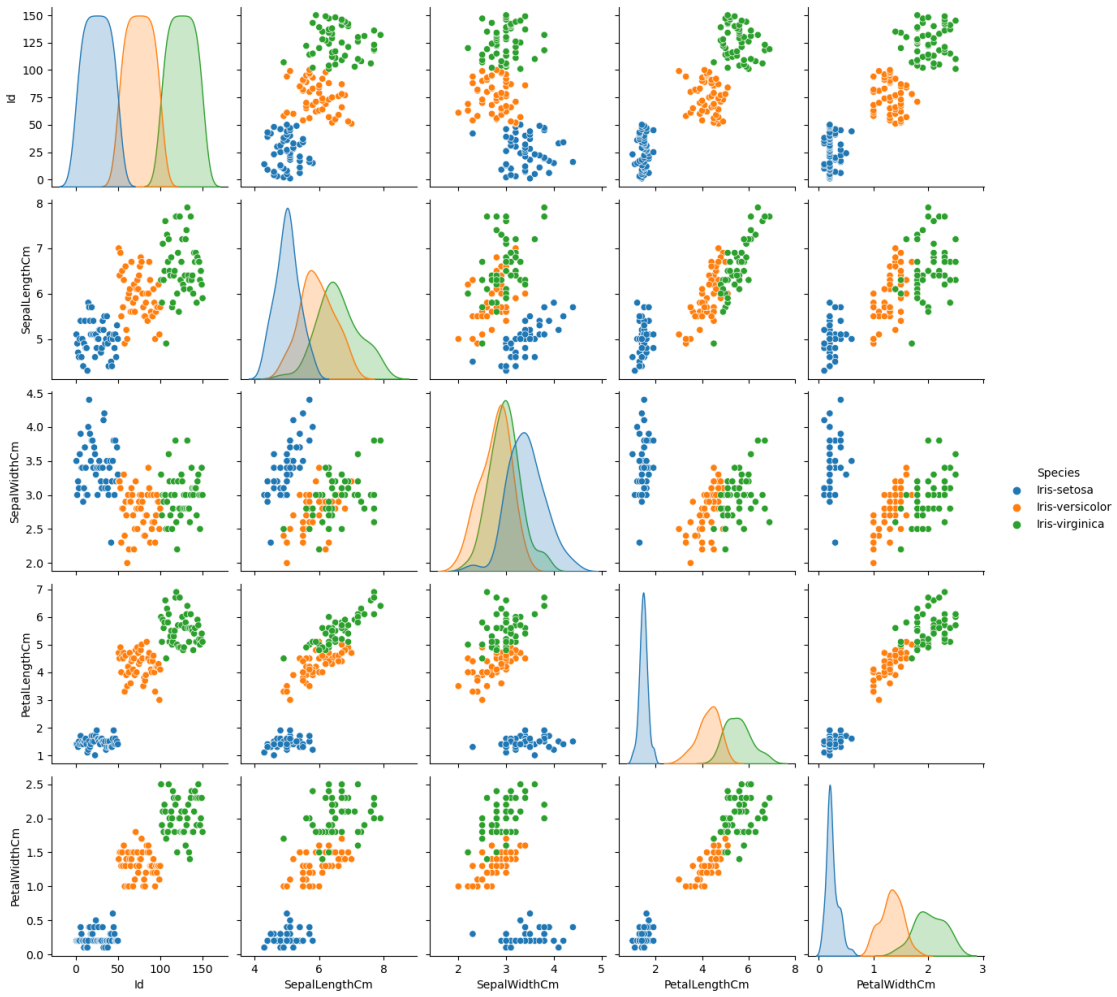
[28]:
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

[29]:
```
sns.countplot(data=df, x='Species',color='black')
plt.grid(linestyle='--')
plt.show()
```



[30]:
```
sns.pairplot(df, hue='Species')
```

[30]: <seaborn.axisgrid.PairGrid at 0x29f816933d0>



[31]:
```
x = df.drop('Species', axis=1)
y = df['Species']
```

[32]:
```
x.head()
```

[32]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	1	5.1	3.5	1.4	0.2
1	2	4.9	3.0	1.4	0.2
2	3	4.7	3.2	1.3	0.2
3	4	4.6	3.1	1.5	0.2
4	5	5.0	3.6	1.4	0.2

[33]:
```
y.head()
```

[33]:
```
0    Iris-setosa
1    Iris-setosa
```

```
2    Iris-setosa
3    Iris-setosa
4    Iris-setosa
Name: Species, dtype: object
```

[34]:
```
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_state=101)
print('x_train', x_train.shape)
print('x_test', x_test.shape)
print('y_train', y_train.shape)
print('y_test', y_test.shape)
```

```
x_train (120, 5)
x_test (30, 5)
y_train (120,)
y_test (30,)
```

[35]:
```
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

[36]:
```
models = {
    ('svm',SVC()),
    ('LogisticRegression',LogisticRegression()),
    ('DecisionTreeClassifier',DecisionTreeClassifier()),
    ('RandomForestClassifier',RandomForestClassifier())

}
models
```

[36]:
```
{('DecisionTreeClassifier', DecisionTreeClassifier()),
 ('LogisticRegression', LogisticRegression()),
 ('RandomForestClassifier', RandomForestClassifier()),
 ('svm', SVC())}
```

[37]:
```
from sklearn.metrics import accuracy_score, mean_squared_error,␣
 ↪classification_report
```

[38]:
```
for model_name, model in models:
    model.fit(x_train,y_train)
    predict = model.predict(x_test)
    accuracy = accuracy_score(y_test, predict)

    print(f'\nmodel = {model_name}\n')
    print(f'accuracy = {accuracy}')
```

```
    print(f'CFR = {classification_report(y_test, predict)}')
```

```
model = svm

accuracy = 1.0
CFR =                   precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00        12
 Iris-virginica       1.00      1.00      1.00         8

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30


model = LogisticRegression

accuracy = 1.0
CFR =                   precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00        12
 Iris-virginica       1.00      1.00      1.00         8

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30


model = RandomForestClassifier

accuracy = 1.0
CFR =                   precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00        12
 Iris-virginica       1.00      1.00      1.00         8

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30


model = DecisionTreeClassifier
```

```
accuracy = 0.9666666666666667
CFR =                   precision    recall  f1-score   support

    Iris-setosa       0.91      1.00      0.95        10
Iris-versicolor       1.00      0.92      0.96        12
 Iris-virginica       1.00      1.00      1.00         8

       accuracy                           0.97        30
      macro avg       0.97      0.97      0.97        30
   weighted avg       0.97      0.97      0.97        30


C:\Users\Admin\anaconda3\Lib\site-
packages\sklearn\linear_model\_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

2.1 The accuracy score of iris flower prediction in nearly 1 to all model

3 TASK 5

[39]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report,\
precision_score, recall_score
```

[40]:
```
df = pd.read_csv('creditcard.csv')
```

[41]:
```
df.head()
```

[41]:
```
   Time        V1        V2        V3        V4        V5        V6        V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9  …       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  … -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  … -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  …  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  … -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  … -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
```

[42]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
```

```
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

[43]:
```
df.isnull().sum()
```

[43]:
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
```

```
V28       0
Amount    0
Class     0
dtype: int64
```

[44]:
```
df.describe().T
```

[44]:

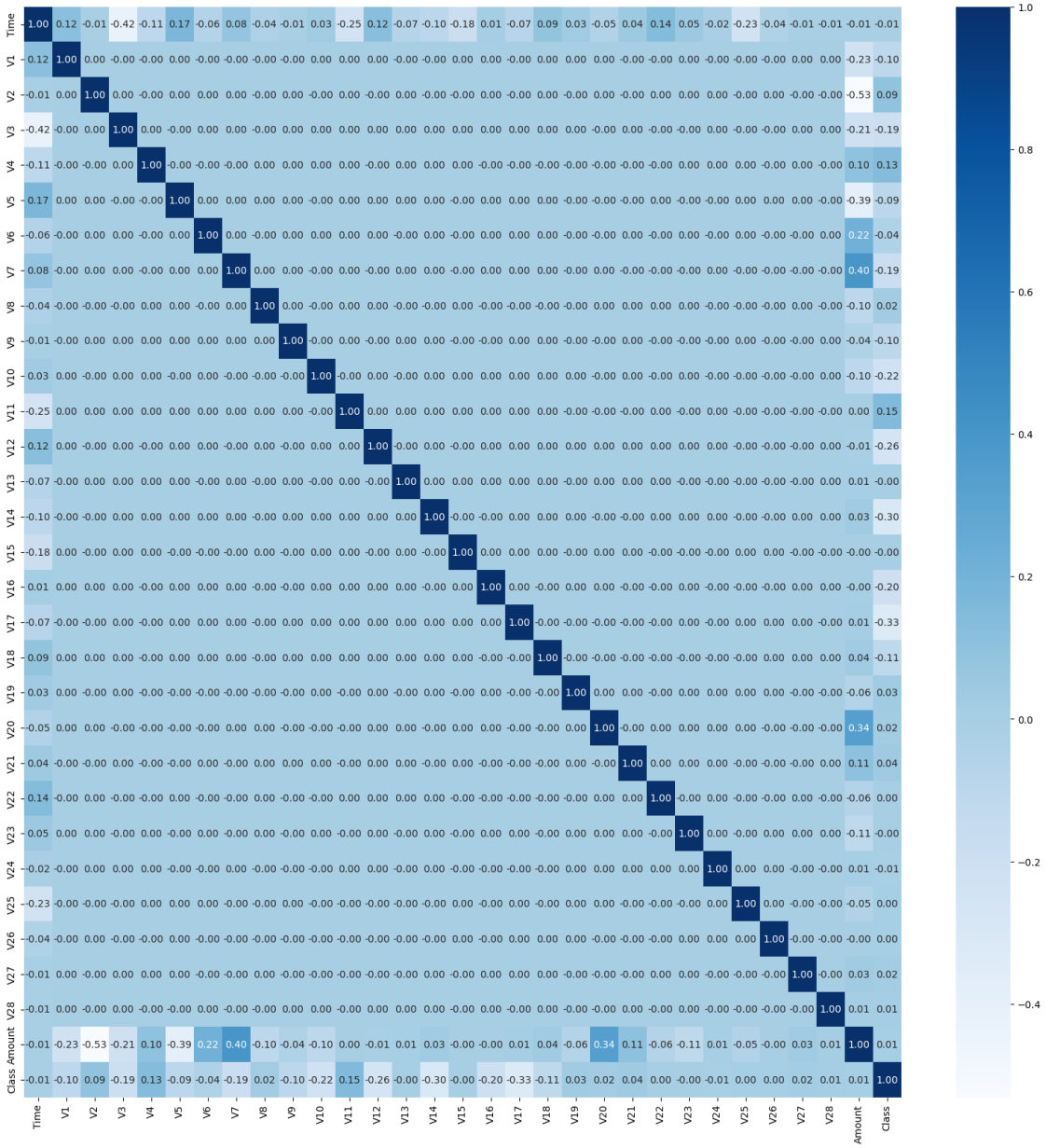
	count	mean	std	min	25%	\
Time	284807.0	9.481386e+04	47488.145955	0.000000	54201.500000	
V1	284807.0	1.168375e-15	1.958696	-56.407510	-0.920373	
V2	284807.0	3.416908e-16	1.651309	-72.715728	-0.598550	
V3	284807.0	-1.379537e-15	1.516255	-48.325589	-0.890365	
V4	284807.0	2.074095e-15	1.415869	-5.683171	-0.848640	
V5	284807.0	9.604066e-16	1.380247	-113.743307	-0.691597	
V6	284807.0	1.487313e-15	1.332271	-26.160506	-0.768296	
V7	284807.0	-5.556467e-16	1.237094	-43.557242	-0.554076	
V8	284807.0	1.213481e-16	1.194353	-73.216718	-0.208630	
V9	284807.0	-2.406331e-15	1.098632	-13.434066	-0.643098	
V10	284807.0	2.239053e-15	1.088850	-24.588262	-0.535426	
V11	284807.0	1.673327e-15	1.020713	-4.797473	-0.762494	
V12	284807.0	-1.247012e-15	0.999201	-18.683715	-0.405571	
V13	284807.0	8.190001e-16	0.995274	-5.791881	-0.648539	
V14	284807.0	1.207294e-15	0.958596	-19.214325	-0.425574	
V15	284807.0	4.887456e-15	0.915316	-4.498945	-0.582884	
V16	284807.0	1.437716e-15	0.876253	-14.129855	-0.468037	
V17	284807.0	-3.772171e-16	0.849337	-25.162799	-0.483748	
V18	284807.0	9.564149e-16	0.838176	-9.498746	-0.498850	
V19	284807.0	1.039917e-15	0.814041	-7.213527	-0.456299	
V20	284807.0	6.406204e-16	0.770925	-54.497720	-0.211721	
V21	284807.0	1.654067e-16	0.734524	-34.830382	-0.228395	
V22	284807.0	-3.568593e-16	0.725702	-10.933144	-0.542350	
V23	284807.0	2.578648e-16	0.624460	-44.807735	-0.161846	
V24	284807.0	4.473266e-15	0.605647	-2.836627	-0.354586	
V25	284807.0	5.340915e-16	0.521278	-10.295397	-0.317145	
V26	284807.0	1.683437e-15	0.482227	-2.604551	-0.326984	
V27	284807.0	-3.660091e-16	0.403632	-22.565679	-0.070840	
V28	284807.0	-1.227390e-16	0.330083	-15.430084	-0.052960	
Amount	284807.0	8.834962e+01	250.120109	0.000000	5.600000	
Class	284807.0	1.727486e-03	0.041527	0.000000	0.000000	

	50%	75%	max
Time	84692.000000	139320.500000	172792.000000
V1	0.018109	1.315642	2.454930
V2	0.065486	0.803724	22.057729
V3	0.179846	1.027196	9.382558
V4	-0.019847	0.743341	16.875344
V5	-0.054336	0.611926	34.801666

```
V6          -0.274187       0.398565      73.301626
V7           0.040103       0.570436     120.589494
V8           0.022358       0.327346      20.007208
V9          -0.051429       0.597139      15.594995
V10         -0.092917       0.453923      23.745136
V11         -0.032757       0.739593      12.018913
V12          0.140033       0.618238       7.848392
V13         -0.013568       0.662505       7.126883
V14          0.050601       0.493150      10.526766
V15          0.048072       0.648821       8.877742
V16          0.066413       0.523296      17.315112
V17         -0.065676       0.399675       9.253526
V18         -0.003636       0.500807       5.041069
V19          0.003735       0.458949       5.591971
V20         -0.062481       0.133041      39.420904
V21         -0.029450       0.186377      27.202839
V22          0.006782       0.528554      10.503090
V23         -0.011193       0.147642      22.528412
V24          0.040976       0.439527       4.584549
V25          0.016594       0.350716       7.519589
V26         -0.052139       0.240952       3.517346
V27          0.001342       0.091045      31.612198
V28          0.011244       0.078280      33.847808
Amount      22.000000      77.165000   25691.160000
Class        0.000000       0.000000       1.000000
```

[45]:
```
plt.figure(figsize=(20,20))
sns.heatmap(df.corr(), annot=True, fmt='.2f', cmap='Blues')
```

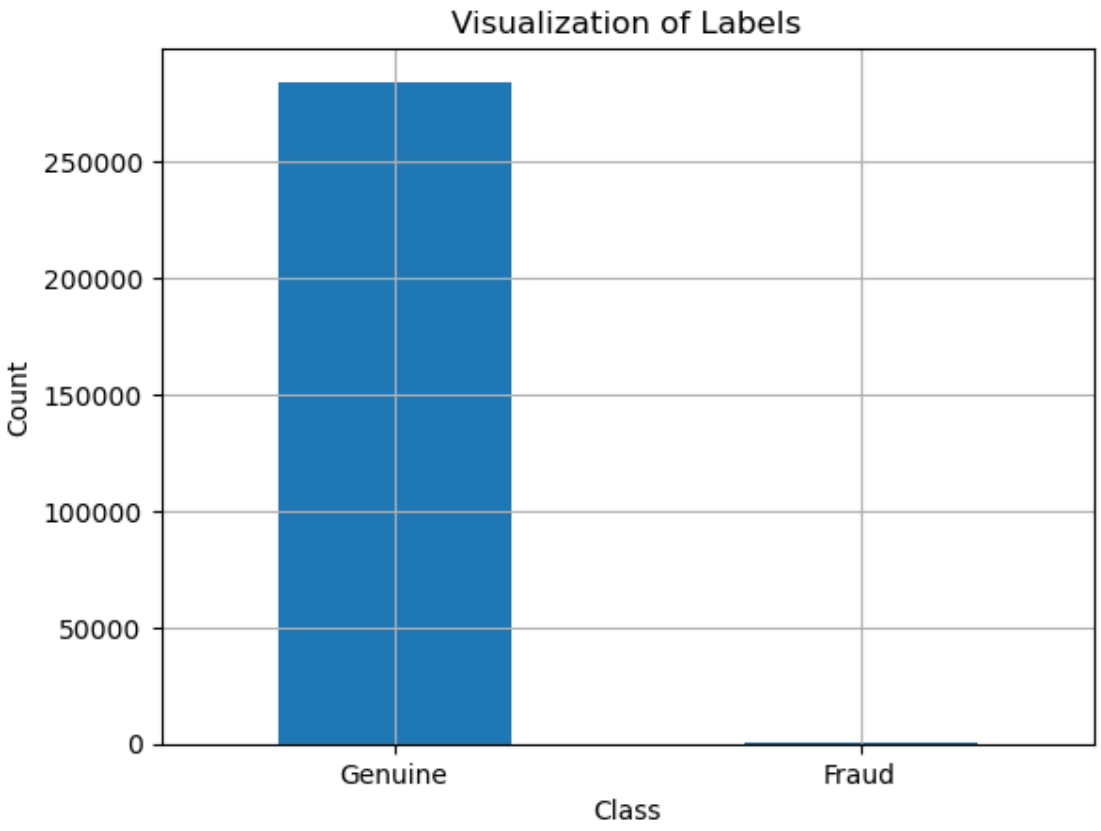
[45]: <Axes: >



[46]:
```
non_fraud = len(df[df.Class == 0])
fraud = len(df[df.Class == 1])
percentage = (fraud/(fraud+non_fraud))*100
print(f'Genuine transaction: {non_fraud}')
print(f'Fraud Transaction: {fraud}')
print('Percentage of fraud transaction:{:.4f}%'.format(percentage))
```

```
Genuine transaction: 284315
Fraud Transaction: 492
Percentage of fraud transaction:0.1727%
```

[47]:
```
labels = ['Genuine','Fraud']
class_count = df.value_counts(df['Class'], sort = True)
class_count.plot(kind='bar', rot=0)
plt.title("Visualization of Labels")
plt.ylabel("Count")
plt.xticks(range(2), labels)
plt.grid()
plt.show()
```



[48]:
```
df.columns
```

[48]:
```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

[49]:
```
df.drop(columns=['Time','Amount'], axis=1, inplace=True)
```

[50]:
```
x = df.drop('Class', axis=1)
y = df['Class']
```

[51]:
```
x.head()
```

[51]:
```
          V1        V2        V3        V4        V5        V6        V7  \
0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

          V8        V9       V10  …       V19       V20       V21       V22  \
0  0.098698  0.363787  0.090794  …  0.403993  0.251412 -0.018307  0.277838
1  0.085102 -0.255425 -0.166974  … -0.145783 -0.069083 -0.225775 -0.638672
2  0.247676 -1.514654  0.207643  … -2.261857  0.524980  0.247998  0.771679
3  0.377436 -1.387024 -0.054952  … -1.232622 -0.208038 -0.108300  0.005274
4 -0.270533  0.817739  0.753074  …  0.803487  0.408542 -0.009431  0.798278

        V23       V24       V25       V26       V27       V28
0 -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053
1  0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724
2  0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752
3 -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458
4 -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153

[5 rows x 28 columns]
```

[52]:
```
y.head()
```

[52]:
```
0    0
1    0
2    0
3    0
4    0
Name: Class, dtype: int64
```

[53]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.
↪2,random_state=101)
print('x_train', x_train.shape)
print('x_test', x_test.shape)
print('y_train', y_train.shape)
print('y_test', y_test.shape)
```

```
x_train (227845, 28)
x_test (56962, 28)
y_train (227845,)
```

```
y_test (56962,)
```

[54]:
```
model1 = LogisticRegression()
model1.fit(x_train, y_train)
y_pred = model1.predict(x_test)

print('accuracy:', accuracy_score(y_test, y_pred))
print('conf_matrix\n', confusion_matrix(y_test, y_pred))
print('pre_score:', precision_score(y_test, y_pred))
print('class_report\n', classification_report(y_test, y_pred))
print('re_score:', recall_score(y_test, y_pred))
```

```
accuracy: 0.9992275552122467
conf_matrix
 [[56853     6]
 [   38    65]]
pre_score: 0.9154929577464789
class_report
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     56859
           1       0.92      0.63      0.75       103

    accuracy                           1.00     56962
   macro avg       0.96      0.82      0.87     56962
weighted avg       1.00      1.00      1.00     56962

re_score: 0.6310679611650486
```

[55]:
```
model1 = RandomForestClassifier()
model1.fit(x_train, y_train)
y_pred = model1.predict(x_test)

print('accuracy:', accuracy_score(y_test, y_pred))
print('conf_matrix\n', confusion_matrix(y_test, y_pred))
print('pre_score:', precision_score(y_test, y_pred))
print('class_report\n', classification_report(y_test, y_pred))
print('re_score:', recall_score(y_test, y_pred))
```

```
accuracy: 0.9995611109160493
conf_matrix
 [[56853     6]
 [   19    84]]
pre_score: 0.9333333333333333
class_report
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     56859
```

```
           1       0.93      0.82      0.87       103

    accuracy                           1.00     56962
   macro avg       0.97      0.91      0.94     56962
weighted avg       1.00      1.00      1.00     56962

re_score: 0.8155339805825242
```

[56]:
```
model1 = DecisionTreeClassifier()
model1.fit(x_train, y_train)
y_pred = model1.predict(x_test)

print('accuracy:', accuracy_score(y_test, y_pred))
print('conf_matrix\n', confusion_matrix(y_test, y_pred))
print('pre_score:', precision_score(y_test, y_pred))
print('class_report\n', classification_report(y_test, y_pred))
print('re_score:', recall_score(y_test, y_pred))
```

```
accuracy: 0.9991222218320986
conf_matrix
 [[56832    27]
 [   23    80]]
pre_score: 0.7476635514018691
class_report
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     56859
           1       0.75      0.78      0.76       103

    accuracy                           1.00     56962
   macro avg       0.87      0.89      0.88     56962
weighted avg       1.00      1.00      1.00     56962

re_score: 0.7766990291262136
```

[ ]: