MOVIE DATA ANALYSIS AND PREDICITNG THE MODEL

Double-click (or enter) to edit

Introduction

Your client is a Media agency. It runs a TV Network in US. The primary source of their revenue/audience comes from the Movies telecasted.

Data Set

The data corresponds to movies shown on a television network in the US. Two datasets have been provided

- Asset revenue data: this has the information on which asset/movie was shown at what time and the revenue generated
- ASSET_METADATA: Characteristics of the asset (Imdb ratings, tomato ratings, release date, genre etc.) Use the two datase

Project Objective

The primary goal is to build a machine-learning model to predict the revenue of a new movie given such features as primetime, release dates, genres. The modeling performance is evaluating based on the Rsquare.

Software Needed

Software: Python and Jupyter Notebook

The following packages (libraries) need to be installed:

1. pandas
2. NumPy
3. scikit Learn
4. Linear Regression
5. GB regressor

Importing Library

```
import numpy as np
import pandas as pd
import sklearn
pd.set_option('max_columns', None)
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
plt.style.use('ggplot')
import datetime
from scipy import stats
from scipy.sparse import hstack, csr_matrix
from sklearn.model_selection import train_test_split, KFold
from sklearn import model_selection
from sklearn.metrics import accuracy_score
from sklearn import model_selection # for splitting into train and test
from sklearn.preprocessing import LabelEncoder
import time
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn import linear_model
import xgboost as xgb
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression
from matplotlib import pyplot
print('Libraries imported..')
```

```
Libraries imported..
```

Gathering Data

```
asset_df  = pd.read_csv('ASSET_METADATA.csv' )
#converting object datatypes to datetime datatypes
schd_df   = pd.read_csv('Schedule DATA.csv' , parse_dates = ['AIRING_START_DATE_TIME
```

```
asset_df.head()
```

	ASSET_ID	RELEASE_YEAR	MPAA_RATING	GENRE	IMDB_RATING	IMDB_RANKING	IMDB
0	204750	2011.0	R	Comedy, Thriller	6.2	46.0	
1	172902	2002.0	R	Action, Sci-Fi, Sport	2.9	14.0	
2	198131	2007.0	PG-13	Adventure, Comedy	2.3	17.0	
3	195393	2006.0	PG-13	Comedy, Drama, Romance	6.8	62.0	
4	185444	2002.0	R	Horror, Mystery	5.4	28.0	

```
schd_df.head()
```

	ASSET_ID	AIRING_START_DATE_TIME	AIRING_END_DATE_TIME	PREMIER_AIRING	EXHIBI1
0	174543	2013-09-13 21:15:00	2013-09-13 23:45:00	0	
1	181758	2014-09-24 00:00:00	2014-09-24 03:00:00	0	
2	191074	2013-09-12 14:15:00	2013-09-12 16:15:00	0	
3	196156	2013-07-14 18:00:00	2013-07-14 21:00:00	1	

```
print('Rows and Columns in asset_df' , asset_df.shape)
print('Rows and Columns in schd_df' , schd_df.shape)
```

```
Rows and Columns in asset_df (5401, 21)
Rows and Columns in schd_df (2517, 10)
```

Merging

Here ASSET_ID is common between both datasets so we can merge our datasets on ASSET_ID

```
df = pd.merge(asset_df , schd_df , on = 'ASSET_ID' , how = 'right')
```

```
# Column names: remove white spaces and convert to lower case
df.columns= df.columns.str.strip().str.lower()
df.columns
```

```
Index(['asset_id', 'release_year', 'mpaa_rating', 'genre', 'imdb_rating',
       'imdb_ranking', 'imdb_votes', 'oscar-nomination', 'oscar-wins',
       'other - nominations', 'other - wins', 'tomato_meter', 'tomato_rating',
       'tomato_reviews', 'tomato_fresh', 'tomato_rotten', 'tomato_user_meter',
       'tomato_user_ratings', 'tomato_user_reviews', 'tomato_image',
       'box_office_earnings', 'airing_start_date_time', 'airing_end_date_time',
       'premier_airing', 'exhibition_airing', 'free_airing',
       'scheduled_runtime', 'day_part', 'airing_revenue', 'c3_rating'],
      dtype='object')
```

```
df.head()
```

	asset_id	release_year	mpaa_rating	genre	imdb_rating	imdb_ranking	imdb_vo
0	185444	2002.0	R	Horror, Mystery	5.4	28.0	70
1	185444	2002.0	R	Horror, Mystery	5.4	28.0	70
2	185444	2002.0	R	Horror, Mystery	5.4	28.0	70
3	185444	2002.0	R	Horror, Mystery	5.4	28.0	70
4	185444	2002.0	R	Horror, Mystery	5.4	28.0	70

```
# get number of rows and columns
df.shape
```

```
(2517, 30)
```

```
#finding null value
df.isnull().sum()*100 / len(df)
```

```
asset_id                0.000000
release_year            0.000000
mpaa_rating             1.191895
genre                   0.000000
imdb_rating             0.000000
imdb_ranking           22.924116
imdb_votes              0.000000
```

```
oscar-nomination           0.000000
oscar-wins                 0.000000
other - nominations        0.000000
other - wins               0.000000
tomato_meter               6.396504
tomato_rating              6.396504
tomato_reviews             6.396504
tomato_fresh               6.396504
tomato_rotten              6.396504
tomato_user_meter          3.138657
tomato_user_ratings        1.986492
tomato_user_reviews        1.986492
tomato_image               6.396504
box_office_earnings       58.760429
airing_start_date_time     0.000000
airing_end_date_time       0.000000
premier_airing             0.000000
exhibition_airing          0.000000
free_airing                0.000000
scheduled_runtime          0.000000
day_part                   0.000000
airing_revenue             0.000000
c3_rating                  0.317839
dtype: float64
```

```
#Let's find duplicate data set
df.duplicated().sum()
```

```
0
```

```
df.isna().sum().sum()
```

```
3239
```

```
 # statistical description, only for numeric values
df.describe()
```

	asset_id	release_year	imdb_rating	imdb_ranking	oscar-nomination	osc w
count	2517.000000	2517.000000	2517.000000	1940.000000	2517.000000	2517.00
mean	180691.980930	1995.406834	6.717163	57.262887	0.800954	0.43
std	7700.604081	11.097619	1.099400	15.964471	1.640999	1.35
min	171646.000000	1931.000000	2.400000	16.000000	0.000000	0.00
25%	173350.000000	1989.000000	6.000000	45.000000	0.000000	0.00
50%	181163.000000	1999.000000	6.600000	58.000000	0.000000	0.00
75%	185529.000000	2003.000000	7.400000	68.000000	1.000000	0.00
max	202224.000000	2011.000000	9.300000	100.000000	11.000000	11.00

```
df.corr()['airing_revenue'].reset_index().sort_values(by = 'airing_revenue' , ascend
```

	index	airing_revenue
20	airing_revenue	1.000000
21	c3_rating	0.565514
19	scheduled_runtime	0.481992
6	other - nominations	0.362417
4	oscar-nomination	0.354495
7	other - wins	0.343653
2	imdb_rating	0.343619
5	oscar-wins	0.341576
11	tomato_fresh	0.337781
14	tomato_user_ratings	0.299939
13	tomato_user_meter	0.290526
9	tomato_rating	0.276545
3	imdb_ranking	0.250159
8	tomato_meter	0.239477
15	tomato_user_reviews	0.231897
10	tomato_reviews	0.216060
17	exhibition_airing	0.059522
16	premier_airing	0.051338
0	asset_id	0.035577
1	release_year	0.026280
12	tomato_rotten	-0.056933
18	free_airing	-0.059522

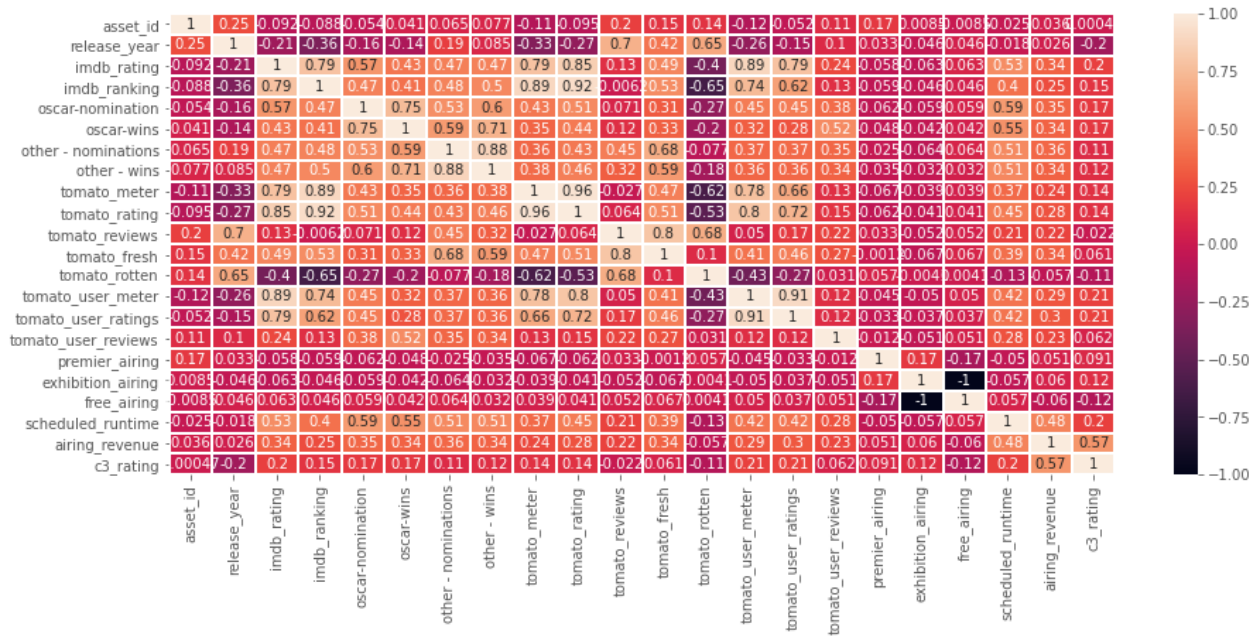
```
# Checking for an zero values in the budget and revenue columns
print("Rows With Zero Values In The  revenue Column:",df[(df['airing_revenue']==0)].
```

```
Rows With Zero Values In The  revenue Column: 0
```

Exploratory Data Analysis

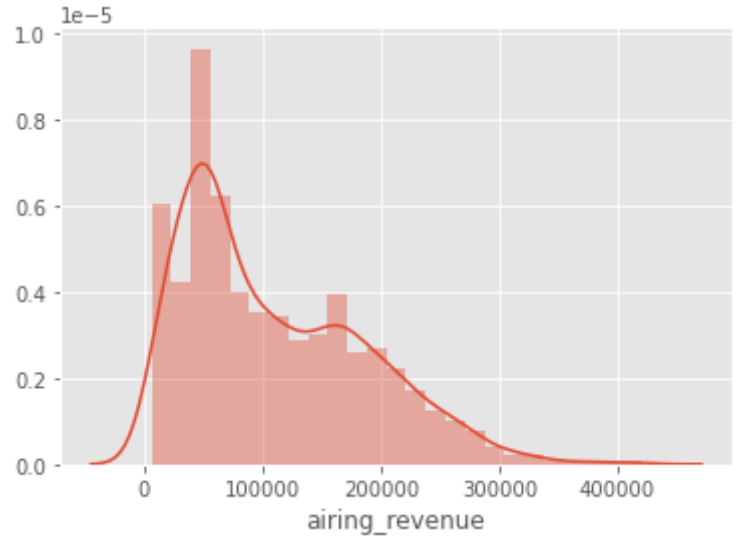
We know that reveune is continous data there we will be using regression method. Regression

```
plt.figure(figsize = (15,6))
sns.heatmap(df.corr(), annot=True, linewidths = 0.2);
```



From this corr chart we can see that revenue is strongly correlated with (c3_rating , scheduled_runtime , imdb_rating) where as least correlated release_year .

```
sns.distplot(df.airing_revenue);
```



We can see that this data is very skewed and therefore it is difficult to draw conclusion from this

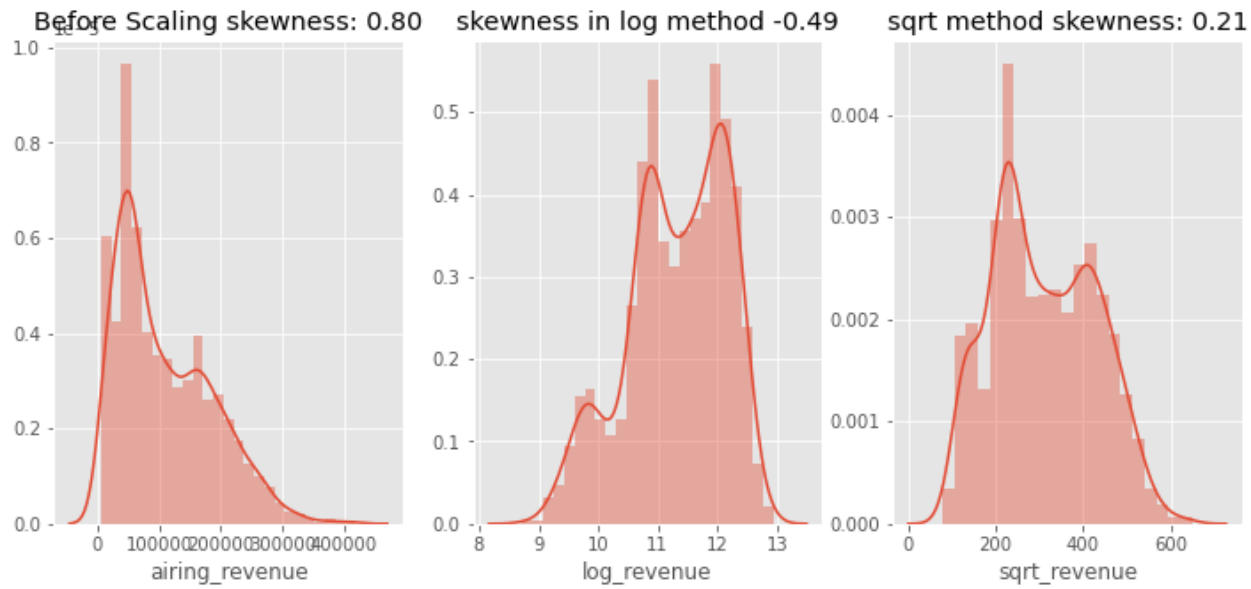
Normalization

```
#creating log transformation and sqrt tranformation for reveune
df['log_revenue'] = np.log1p(df['airing_revenue']) #we are not using log0 to avoid &
df['sqrt_revenue']=np.sqrt(df['airing_revenue'])
```

comapring distribution of reveune and log revune side by side with histogram

```
fig, (ax1, ax2 , ax3) = plt.subplots(ncols=3, figsize=(12, 5))

sns.distplot(df['airing_revenue'] , ax = ax1)
ax1.set_title("Before Scaling skewness: {:0.2f}".format(df['airing_revenue'].skew())
sns.distplot(df['log_revenue'] , ax = ax2)
ax2.set_title('skewness in log method {:0.2f}'.format(df['log_revenue'].skew()))
sns.distplot(df['sqrt_revenue'] , ax = ax3)
ax3.set_title('sqrt method skewness: {:0.2f}'.format(df['sqrt_revenue'].skew()))
plt.show()
```



We can see that orginal distribution i.e (one without log) is extremely skewed. We used log transformation method and made data normally distribution which has less skeweness and kurtosis.after that i used sqrt method that is less skew than both.

Featured Engineering

fixing date column

```
df['release_year']= df['release_year'].astype('int')
```

Extracting decade of released movie

```
def fix_date(x):
    '''
    if the value of date here is less than
    or equal to 19 we can prepend 20 infront of this
    to say that movie is from 2000s
    else we can prepend 19 to say that the movie is
    from 1900s
    '''
    if (x>=1990) & (x<=1999):
        return '90s'
    elif (x>=1980) & (x<=1989):
        return '80s'
    elif (x>=1970) & (x<=1979):
        return '70s'
    elif x<1970:
        return '60s'
    else:
        return '20s'

df['decade'] = df['release_year'].apply(lambda x: fix_date(x)) #applying lambda func
```

Extracting weekday , month , year of airing

```
def process_date(df_date):
    '''this function add column like
     year, weeekday, month and so on column
    and add prefix of airing_date before
    all the above column eg airing_date_year'''
    date_parts = ["year", "weekday", "month", 'weekofyear', 'day', 'quarter']
    for part in date_parts:
        part_col = 'airing_date' + "_" + part #add prefix as  "release_date" before
        df[part_col] = getattr(df['airing_start_date_time'].dt, part).astype(int)

    return df_date


df = process_date(df)
```

Prime Time

```
df['prime_time'] = np.where(df['day_part'].isin(['Early Fringe' , 'Late Fringe']) ,
```

Seasons

```
#adding Month column and season column
def getseason(x):
    if (x >11  or x <3):
       return "WINTER"
    elif (x>=3 and x<= 5):
       return "SPRING"
    elif (x>=6  and x<9):
       return "SUMMER"
    else:
       return "FALL"


df['season'] = df['airing_date_month'].apply(getseason)
```

Weekend

```
df['weekend']= np.where(df['airing_date_weekday'].isin([6,7]) , 1 , 0 )
```

Holidays

```
from pandas.tseries.holiday import USFederalHolidayCalendar as calendar
cal = calendar()
holidays = cal.holidays(start=df['airing_start_date_time'].min(), end=df['airing_sta
# df['airing_date_daystart_date_time'] = pd.to_datetime(df['airing_start_date_time']
df['holidays'] = np.where(df['airing_start_date_time'].isin(holidays) , 1 , 0 )


df.head(5)
```

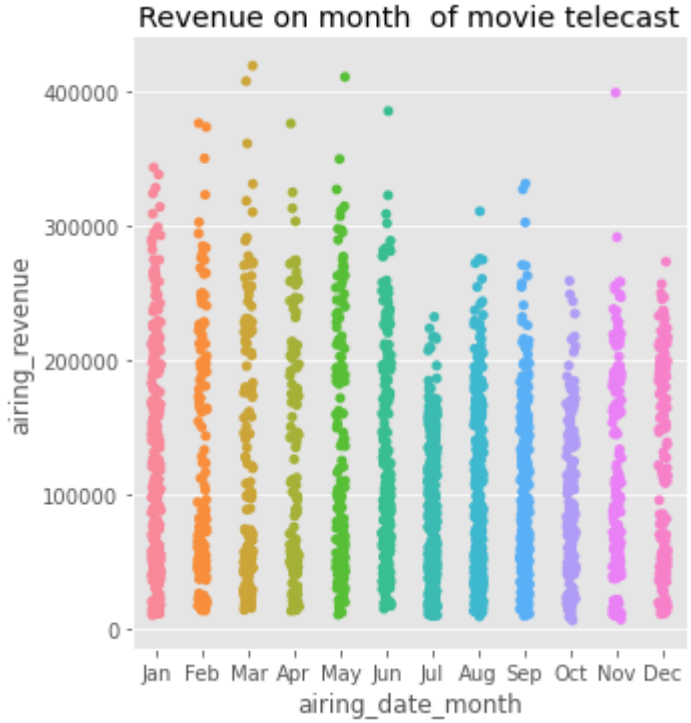
	asset_id	release_year	mpaa_rating	genre	imdb_rating	imdb_ranking	imdb_v
0	185444	2002	R	Horror, Mystery	5.4	28.0	70
1	185444	2002	R	Horror, Mystery	5.4	28.0	70
2	185444	2002	R	Horror, Mystery	5.4	28.0	70
3	185444	2002	R	Horror, Mystery	5.4	28.0	70
4	185444	2002	R	Horror, Mystery	5.4	28.0	70

Do airing date affects revenue ?

we will be creating categorical plot as day of the week, month are not continuous data.

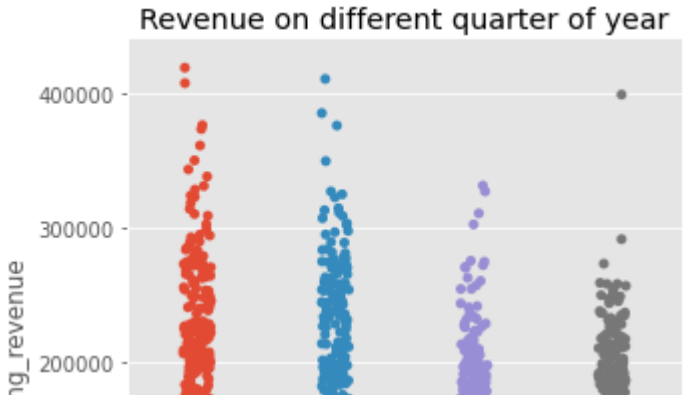
```
#since day, month are categorical variable
plt.figure(figsize=(15,6));
sns.catplot(x='airing_date_month', y='airing_revenue', data=df);
plt.title('Revenue on month  of movie telecast');
#lets replace number by actual month name
loc, labels = plt.xticks()
loc, labels = loc, ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "
plt.xticks(loc, labels,fontsize=10)
plt.show()
```

```
<Figure size 1080x432 with 0 Axes>
```

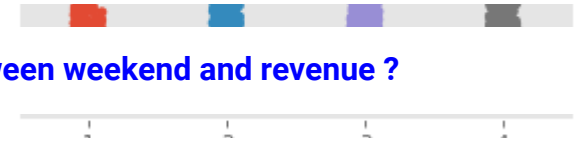


From the above chart we can see that movie telecasted in `March and May` has maximum revenue where as movie released in `Jul` has less revenue compared to other months.

```
sns.catplot(x='airing_date_quarter', y='airing_revenue', data=df);
plt.title('Revenue on different quarter of year');
```

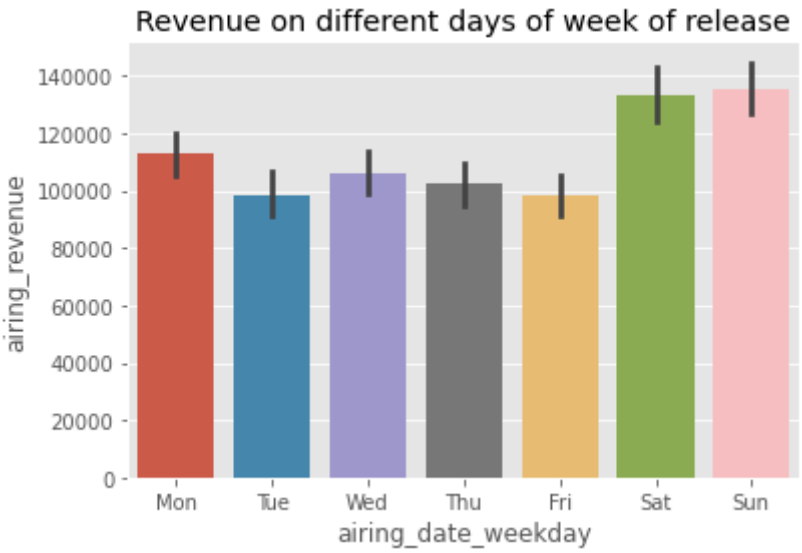


From the above chart we can see that movie released in second quarter (January-March) has more revenue compared to movie released in last quarter



Relation between weekend and revenue ?

```
sns.barplot(x='airing_date_weekday', y='airing_revenue', data=df);
plt.title('Revenue on different days of week of release');
loc, labels = plt.xticks()
#putting label for days
loc, labels = loc, ["Mon","Tue","Wed","Thu","Fri","Sat","Sun"]
plt.xticks(loc, labels)
plt.show()
```

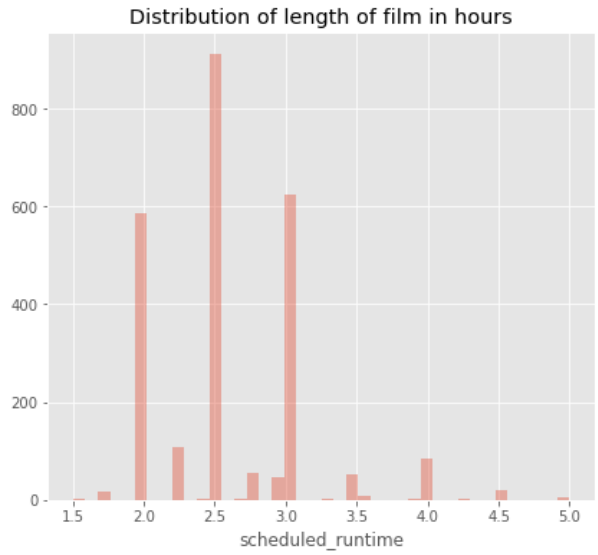


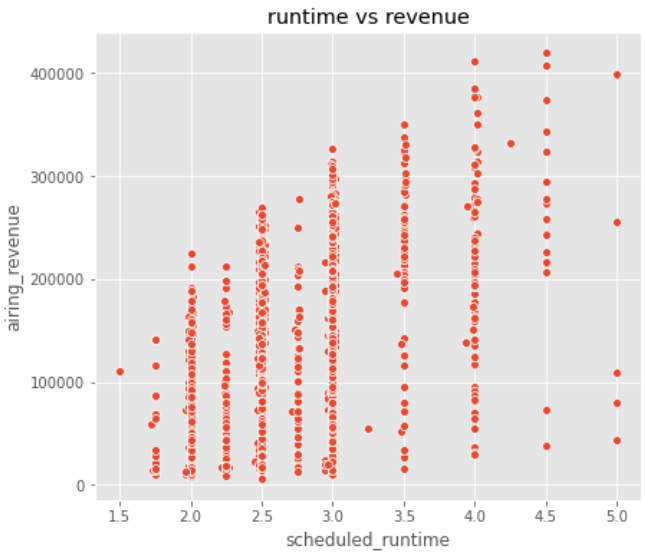
movie telecasted on weekend has more revenue.

Relation between runtime and revenue ?

```
plt.figure(figsize=(15, 6))
plt.subplot(1, 2, 1)
sns.distplot(df['scheduled_runtime'].fillna(0) / 60, bins=40, kde=False); #filling r
plt.title('Distribution of length of film in hours');
plt.subplot(1, 2, 2)
```

```
sns.scatterplot(df['scheduled_runtime'].fillna(0)/60, df['airing_revenue'])
plt.title('runtime vs revenue');
```



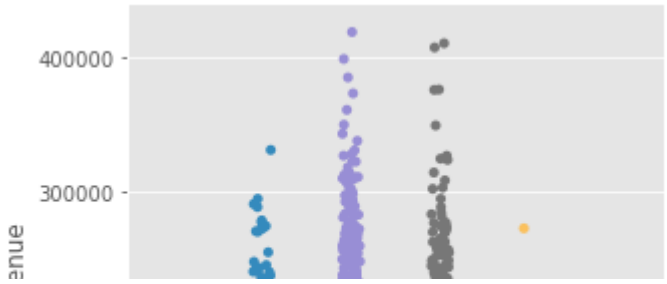


Here we have run time in hour on x-axis and freq of movie in on y axis and then we cas see that most of the movie are between 2-3 hr.And the movie that fall on 4 ot 4.5 hours duration has highest revenue.

Relation between day_part and revenue?

```
plt.figure(figsize = (15,6))
sns.catplot(x = 'day_part' ,  y = 'airing_revenue' , data = df)
plt.xticks(rotation = 60)
plt.show()
```

```
<Figure size 1080x432 with 0 Axes>
```

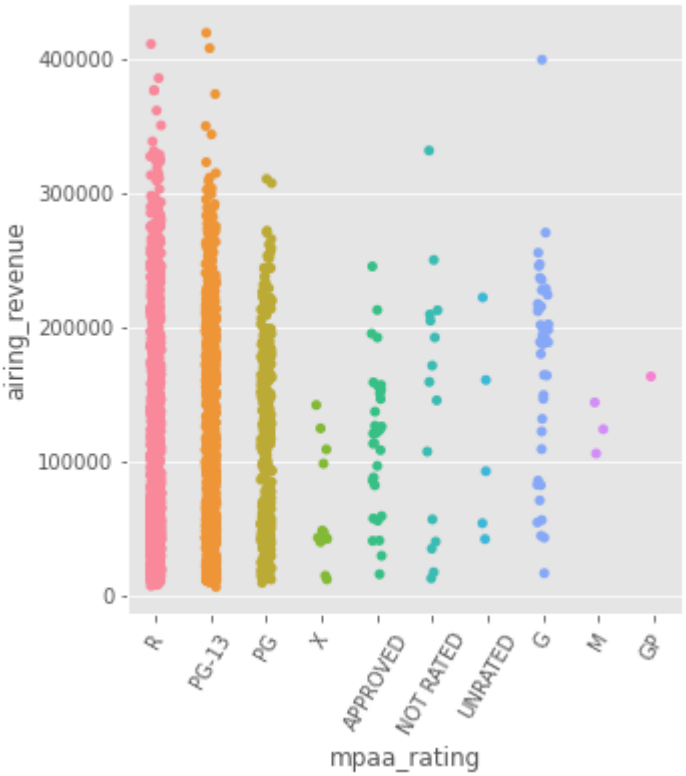


From the above fig, we can see that movie that has telecasted at prime time has more revenue compared to the movie that has telecasted at normal daytime.

Relation between mpaa_rating and revenue?

```
plt.figure(figsize=(15,6))
sns.catplot(x = 'mpaa_rating' ,  y = 'airing_revenue' , data = df)
plt.xticks(rotation = 60)
plt.show()
```

```
<Figure size 1080x432 with 0 Axes>
```

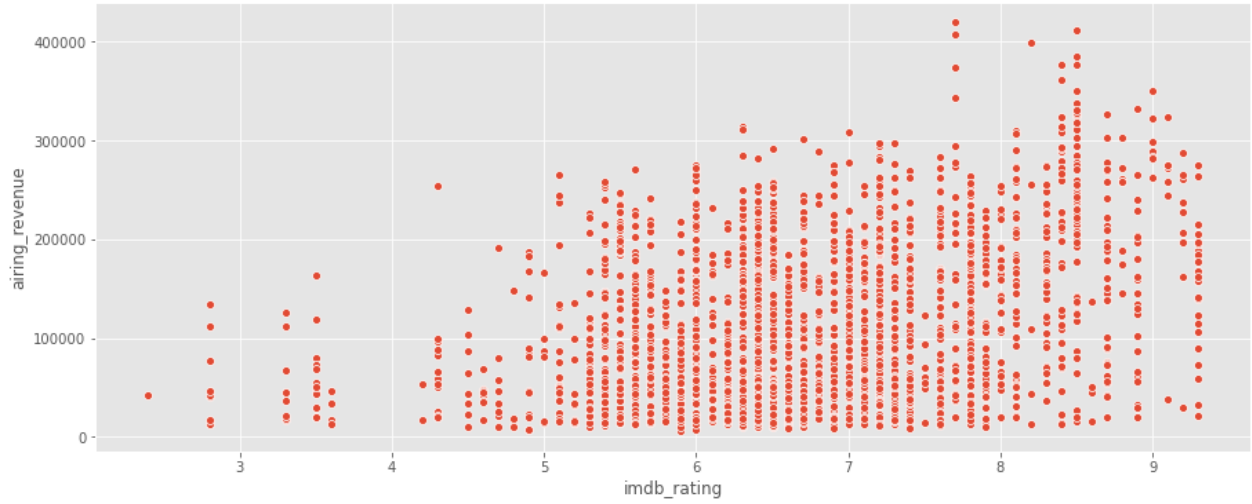


from this above bargraph R , PG-13 rated movie has high revenue compared to all others rated movies

Relation between imdb_rating and revenue?

```
plt.figure(figsize=(15,6))
sns.scatterplot(x = 'imdb_rating' ,  y = 'airing_revenue' , data =df  )
```

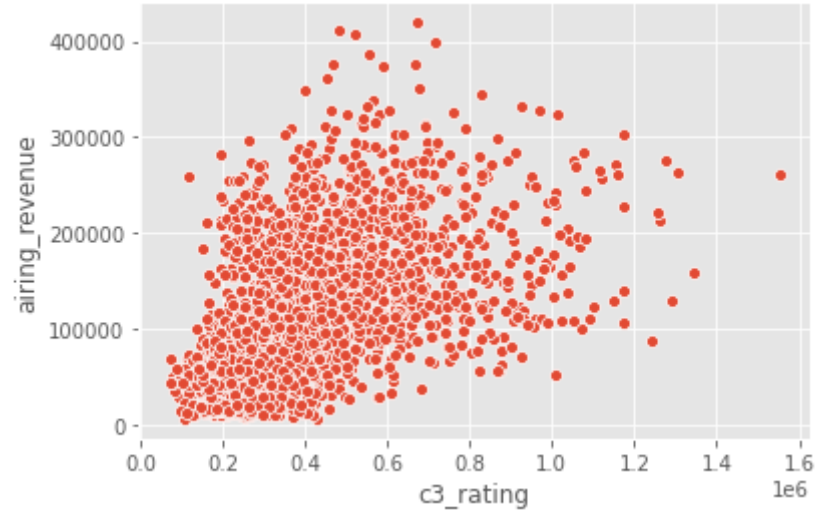
```
<AxesSubplot:xlabel='imdb_rating', ylabel='airing_revenue'>
```



Relation between c3_rating and revenue?

```
sns.scatterplot(x = 'c3_rating' ,  y = 'airing_revenue' , data = df)
```

```
<AxesSubplot:xlabel='c3_rating', ylabel='airing_revenue'>
```



From above graph we can clearly see that C3_RATING is strongly correlated with Revenue

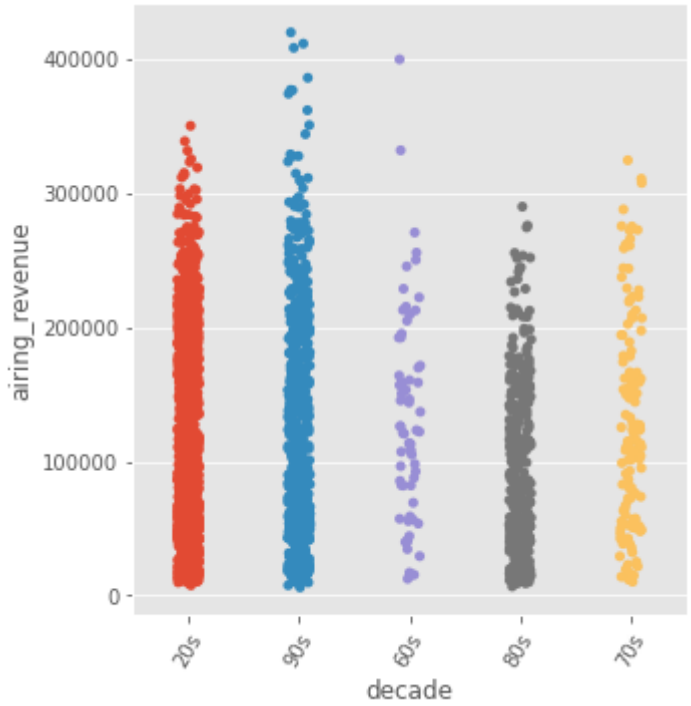
Relation between decade and revenue?

```
df['decade'].unique()
```

```
array(['20s', '90s', '60s', '80s', '70s'], dtype=object)
```

```
plt.figure(figsize=(15,6))
sns.catplot(x = 'decade' ,  y = 'airing_revenue' , data = df)
plt.xticks(rotation = 60)
plt.show()
```

```
<Figure size 1080x432 with 0 Axes>
```



Movies released in `90s and 60s` has more revenue generateed

```
df['genre'] =df['genre'].str.split(',')
```

```
genres_df =df['genre'].apply(pd.Series)
genres_df.head(10)
```

	0	1	2
0	Horror	Mystery	NaN
1	Horror	Mystery	NaN
2	Horror	Mystery	NaN
3	Horror	Mystery	NaN
4	Horror	Mystery	NaN
5	Horror	Mystery	NaN
6	Drama	Sci-Fi	Thriller
7	Drama	Sci-Fi	Thriller
8	Drama	Sci-Fi	Thriller
9	Action	Comedy	Crime

So now, we have N number of columns (number of distinct genres) for each movie, and non null columns represent each of the movies' genres.

```
genres_df[1].unique()
```

```
array([' Mystery', ' Sci-Fi', ' Comedy', ' Drama', ' Crime', ' Adventure',
       ' Family', ' Music', nan, ' Thriller', ' History', ' Romance',
       ' War', ' Horror', ' Western', ' Musical', ' Fantasy', ' Sport',
       ' Biography', ' Documentary', ' Short'], dtype=object)
```

there are some leading white spaces are in genre

Removing whitespaces frpm genres_df[1] columns

```
genres_df[1]= genres_df[1].str.strip()
genres_df[2]= genres_df[2].str.strip()
```

In order to create dummies for each of these movies, we need to stack to reshape our DataFrame.

So, for each of the genres for each movie we now have a row. Here are the rows corresponding our first movie:

```
genres_df.head()
```

	0	1	2
0	Horror	Mystery	NaN
1	Horror	Mystery	NaN
2	Horror	Mystery	NaN
3	Horror	Mystery	NaN
4	Horror	Mystery	NaN

```
stacked_genres = genres_df.stack()
stacked_genres.head()
```

```
0  0     Horror
   1    Mystery
1  0     Horror
   1    Mystery
2  0     Horror
dtype: object
```

link textSo we converted our cleaned list `[Horror , Mystery]` to 2 rows, each represents only 1 particular genre. We are now ready to convert this to dummies!

```
raw_dummies = pd.get_dummies(stacked_genres)
raw_dummies.head()[['Horror' , 'Mystery']]
```

		Horror	Mystery
0	**0**	1	0
	1	0	1
1	**0**	1	0
	1	0	1
2	**0**	1	0

```
raw_dummies.head()[['Horror' ,'Mystery','Comedy','Family','Crime','Documentary']]
```

		Horror	Mystery	Comedy	Family	Crime	Documentary
0	**0**	1	0	0	0	0	0
	1	0	1	0	0	0	0
1	**0**	1	0	0	0	0	0
	1	0	1	0	0	0	0
2	**0**	1	0	0	0	0	0

Here's how the new DataFrame looks like, for simplicity I only selected a few columns. But in actual DataFrame, we have all the genres.

And if you focus on the first 3 rows, you see that Animation,Comedy and Family columns are 1, and the rest are 0.

we need these dummy columns but we only need one row per movie. So how do we do that?

We just aggregate the newly created DataFrame by summing on index level.

```
genre_dummies = raw_dummies.sum(level=0)
genre_dummies.head(10)
```

	Action	Adventure	Biography	Comedy	Crime	Documentary	Drama	Family	Fantas
0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	
5	0	0	0	0	0	0	0	0	
6	0	0	0	0	0	0	1	0	

```
unique_genres = raw_dummies.columns
print("Number of genres: {}".format(len(unique_genres)))
print("Genres: {}".format(unique_genres))
```

```
Number of genres: 21
Genres: Index(['Action', 'Adventure', 'Biography', 'Comedy', 'Crime', 'Documentary',
       'Drama', 'Family', 'Fantasy', 'History', 'Horror', 'Music', 'Musical',
       'Mystery', 'Romance', 'Sci-Fi', 'Short', 'Sport', 'Thriller', 'War',
       'Western'],
      dtype='object')
```

One Hot Encding

[ ] ↳ *16 cells hidden*

Linear Regression

```
#linear regression
lm = LinearRegression() #our 6th model
lm.fit(X_train, y_train)
lm_preds = lm.predict(X_test)
print("R Square: ", r2_score(y_test, lm_preds))
```

```
R Square:  0.7700134734566438
```

Our R square value is 77%

Random Forrest

```
#random forrest
```

```
import sklearn.metrics as metrics
from sklearn.ensemble import RandomForestRegressor

RF_model = RandomForestRegressor(random_state =0, n_estimators=500, max_depth=10)
RF_model.fit(X_train, y_train)

y_hat = RF_model.predict(X_test)
print ("R-Squared is:", metrics.r2_score(y_hat, y_test))
```

```
R-Squared is: 0.813252235836355
```

Our predicted from Random forest is 81% accurate.

Importance features

```
importances = pd.DataFrame({'feature':X_train.columns,'importance':np.round(RF_model
importances = importances.sort_values('importance',ascending=False).set_index('featu
print(importances)
plt.figure(figsize = (15,6))
sns.barplot(x='feature' , y ='importance' , data = importances)
plt.xticks(rotation = 90)
plt.show()
```

	feature	importance
0	prime_time	0.505
1	scheduled_runtime	0.158
2	c3_rating	0.139
3	airing_date_weekofyear	0.034
4	day_part_Daytime	0.032
..	...	...
61	mpaa_rating_M	0.000
62	mpaa_rating_GP	0.000
63	holidays	0.000
64	premier_airing	0.000

GB regressor

[ ] ↳ *6 cells hidden*

Conclusion

↳ *1 cell hidden*