When a movie is produced then the director would certainly like to maximize his/her movie's revenue. But can we predict what will be the revenue of a movie by using its genre or budget information? This is exactly what we'll learn in this project, we will predict a box office revenue by using the genre of the movie and other related features.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import metrics
from sklearn.metrics import r2_score
from xgboost import XGBRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor

import warnings
warnings.filterwarnings('ignore')
```

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

Out[2]:

	title	domestic_revenue	world_revenue	distributor	opening_revenue	opening_theaters	budget	MPAA	
0	Star Wars: Episode VIII - The Last Jedi	$620,181,382	$1,332,539,889	Walt Disney Studios Motion Pictures	$220,009,584	4,232	$317,000,000	PG-13	Action,Adve
1	The Fate of the Furious	$226,008,385	$1,236,005,118	Universal Pictures	$98,786,705	4,310	$250,000,000	PG-13	Actic
2	Wonder Woman	$412,563,408	$821,847,012	Warner Bros.	$103,251,471	4,165	$149,000,000	PG-13	Action,Adv
3	Guardians of the Galaxy Vol. 2	$389,813,101	$863,756,051	Walt Disney Studios Motion Pictures	$146,510,104	4,347	$200,000,000	PG-13	Action,Adve
4	Beauty and the Beast	$504,014,165	$1,263,521,126	Walt Disney Studios Motion Pictures	$174,750,616	4,210	$160,000,000	PG	Family,Fantas

In [3]:
```
df.shape
```

Out[3]:
```
(2694, 10)
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2694 entries, 0 to 2693
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   title             2694 non-null   object
 1   domestic_revenue  2694 non-null   object
 2   world_revenue     2694 non-null   object
 3   distributor       2694 non-null   object
 4   opening_revenue   2390 non-null   object
 5   opening_theaters  2383 non-null   object
 6   budget            397 non-null    object
 7   MPAA              1225 non-null   object
 8   genres            2655 non-null   object
 9   release_days      2694 non-null   object
dtypes: object(10)
memory usage: 210.6+ KB
```

In [5]:
```
df.describe().T
```

Out[5]:

	count	unique	top	freq
title	2694	2468	A Beautiful Planet	3
domestic_revenue	2694	2495	$11,272,008	3
world_revenue	2694	2501	$25,681,505	3
distributor	2694	248	Fathom Events	292
opening_revenue	2390	2176	$4,696	3
opening_theaters	2383	732	1	503
budget	397	124	$40,000,000	14
MPAA	1225	8	R	568
genres	2655	567	Documentary	351
release_days	2694	457	347	35

EDA

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
title                  0
domestic_revenue       0
world_revenue          0
distributor            0
opening_revenue      304
opening_theaters     311
budget              2297
MPAA                1469
genres                39
release_days           0
dtype: int64
```

Handling the null value columns

In [7]:
```
df.drop('budget', axis=1, inplace=True)

for col in ['MPAA', 'genres']:
    df[col] = df[col].fillna(df[col].mode()[0])
    df.dropna(inplace=True)
    df.isnull().sum().sum()
```

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

Out[8]:
```
title               0
domestic_revenue    0
world_revenue       0
distributor         0
opening_revenue     0
opening_theaters    0
MPAA                0
genres              0
release_days        0
dtype: int64
```

In [9]:
```
df['domestic_revenue'] = df['domestic_revenue'].str[1:]

for col in ['domestic_revenue', 'opening_theaters', 'release_days','opening_revenue','world_revenue']:
    df[col] = df[col].str.replace(',', '')
    temp = (~df[col].isnull())
    df[temp][col] = df[temp][col].convert_dtypes(float)
    df[col] = pd.to_numeric(df[col], errors='coerce')
```

In [10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2356 entries, 0 to 2693
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   title             2356 non-null   object
 1   domestic_revenue  2356 non-null   int64
 2   world_revenue     0 non-null      float64
 3   distributor       2356 non-null   object
 4   opening_revenue   0 non-null      float64
 5   opening_theaters  2356 non-null   int64
 6   MPAA              2356 non-null   object
 7   genres            2356 non-null   object
 8   release_days      2356 non-null   int64
dtypes: float64(2), int64(3), object(4)
memory usage: 184.1+ KB
```

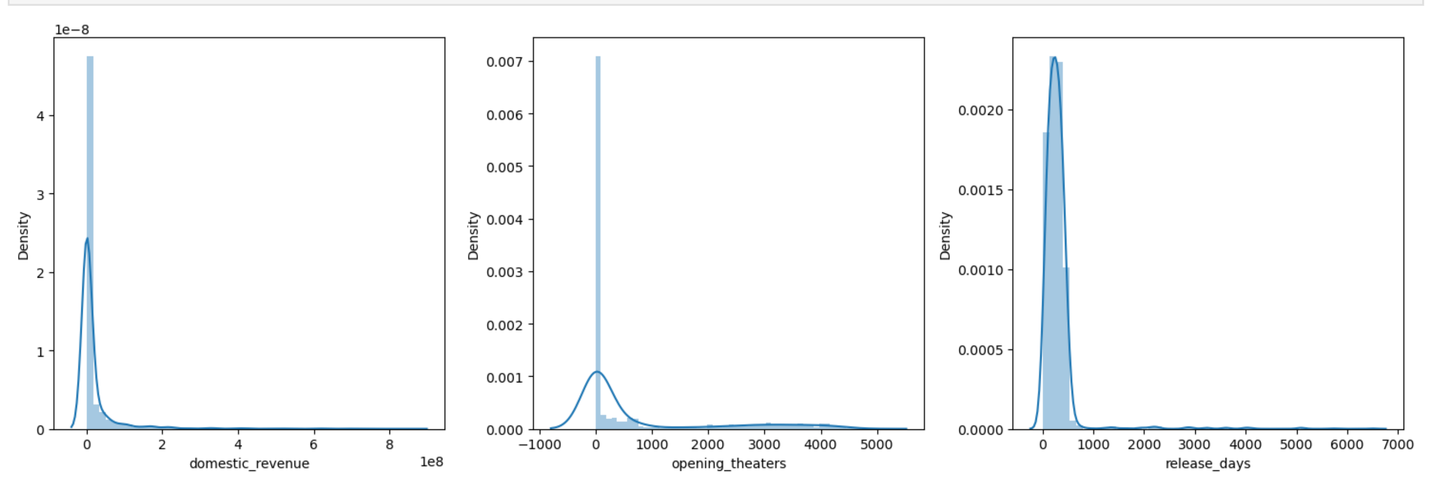
In [11]:
```
df['MPAA'].info()
```

```
<class 'pandas.core.series.Series'>
Int64Index: 2356 entries, 0 to 2693
Series name: MPAA
Non-Null Count  Dtype
--------------  -----
2356 non-null   object
dtypes: object(1)
memory usage: 36.8+ KB
```

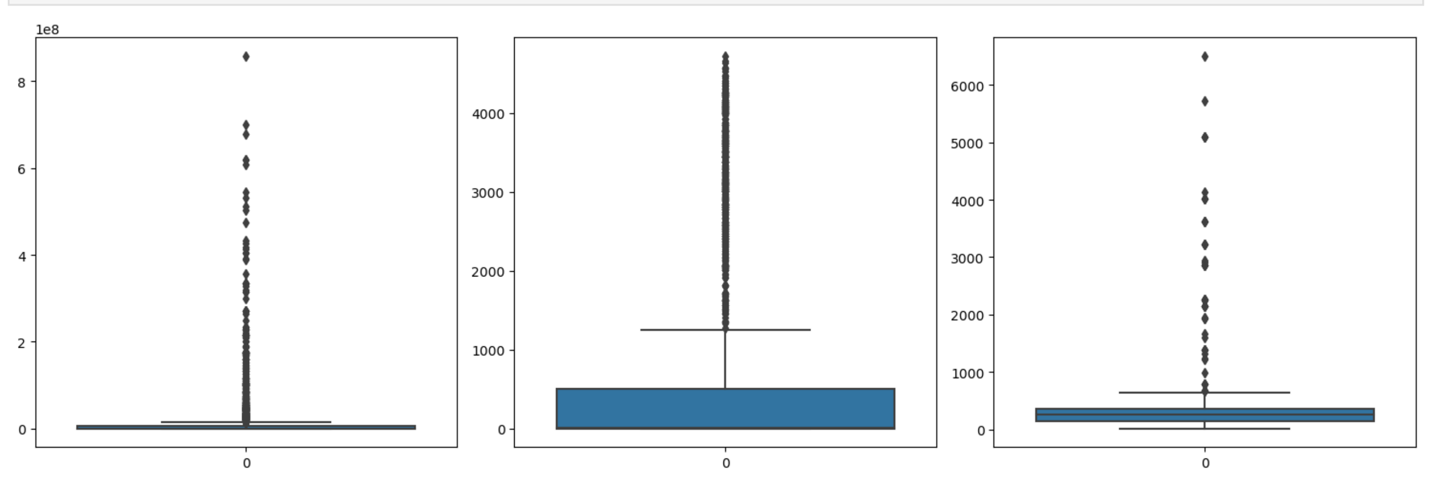
In [12]:
```
df.groupby('MPAA').mean()['domestic_revenue']
```

Out[12]:
```
MPAA
G            3.539276e+07
M/PG         5.113500e+05
NC-17        1.368800e+04
Not Rated    4.897703e+05
PG           5.379622e+07
PG-13        5.891966e+07
R            6.689533e+06
Name: domestic_revenue, dtype: float64
```

In [13]:
```
plt.subplots(figsize=(15, 5))

features = ['domestic_revenue', 'opening_theaters', 'release_days']
for i, col in enumerate(features):
    plt.subplot(1, 3, i+1)
    sb.distplot(df[col])
plt.tight_layout()
plt.show()
```



In [14]:
```
plt.subplots(figsize=(15, 5))
for i, col in enumerate(features):
    plt.subplot(1, 3, i+1)
    sb.boxplot(df[col])
plt.tight_layout()
plt.show()
```



In [15]:
```
for col in features:
    df[col] = df[col].apply(lambda x: np.log10(x))
df[col]
```

Out[15]:
```
0       2.582063
1       2.418301
2       2.336460
3       2.382017
4       2.462398
          ...
2689    2.829947
2690    2.511883
2691    2.607455
2692    2.225309
2693    2.612784
Name: release_days, Length: 2356, dtype: float64
```

In [16]:
```
#pip show scikit-learn
```

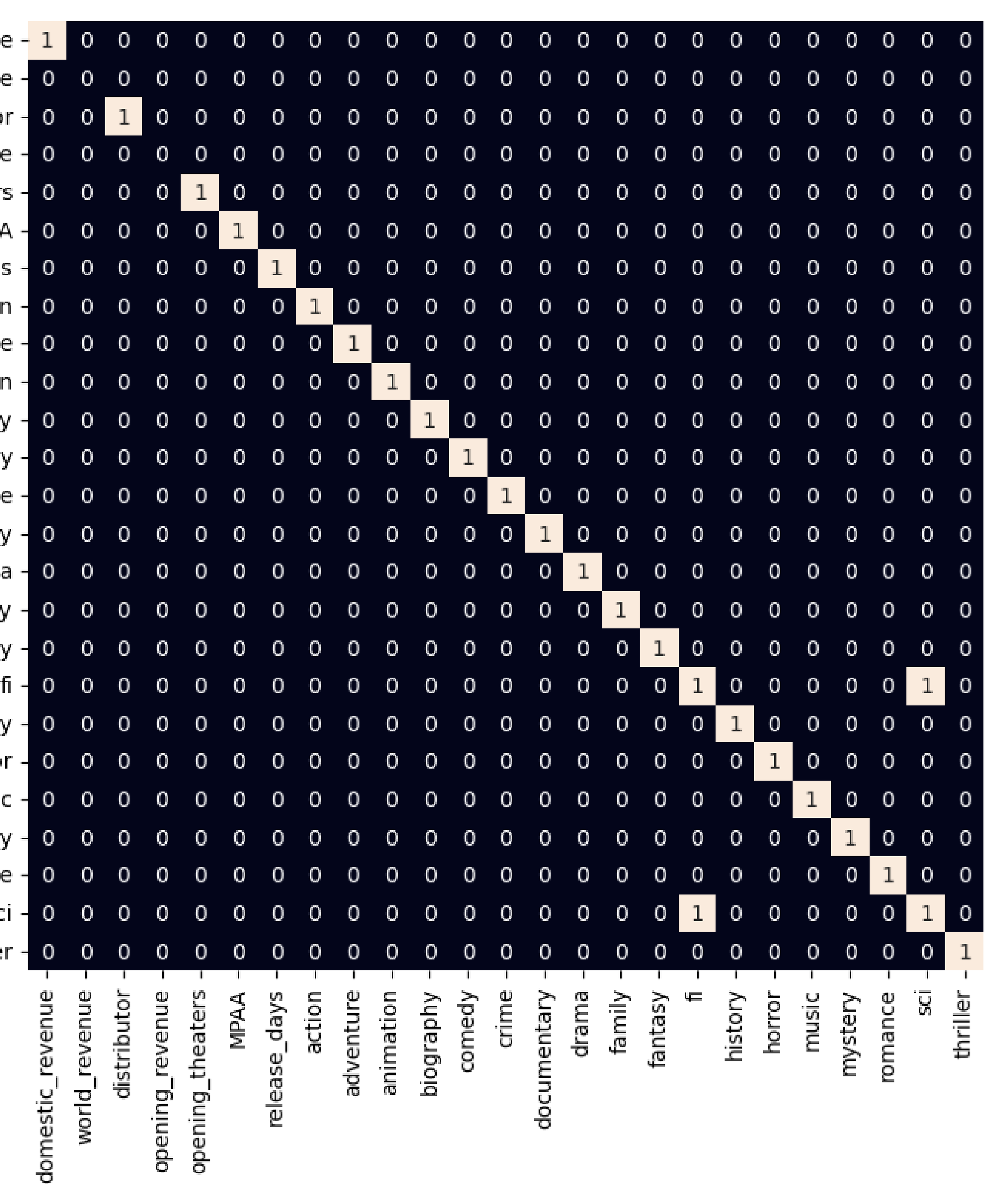
In [17]:
```
vectorizer = CountVectorizer()
vectorizer.fit(df['genres'])
features = vectorizer.transform(df['genres']).toarray()

genres = vectorizer.get_feature_names_out()
for i, name in enumerate(genres):
    df[name] = features[:, i]

df.drop('genres', axis=1, inplace=True)
```

In [18]:
```
removed = 0
for col in df.loc[:, 'action':'western'].columns:

        # Removing columns having more
        # than 95% of the values as zero.
    if (df[col] == 0).mean() > 0.95:
        removed += 1
        df.drop(col, axis=1, inplace=True)

print(removed)
print(df.shape)
```

```
11
(2356, 26)
```

In [19]:
```
for col in ['distributor', 'MPAA']:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
```

In [20]:
```
plt.figure(figsize=(8, 8))
sb.heatmap(df.corr() > 0.8,annot=True,cbar=False)
plt.show()
```



MODEL DEVELOPMENT

In [21]:
```
x = df.drop(['title', 'domestic_revenue', 'fi','world_revenue','opening_revenue'], axis=1)
y = df['domestic_revenue'].values

X_train, X_test,Y_train, Y_test = train_test_split(x, y,test_size=0.1,random_state=22)
X_train.shape, X_test.shape
```

Out[21]:
```
((2120, 21), (236, 21))
```

In [22]:
```
# Normalizing the features for stable and fast training.
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [23]:
```
from sklearn.metrics import mean_absolute_error as mae
xgb = XGBRegressor()
xgb.fit(X_train, Y_train)
```

Out[23]:

▾ XGBRegressor

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

In [24]:
```
train_preds = xgb.predict(X_train)
print('Training Error : ', mae(Y_train, train_preds))

test_preds = xgb.predict(X_test)
print('Testing Error : ', mae(Y_test, test_preds))
print()
```

```
Training Error :  0.13336991677671187
Testing Error :  0.3904341836136789
```

In [25]:
```
ypred=xgb.predict(X_test)
xgb_score=r2_score(Y_test,ypred)
print("Accuracy using XGBooster=",xgb_score*100,"%")
```

```
Accuracy using XGBooster= 82.70998570886019 %
```

In [26]:
```
X_test
```

Out[26]:
```
array([[ 1.12797562, -0.84017049,  0.48116691, ..., -0.38121759,
        -0.28280669, -0.4620702 ],
       [-0.00455565,  0.10659028,  0.48116691, ..., -0.38121759,
        -0.28280669, -0.4620702 ],
       [ 0.31003637, -1.06945019,  0.48116691, ..., -0.38121759,
        -0.28280669, -0.4620702 ],
       ...,
       [-0.28768847, -0.70605046,  0.48116691, ..., -0.38121759,
        -0.28280669, -0.4620702 ],
       [ 0.19992916, -0.53707921,  0.48116691, ..., -0.38121759,
        -0.28280669,  2.16417331],
       [ 1.47402684,  0.14238524,  0.48116691, ..., -0.38121759,
        -0.28280669, -0.4620702 ]])
```

In [27]:
```
lr=LinearRegression()
lr.fit(X_train, Y_train)
ypred=lr.predict(X_test)
lr_score=r2_score(Y_test,ypred)
print("Accuracy using Linear Regression=",lr_score*100,"%")
```

```
Accuracy using Linear Regression= 54.89226215727201 %
```

In [28]:
```
rf=RandomForestRegressor()
rf.fit(X_train, Y_train)
```

Out[28]:

▾ RandomForestRegressor

```
RandomForestRegressor()
```

In [29]:
```
ypred=rf.predict(X_test)
rf_score=r2_score(Y_test,ypred)
rf_score=r2_score(Y_test,ypred)
print("Accuracy using Random Forest=",rf_score*100,"%")
```

```
Accuracy using Random Forest= 78.94030770338487 %
```

XGBOOST and Random Forest is giving best Accuracy than Linear Regression