Bharat Intern - Success of an upcoming movie

In [1]:

```
import pandas as pd
data = pd.read_csv("movie_success_rate.csv")
```

In [2]:

```
data.head()
```

Out[2]:

	Rank	Title	Genre	Description	Director	Actors	Year	Runtime (Minutes)	Rating	Votes	...
0	1.0	Guardians of the Galaxy	Action,Adventure,Sci-Fi	A group of intergalactic criminals are forced ...	James Gunn	Chris Pratt, Vin Diesel, Bradley Cooper, Zoe S...	2014.0	121.0	8.1	757074.0	...
1	2.0	Prometheus	Adventure,Mystery,Sci-Fi	Following clues to the origin of mankind, a te...	Ridley Scott	Noomi Rapace, Logan Marshall-Green, Michael Fa...	2012.0	124.0	7.0	485820.0	...
2	3.0	Split	Horror,Thriller	Three girls are kidnapped by a man with a diag...	M. Night Shyamalan	James McAvoy, Anya Taylor-Joy, Haley Lu Richar...	2016.0	117.0	7.3	157606.0	...
3	4.0	Sing	Animation,Comedy,Family	In a city of humanoid animals, a hustling thea...	Christophe Lourdelet	Matthew McConaughey,Reese Witherspoon, Seth Ma...	2016.0	108.0	7.2	60545.0	...
4	5.0	Suicide Squad	Action,Adventure,Fantasy	A secret government agency recruits some of th...	David Ayer	Will Smith, Jared Leto, Margot Robbie, Viola D...	2016.0	123.0	6.2	393727.0	...

5 rows × 33 columns

In [3]:

```
data.tail()
```

Out[3]:

	Rank	Title	Genre	Description	Director	Actors	Year	Runtime (Minutes)	Rating	Votes	...	Music
834	995.0	Project X	Comedy	3 high school seniors throw a birthday party t...	Nima Nourizadeh	Thomas Mann, Oliver Cooper, Jonathan Daniel Br...	2012.0	88.0	6.70000	164088.0000	...	0.0
835	997.0	Hostel: Part II	Horror	Three American college students studying abroa...	Eli Roth	Lauren German, Heather Matarazzo, Bijou Philli...	2007.0	94.0	5.50000	73152.0000	...	0.0
836	998.0	Step Up 2: The Streets	Drama,Music,Romance	Romantic sparks occur between two dance studen...	Jon M. Chu	Robert Hoffman, Briana Evigan, Cassie Ventura,...	2008.0	98.0	6.20000	70699.0000	...	1.0
837	1000.0	Nine Lives	Comedy,Family,Fantasy	A stuffy businessman finds himself trapped ins...	Barry Sonnenfeld	Kevin Spacey, Jennifer Garner, Robbie Amell,Ch...	2016.0	87.0	5.30000	12435.0000	...	0.0
838	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	6.81432	193230.2518	...	NaN

5 rows × 33 columns

Data Description

In [4]:

```
data.columns
```

Out[4]:

```
Index(['Rank', 'Title', 'Genre', 'Description', 'Director', 'Actors', 'Year',
       'Runtime (Minutes)', 'Rating', 'Votes', 'Revenue (Millions)',
       'Metascore', 'Action', 'Adventure', 'Aniimation', 'Biography', 'Comedy',
       'Crime', 'Drama', 'Family', 'Fantasy', 'History', 'Horror', 'Music',
       'Musical', 'Mystery', 'Romance', 'Sci-Fi', 'Sport', 'Thriller', 'War',
       'Western', 'Success'],
      dtype='object')
```

In [5]:

```
data.values
```

Out[5]:

```
array([[1.0, 'Guardians of the Galaxy', 'Action,Adventure,Sci-Fi', ...,
        0.0, 0.0, 1.0],
       [2.0, 'Prometheus', 'Adventure,Mystery,Sci-Fi', ..., 0.0, 0.0,
        1.0],
       [3.0, 'Split', 'Horror,Thriller', ..., 0.0, 0.0, 0.0],
       ...,
       [998.0, 'Step Up 2: The Streets', 'Drama,Music,Romance', ..., 0.0,
        0.0, 0.0],
       [1000.0, 'Nine Lives', 'Comedy,Family,Fantasy', ..., 0.0, 0.0,
        0.0],
       [nan, nan, nan, ..., nan, nan, nan]], dtype=object)
```

In [6]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 839 entries, 0 to 838
Data columns (total 33 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Rank                838 non-null    float64
 1   Title               838 non-null    object 
 2   Genre               838 non-null    object 
 3   Description         838 non-null    object 
 4   Director            838 non-null    object 
 5   Actors              838 non-null    object 
 6   Year                838 non-null    float64
 7   Runtime (Minutes)   838 non-null    float64
 8   Rating              839 non-null    float64
 9   Votes               839 non-null    float64
 10  Revenue (Millions)  839 non-null    float64
 11  Metascore           838 non-null    float64
 12  Action              838 non-null    float64
 13  Adventure           838 non-null    float64
 14  Aniimation          838 non-null    float64
 15  Biography           838 non-null    float64
 16  Comedy              838 non-null    float64
 17  Crime               838 non-null    float64
 18  Drama               838 non-null    float64
 19  Family              838 non-null    float64
 20  Fantasy             838 non-null    float64
 21  History             838 non-null    float64
 22  Horror              838 non-null    float64
 23  Music               838 non-null    float64
 24  Musical             838 non-null    float64
 25  Mystery             838 non-null    float64
 26  Romance             838 non-null    float64
 27  Sci-Fi              838 non-null    float64
 28  Sport               838 non-null    float64
 29  Thriller            838 non-null    float64
 30  War                 838 non-null    float64
 31  Western             838 non-null    float64
 32  Success             838 non-null    float64
dtypes: float64(28), object(5)
memory usage: 216.4+ KB
```

In [7]:

```
data.nunique()
```

Out[7]:

```
Rank                  838
Title                 837
Genre                 189
Description           838
Director              524
Actors                834
Year                   11
Runtime (Minutes)      90
Rating                 51
Votes                 838
Revenue (Millions)    790
Metascore              82
Action                  2
Adventure               2
Aniimation              2
Biography               2
Comedy                  2
Crime                   2
Drama                   2
Family                  2
Fantasy                 2
History                 2
Horror                  2
Music                   2
Musical                 2
Mystery                 2
Romance                 2
Sci-Fi                  2
Sport                   2
Thriller                2
War                     2
Western                 2
Success                 2
dtype: int64
```

In [8]:

```
data.describe()
```

Out[8]:

	Rank	Year	Runtime (Minutes)	Rating	Votes	Revenue (Millions)	Metascore	Action	Adventure	Aniimation
count	838.000000	838.00000	838.000000	839.00000	8.390000e+02	839.000000	838.000000	838.000000	838.000000	838.000000
mean	485.247017	2012.50716	114.638425	6.81432	1.932303e+05	84.564558	59.575179	0.330549	0.291169	0.053699
std	286.572065	3.17236	18.470922	0.87723	1.929838e+05	104.457845	16.952416	0.470692	0.454573	0.225558
min	1.000000	2006.00000	66.000000	1.90000	1.780000e+02	0.000000	11.000000	0.000000	0.000000	0.000000
25%	238.250000	2010.00000	101.000000	6.30000	6.145500e+04	13.975000	47.000000	0.000000	0.000000	0.000000
50%	475.500000	2013.00000	112.000000	6.90000	1.371170e+05	48.240000	60.000000	0.000000	0.000000	0.000000
75%	729.750000	2015.00000	124.000000	7.50000	2.708650e+05	116.730000	72.000000	1.000000	1.000000	0.000000
max	1000.000000	2016.00000	187.000000	9.00000	1.791916e+06	936.630000	100.000000	1.000000	1.000000	1.000000

8 rows × 28 columns

In [9]:

```
data.shape
```

Out[9]:

```
(839, 33)
```

Data Cleaning

In [10]:

```
data.isnull()
```

Out[10]:

	Rank	Title	Genre	Description	Director	Actors	Year	Runtime (Minutes)	Rating	Votes	...	Music	Musical	Mystery	Romance	S
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	Fal
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	Fal
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	Fal
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	Fal
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	Fal
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
834	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	Fal
835	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	Fal
836	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	Fal
837	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	Fal
838	True	True	True	True	True	True	True	True	False	False	...	True	True	True	True	Tr

839 rows × 33 columns

In [11]:

```
data.isnull().sum()
```

Out[11]:

```
Rank                  1
Title                 1
Genre                 1
Description           1
Director              1
Actors                1
Year                  1
Runtime (Minutes)     1
Rating                0
Votes                 0
Revenue (Millions)    0
Metascore             1
Action                1
Adventure             1
Aniimation            1
Biography             1
Comedy                1
Crime                 1
Drama                 1
Family                1
Fantasy               1
History               1
Horror                1
Music                 1
Musical               1
Mystery               1
Romance               1
Sci-Fi                1
Sport                 1
Thriller              1
War                   1
Western               1
Success               1
dtype: int64
```

In [12]:

```
data.dropna(inplace = True)
data.shape
```

Out[12]:

```
(838, 33)
```

In [13]:

```
data.rename(columns={'Aniimation' : 'Animation'},inplace = True)
```

In [14]:

```
numerical_vars=[]
categorical_vars=[]

for var in data.columns:
    if data[var].dtype in ['int64','float64']:
        numerical_vars.append(var)
    elif data[var].dtype in ['object']:
        categorical_vars.append(var)

print('Numerical Variables: ')
print(numerical_vars)
print('Categorical Variables: ')
print(categorical_vars)
```

```
Numerical Variables:
['Rank', 'Year', 'Runtime (Minutes)', 'Rating', 'Votes', 'Revenue (Millions)', 'Metascore', 'Action', 'Ad
venture', 'Animation', 'Biography', 'Comedy', 'Crime', 'Drama', 'Family', 'Fantasy', 'History', 'Horror',
'Music', 'Musical', 'Mystery', 'Romance', 'Sci-Fi', 'Sport', 'Thriller', 'War', 'Western', 'Success']
Categorical Variables:
['Title', 'Genre', 'Description', 'Director', 'Actors']
```

In [15]:

```
numerical=['Rank', 'Year', 'Runtime (Minutes)', 'Votes', 'Metascore']
data[numerical]= data[numerical].astype(int)

decimal= ['Rating', 'Revenue (Millions)']
data[decimal]=data[decimal].astype(float)

boolean = ['Action', 'Adventure', 'Animation', 'Biography', 'Comedy', 'Crime', 'Drama', 'Family', 'Fantasy', 'History',
data[boolean]=data[boolean].astype(bool)

print(data.dtypes)
```

```
Rank                    int32
Title                  object
Genre                  object
Description            object
Director               object
Actors                 object
Year                    int32
Runtime (Minutes)       int32
Rating                float64
Votes                   int32
Revenue (Millions)    float64
Metascore               int32
Action                   bool
Adventure                bool
Animation                bool
Biography                bool
Comedy                   bool
Crime                    bool
Drama                    bool
Family                   bool
Fantasy                  bool
History                  bool
Horror                   bool
Music                    bool
Musical                  bool
Mystery                  bool
Romance                  bool
Sci-Fi                   bool
Sport                    bool
Thriller                 bool
War                      bool
Western                  bool
Success                  bool
dtype: object
```

EDA

In [16]:

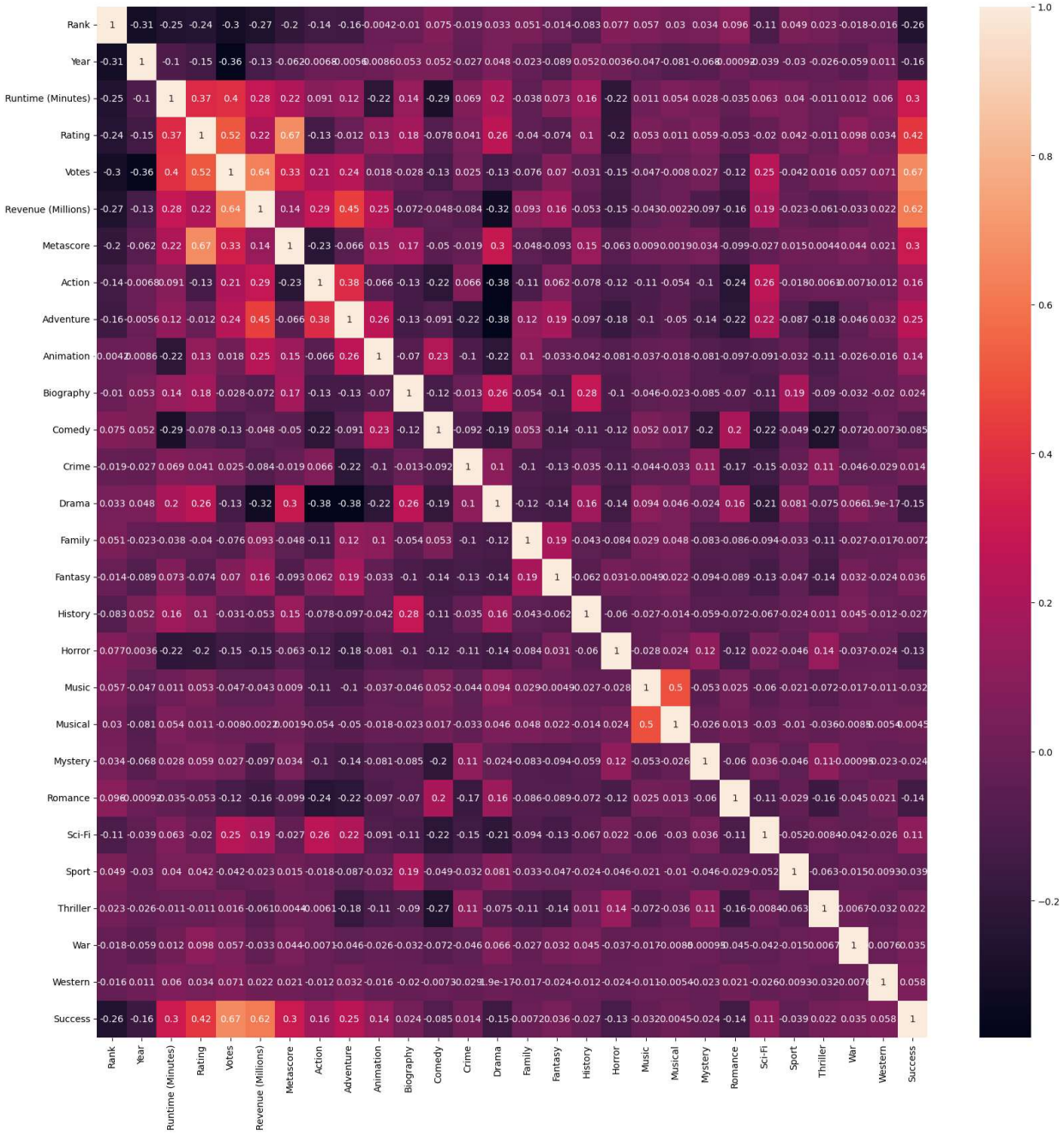
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(20,20))
sns.heatmap(data.corr(),annot = True)
```

```
C:\Users\91811\AppData\Local\Temp\ipykernel_32096\3331294853.py:5: FutureWarning: The default value of nu
meric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only va
lid columns or specify the value of numeric_only to silence this warning.
  sns.heatmap(data.corr(),annot = True)
```

Out[16]:

```
<Axes: >
```



In [17]:

```
director_success_rates = data.groupby("Director")["Success"].mean()
director_success_rates = director_success_rates.sort_values(ascending=False)

# the top 5 most successful directors
print(director_success_rates.head(5))
```

```
Director
James Cameron    1.0
Anthony Russo    1.0
J.J. Abrams      1.0
Tim Miller       1.0
Nathan Greno     1.0
Name: Success, dtype: float64
```

In [19]:

```
actor_success_rates = data.groupby("Actors")["Success"].mean()
actor_success_rates = actor_success_rates.sort_values(ascending=False)

# the top 5 most successful starcast
print(actor_success_rates.head())
```

```
Actors
Robert Downey Jr., Jude Law, Jared Harris, Rachel McAdams            1.0
Chris Pratt, Will Ferrell, Elizabeth Banks, Will Arnett              1.0
Chris Evans, Samuel L. Jackson,Scarlett Johansson, Robert Redford    1.0
Chris Hemsworth, Anthony Hopkins, Natalie Portman, Tom Hiddleston    1.0
Seth Rogen, Katherine Heigl, Paul Rudd, Leslie Mann                  1.0
Name: Success, dtype: float64
```

In [20]:

```
top_movies_by_rating = data.nlargest(5,"Rating")

# top 5 movies by rating
print(top_movies_by_rating[['Title','Rating']])
```

```
                Title  Rating
45    The Dark Knight     9.0
69          Inception     8.8
31       Interstellar     8.6
85      Kimi no na wa     8.6
221  The Intouchables     8.6
```

In [21]:

```
top_movies_by_revenue = data.nlargest(5,"Revenue (Millions)")

# top 5 movies by revenue
print(top_movies_by_revenue[['Title','Revenue (Millions)']])
```

```
                                      Title  Revenue (Millions)
41  Star Wars: Episode VII - The Force Awakens              936.63
76                                   Avatar              760.51
74                            Jurassic World              652.18
65                              The Avengers              623.28
45                           The Dark Knight              533.32
```

In [22]:

```
top_movies_by_metascore = data.nlargest(5,"Metascore")

# top 5 movies by metascore
print(top_movies_by_metascore[['Title','Metascore']])
```

```
                    Title  Metascore
564               Boyhood        100
35              Moonlight         99
202       Pan's Labyrinth         98
20   Manchester by the Sea         96
98       12 Years a Slave         96
```

In [23]:

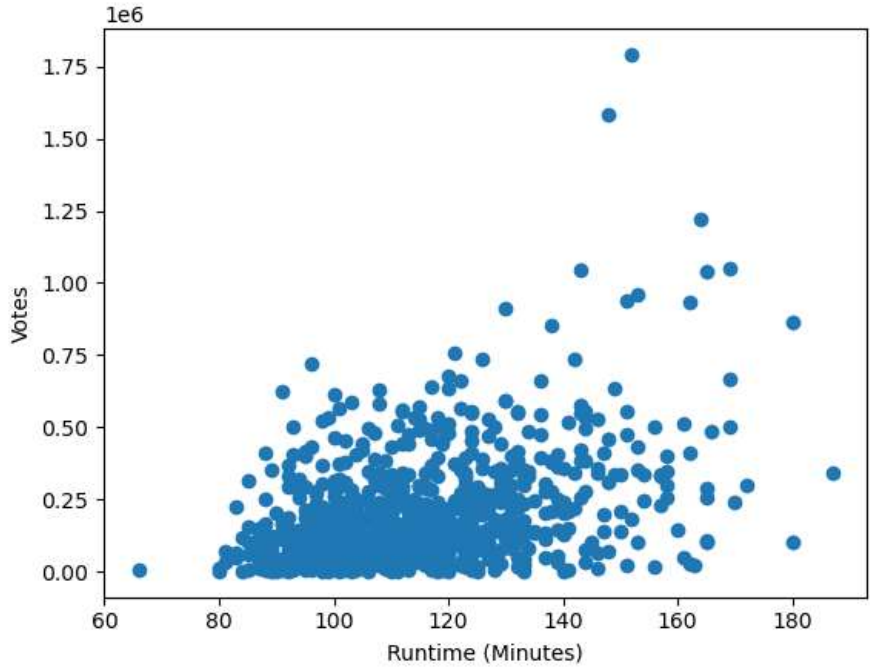
```
top_movies_by_votes = data.nlargest(5,"Votes")

# top 5 movies by votes
print(top_movies_by_votes[['Title','Votes']])
```

```
                    Title    Votes
45          The Dark Knight  1791916
69                Inception  1583625
108  The Dark Knight Rises  1222645
31             Interstellar  1047747
65             The Avengers  1045588
```

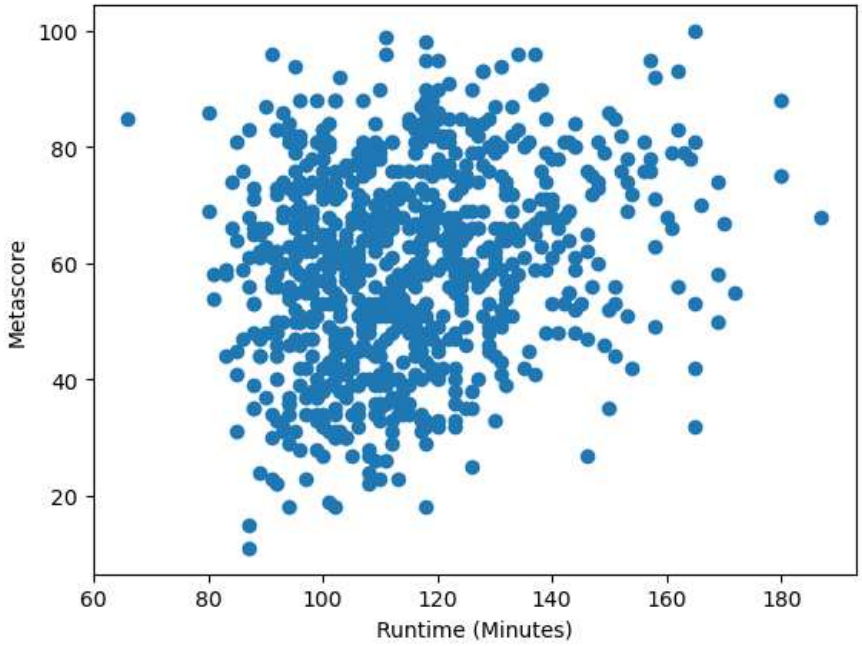
In [24]:

```
plt.scatter(data["Runtime (Minutes)"], data["Votes"])
plt.xlabel("Runtime (Minutes)")
plt.ylabel("Votes")
plt.show()
```



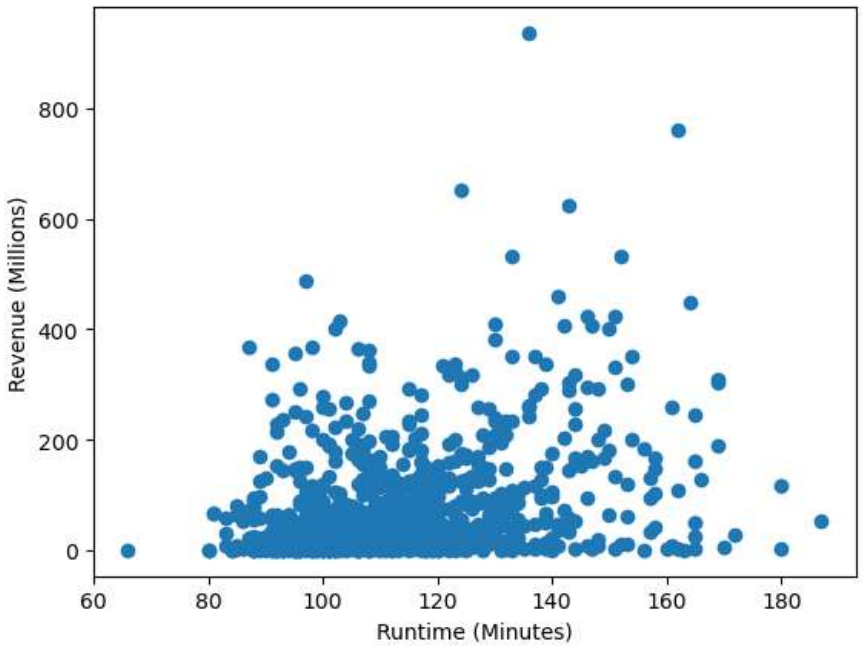
In [25]:

```
plt.scatter(data["Runtime (Minutes)"], data["Metascore"])
plt.xlabel("Runtime (Minutes)")
plt.ylabel("Metascore")
plt.show()
```



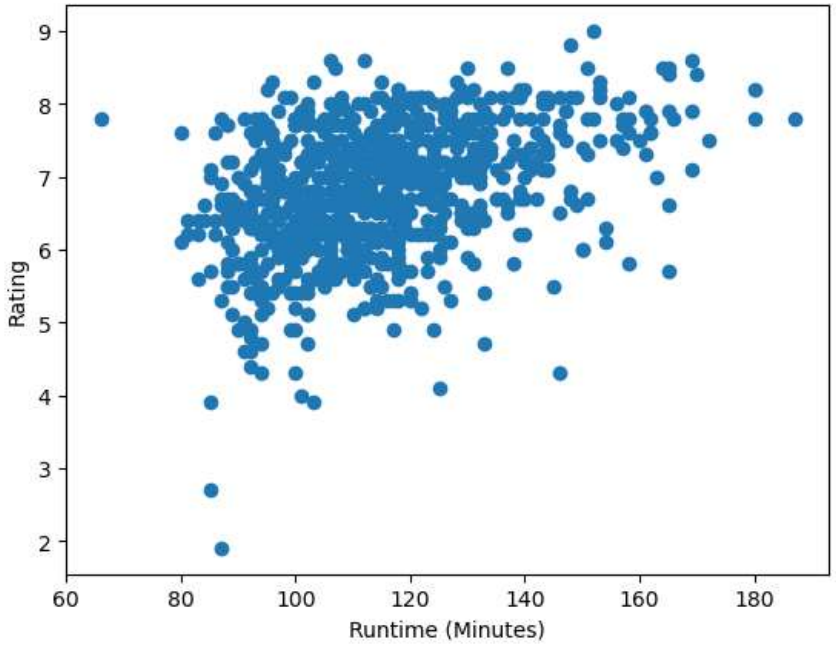
In [26]:

```
plt.scatter(data["Runtime (Minutes)"], data["Revenue (Millions)"])
plt.xlabel("Runtime (Minutes)")
plt.ylabel("Revenue (Millions)")
plt.show()
```



In [27]:

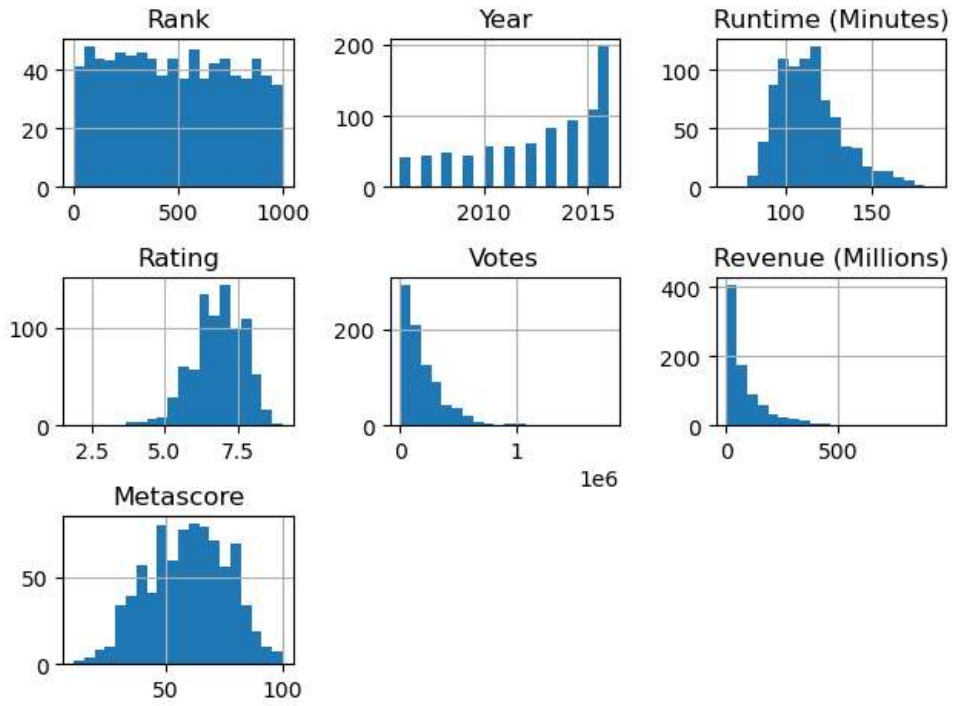
```
plt.scatter(data["Runtime (Minutes)"], data["Rating"])
plt.xlabel("Runtime (Minutes)")
plt.ylabel("Rating")
plt.show()
```



In [28]:

```
plt.figure(figsize=(12, 8))
data[numerical_vars].hist(bins=20)
plt.tight_layout()
plt.show()
```

```
<Figure size 1200x800 with 0 Axes>
```



Defining Success Criteria

In [30]:

```
success_columns = ['Rank', 'Year', 'Runtime (Minutes)', 'Rating', 'Votes', 'Revenue (Millions)', 'Metascore', 'Action',

success_movies = data[data['Success']==1]
success_means = success_movies[success_columns].mean()

unsuccessful_movies = data[data['Success']==0]
unsuccessful_means = unsuccessful_movies[success_columns].mean()

print("Successful Movies Criteria : ")
print(success_means)
print()

print("Unsuccessful Movies Criteria : ")
print(unsuccessful_means)
print()
```

```
Successful Movies Criteria :
Rank                     326.288591
Year                    2011.382550
Runtime (Minutes)        126.671141
Rating                     7.608725
Votes                 469401.315436
Revenue (Millions)       223.159329
Metascore                 70.671141
Action                     0.496644
Adventure                  0.536913
Animation                  0.120805
Biography                  0.093960
Comedy                     0.214765
Crime                      0.161074
Drama                      0.335570
Family                     0.053691
Fantasy                    0.134228
History                    0.020134
Horror                     0.020134
```

Splitting train and test set

In [33]:

```
from sklearn.model_selection import train_test_split

# Drop columns that are not needed
data = data.drop(["Rank", "Title", "Description", "Director", "Actors"], axis=1)

# Convert the categorical features to numerical features
for col in data.columns:
    if data[col].dtype == "object":
        data[col] = data[col].astype("category")
        data[col] = data[col].cat.codes

# Split the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(data, data["Success"], test_size=0.25)
```

...

In [34]:

```
print("Training set shape:", X_train.shape)
print("Testing set shape:", X_test.shape)
```

```
Training set shape: (628, 28)
Testing set shape: (210, 28)
```

Building the Model

In [37]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

model = LogisticRegression()
model.fit(X_train, y_train)

# Evaluate the model on the test set
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("Confusion matrix:")
print(confusion)
```

```
Accuracy: 0.9142857142857143
Precision: 0.7575757575757576
Recall: 0.7142857142857143
F1-score: 0.7352941176470589
Confusion matrix:
[[167   8]
 [ 10  25]]
```

Enhancing Performance

In [38]:

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()
model.fit(X_train, y_train)

# Evaluate the model on the test set
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("Confusion matrix:")
print(confusion)
```

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1-score: 1.0
Confusion matrix:
[[175   0]
 [  0  35]]
```

Taking User input and Predicting

In [39]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)

# Defining the input structure
input_structure = {
    "Title": "str",
    "Genre": "str",
    "Runtime (Minutes)": "int",
    "Lead Actors": "str",
    "Director": "str"
}

# Getting user input for multiple genres
def get_genres():
    genre_list = []
    while True:
        genre = input("Enter a genre (type 'done' to finish): ")
        if genre.lower() == 'done':
            break
        genre_list.append(genre.strip().title())  # Convert to title case (e.g., "action" -> "Action")
    return ', '.join(genre_list)

# Get user input for the movie details
user_input = {}
for field, field_type in input_structure.items():
    if field_type == "str":
        user_input[field] = input(f"Enter the {field}: ").strip()
    elif field_type == "int":
        user_input[field] = int(input(f"Enter the {field}: "))
    elif field == "Genre":
        user_input[field] = get_genres()

# Converting user input to DataFrame format
user_movie = pd.DataFrame(user_input, index=[0])

# Preprocessing the user input to match the encoded dataset columns
user_movie_encoded = pd.get_dummies(user_movie, columns=['Genre'])

# Reindexing the user_movie_encoded DataFrame to match the column order of df_encoded
user_movie_encoded = user_movie_encoded.reindex(columns=data.columns, fill_value=0)

# Scale the user input
user_movie_scaled = scaler.transform(user_movie_encoded)

# Predictions on the user input using the trained model
predicted_success = model.predict(user_movie_scaled)

# Predicting
print("\nPredicted Success for the Inputted Movie:")
if predicted_success[0] == 1:
    print("The movie is predicted to be successful!")
else:
    print("The movie is predicted to be unsuccessful.")
```

...