Amazon Prime Video Data Analysis and Prediction

Part 0 Load packages, load data

```
#import neccessary libraries
import numpy as np
import pandas as pd
import sklearn as sl
import sklearn.preprocessing as preprocessing
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
```

```
pd.set_option('display.float_format', lambda x: '%.3f' % x)
pd.set_option('display.max_columns',None)
pd.set_option('display.max_rows',None)
pd.set_option('max_colwidth',100)
```

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

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving TVdata.txt to TVdata.txt

```
TV=pd.read_table('TVdata.txt',header=0,sep=',',lineterminator='\n')
print(TV.head())
```

```
   video_id  cvt_per_day  weighted_categorical_position  \
0    385504   307127.606                              1
1    300175   270338.426                              1
2    361899   256165.867                              1
3    308314   196622.721                              3
4    307201   159841.652                              1

   weighted_horizontal_poition  import_id  release_year  \
0                            3  lionsgate          2013
1                            3  lionsgate          2013
2                            3      other          2012
3                            4  lionsgate          2008
4                            3  lionsgate          2013

                                        genres  imdb_votes    budget  \
0                       Action,Thriller,Drama       69614  15000000
1                       Comedy,Crime,Thriller       46705  15000000
2                                 Crime,Drama      197596  26000000
3  Thriller,Drama,War,Documentary,Mystery,Action      356339  15000000
4            Crime,Thriller,Mystery,Documentary       46720  27220000
```

```
   boxoffice  imdb_rating  duration_in_mins  metacritic_score       awards  \
0   42930462        6.500           112.301                51  other award
1    3301046        6.500            94.983                41     no award
2   37397291        7.300           115.764                58  other award
3   15700000        7.600           130.704                94        Oscar
4    8551228        6.400           105.546                37  other award

    mpaa  star_category
0  PG-13          1.710
1      R          3.250
2      R          2.647
3      R          1.667
4      R          3.067
```

Part 1: Data Exploration

1.1 Exclude erroneous data

Each video should only appear once in the list, duplicated video will be removed.

```
if TV['video_id'].duplicated().sum()==0:
  print('no duplicated index')
```

```
no duplicated index
```

```
type(TV)
```

```
pandas.core.frame.DataFrame
```

1.2 Understand numerical features

1.2.1 Overview

```
TV.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4226 entries, 0 to 4225
Data columns (total 16 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   video_id                        4226 non-null   int64
 1   cvt_per_day                     4226 non-null   float64
 2   weighted_categorical_position   4226 non-null   int64
 3   weighted_horizontal_poition     4226 non-null   int64
 4   import_id                       4226 non-null   object
 5   release_year                    4226 non-null   int64
```

```
 6   genres                             4226 non-null   object
 7   imdb_votes                         4226 non-null   int64
 8   budget                             4226 non-null   int64
 9   boxoffice                          4226 non-null   int64
 10  imdb_rating                        4226 non-null   float64
 11  duration_in_mins                   4226 non-null   float64
 12  metacritic_score                   4226 non-null   int64
 13  awards                             4226 non-null   object
 14  mpaa                               4226 non-null   object
 15  star_category                      4226 non-null   float64
dtypes: float64(4), int64(8), object(4)
memory usage: 528.4+ KB
```

```
print(TV.drop(columns=['video_id','release_year'],axis=1).describe(percentiles=[0.1
```

```
       cvt_per_day  weighted_categorical_position  \
count     4226.000                       4226.000
mean      4218.630                          7.783
std      13036.080                          6.134
min          2.188                          1.000
10%        141.985                          3.000
25%        351.169                          4.000
50%       1193.500                          6.000
75%       3356.789                          9.000
95%      14692.834                         22.000
max     307127.606                         41.000

       weighted_horizontal_poition  imdb_votes        budget     boxoffice  \
count                     4226.000    4226.000      4226.000      4226.000
mean                        28.104    6462.924   2150743.439   2536338.472
std                         11.864   31596.007   7176604.483   8243516.266
min                          1.000       0.000         0.000         0.000
10%                         13.000       8.000         0.000         0.000
25%                         20.000      81.000         0.000         0.000
50%                         28.000     535.000         0.000         0.000
75%                         36.000    3053.000   1500000.000         0.000
95%                         48.000   26199.500  12000000.000   8551228.000
max                         70.000  948630.000 107000000.000 184208848.000

       imdb_rating  duration_in_mins  metacritic_score  star_category
count     4226.000          4226.000          4226.000       4226.000
mean         5.257            89.556            15.974          0.955
std          2.123            21.086            26.205          0.955
min          0.000             4.037             0.000          0.000
10%          2.300            62.391             0.000          0.000
25%          4.300            82.602             0.000          0.000
50%          5.800            90.730             0.000          1.000
75%          6.800            99.500            41.000          1.667
95%          7.800           119.131            65.000          2.597
max         10.000           246.017           100.000          4.000
```

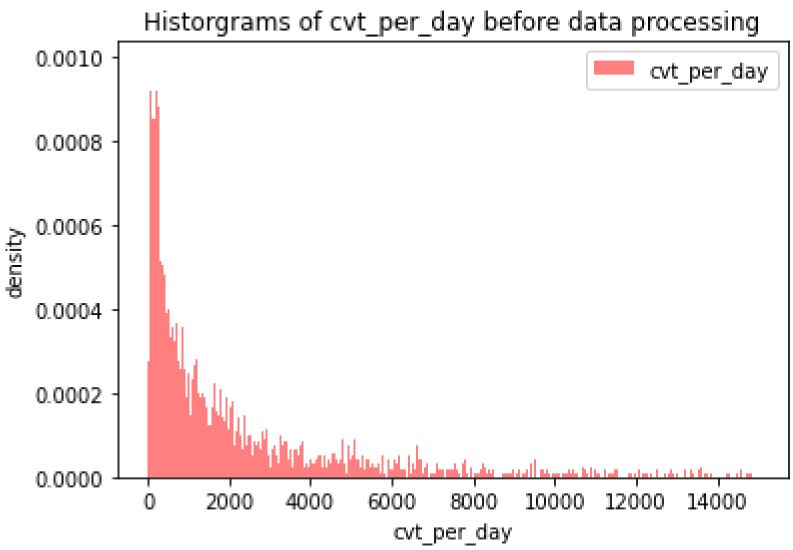
```
(TV==0).sum(axis=0)/TV.shape[0]
```

```
video_id                        0.000
cvt_per_day                     0.000
weighted_categorical_position   0.000
weighted_horizontal_poition     0.000
import_id                       0.000
```

```
release_year              0.000
genres                    0.000
imdb_votes                0.081
budget                    0.581
boxoffice                 0.756
imdb_rating               0.081
duration_in_mins          0.000
metacritic_score          0.713
awards                    0.000
mpaa                      0.000
star_category             0.437
dtype: float64
```

1.2.2 cvt_per_day feature

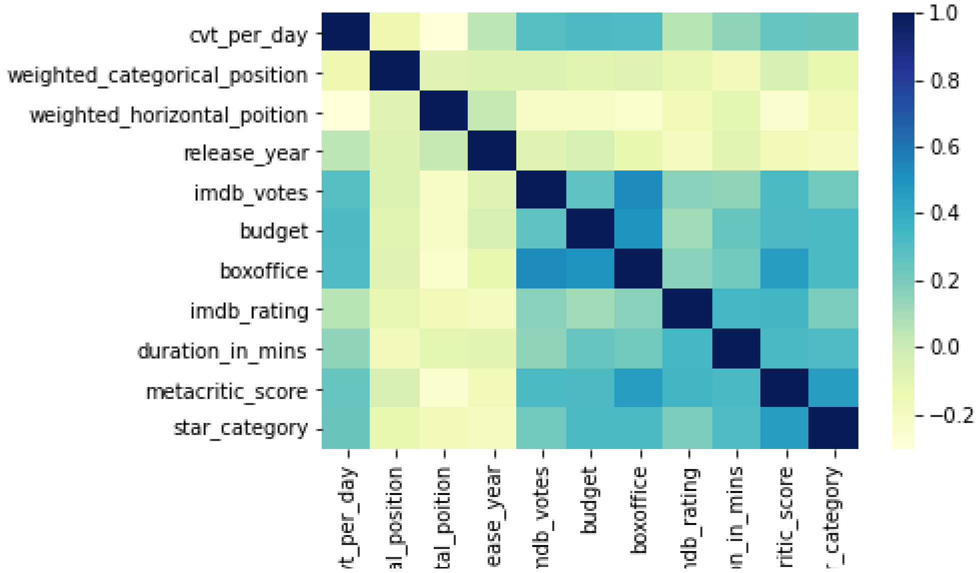
```
plt.hist(TV['cvt_per_day'],bins=range(0,15000,30),color='r',label='cvt_per_day',den
plt.title('Historgrams of cvt_per_day before data processing')
plt.legend(loc='upper right')
plt.xlabel('cvt_per_day')
plt.ylabel('density')
plt.show()
```



1.2.3 Correlation among numerical features

```
corr = TV[['cvt_per_day','weighted_categorical_position','weighted_horizontal_poiti
               ,'release_year', 'imdb_votes', 'budget', 'boxoffice' ,'imdb_rating',
               'duration_in_mins', 'metacritic_score', 'star_category']].corr()

sns.heatmap(corr, cmap="YlGnBu")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9b33371978>
```



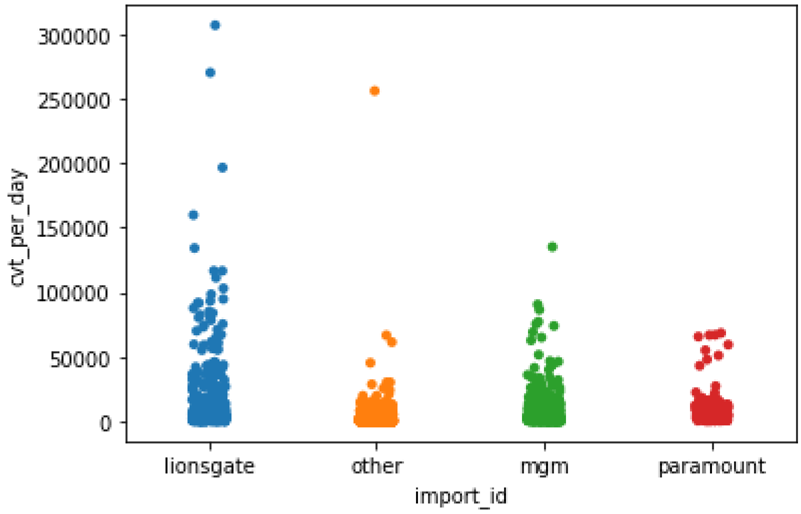

```
corr
```

	cvt_per_day	weighted_categorical_position	weighte
cvt_per_day	1.000	-0.148	
weighted_categorical_position	-0.148	1.000	
weighted_horizontal_poition	-0.302	-0.084	
release_year	0.046	-0.069	
imdb_votes	0.298	-0.064	
budget	0.316	-0.090	
boxoffice	0.312	-0.074	
imdb_rating	0.059	-0.116	
duration_in_mins	0.152	-0.174	
metacritic_score	0.249	-0.044	
star_category	0.247	-0.123	

1.3 Understand categorical features

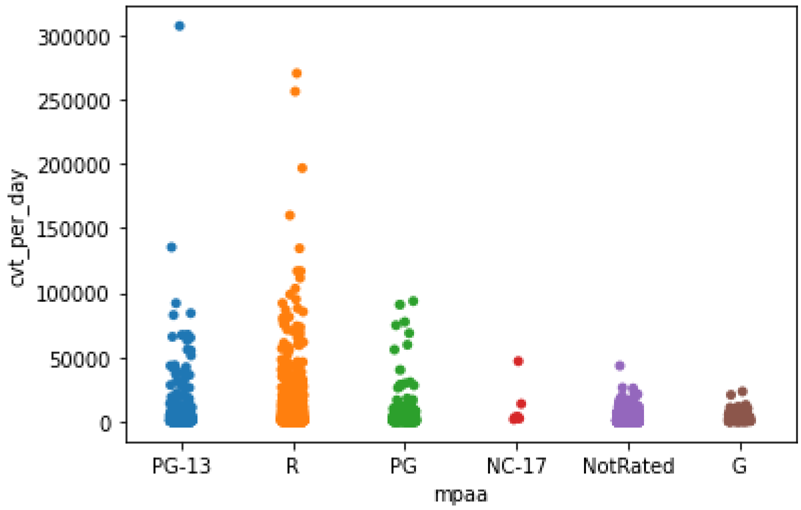
1.3.1 Distribution of standard categorical features

```
sns.stripplot(x='import_id', y='cvt_per_day', data=TV,jitter=True)
plt.show()
print(TV['import_id'].value_counts())
```



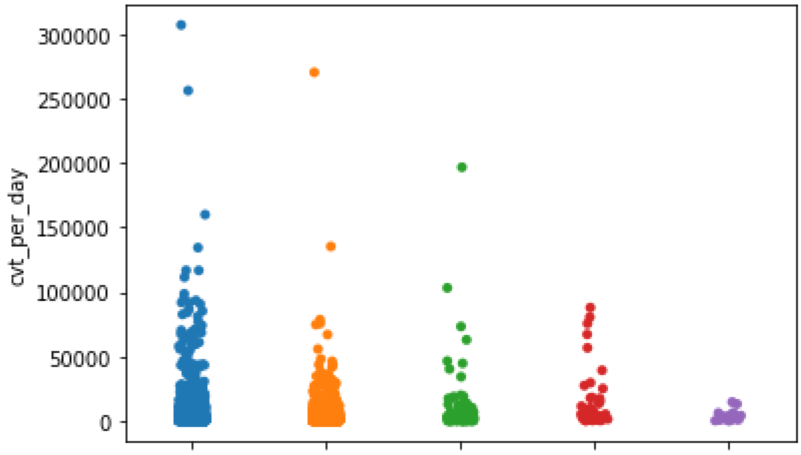

```
other        2963
lionsgate     677
mgm           445
```

```
sns.stripplot(x='mpaa', y='cvt_per_day', data=TV,jitter=True)
plt.show()
print(TV['mpaa'].value_counts())
```




```
NotRated    2158
R           1158
PG-13        426
PG           353
G            125
NC-17          6
Name: mpaa, dtype: int64
```

```
sns.stripplot(x='awards', y='cvt_per_day', data=TV, jitter=True)
plt.show()
print(TV['awards'].value_counts())
```



After very basic Exploratory Data Analysis, we have to do some data cleaning and data preprocessing. We need three steps to finish this. First, we need to encode the categorical feature. Second, we need to impute the missing value for both numeric and categorical feature. Third, we need to scale out feature, which can be better for our models' performance.

```
Name: awards, dtype: int64
```

1.3.2 Distribution of splited genres

Some videos belongs to more than 1 genre, the genre of each video is splited, this would help emphasize the effect of each individual genre.

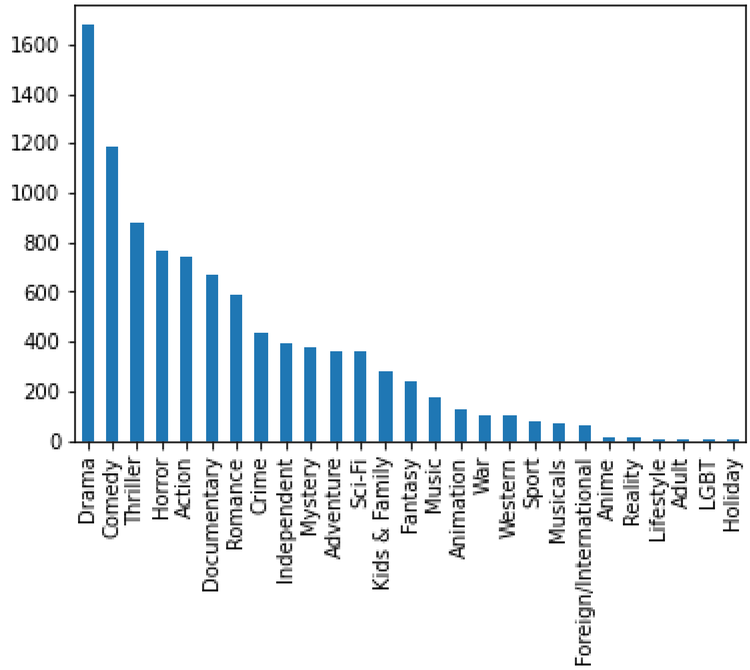
```
# generes explore, split the genre of each video
gen_split = TV['genres'].str.get_dummies(sep=',').sum()
print(gen_split)
```

```
Action                   739
Adult                      3
Adventure                363
Animation                129
Anime                     11
Comedy                  1184
Crime                    437
Documentary              671
Drama                   1677
Fantasy                  243
Foreign/International     64
Holiday                    1
Horror                   762
Independent              393
Kids & Family            280
LGBT                       2
Lifestyle                  7
Music                    171
Musicals                  68
Mystery                  375
Reality                    9
Romance                  591
Sci-Fi                   363
Sport                     77
Thriller                 879
War                      102
```

```
Western              102
dtype: int64
```

```
gen_split.sort_values(ascending=False).plot.bar()
```

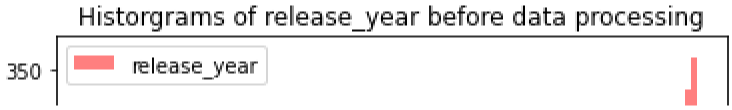
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9b2962ef60>
```



1.3.3 Distribution of release_year

The release year of video varies through a wide range. Considering the popularity of a video usually decays over time, the release_year should be bucketed based on the release_year range.

```
plt.hist(TV['release_year'].values, bins = range(1910, 2017, 1), alpha = 0.5, color
plt.legend(loc ='upper left')
plt.title('Historgrams of release_year before data processing')
plt.xlabel('release_year')
plt.ylabel('Count')
plt.show()
```



Part 2: Feature Preprocessing

2.1 Categorical features

There are 5 categorical features: import_id, mpaa, awards, genres, and release_year. There is no missing data in them. They can be converted into dummy/indicators.

The first 3 have relatively small sub-types, they can be easily converted to dummies.

The 'genres' have 27 different sub-types, 6 of them are rarely observed (refer to previous section). It's reasonable to group these 6 into

1. Note: a video may have more than one genre, in the feature preprocessing, all genres are handled individually.
2. The release_year is binned into 10 buckets based on the year range between 1917 and 2017.

```
# Convert 3 Categorical variables into dummy variables
d_import_id = pd.get_dummies(TV['import_id']).astype(np.int64)
d_mpaa = pd.get_dummies(TV['mpaa']).astype(np.int64)
d_awards = pd.get_dummies(TV['awards']).astype(np.int64)
```

```
d_awards.head(10)
```

	BAFTA	Golden Globe	Oscar	no award	other award
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	0	0	1
3	0	0	1	0	0
4	0	0	0	0	1
5	0	0	0	1	0
6	0	0	0	0	1
7	0	0	0	0	1
8	0	0	0	0	1
9	0	0	0	0	1

```
# Convert 'genres' into dummy variables
```

```
# Convert 'genres' into dummy variables
d_genres=TV['genres'].str.get_dummies(sep=',').astype(np.int64)
d_genres['Misc_genres']=d_genres['Anime']|d_genres['Reality']|d_genres['Lifestyle']
d_genres.drop(['Anime', 'Reality','Lifestyle', 'Adult','LGBT','Holiday'], inplace=T
```

```
d_genres.head(10)
```

	Action	Adventure	Animation	Comedy	Crime	Documentary	Drama	Fantasy	Fo
0	1	0	0	0	0	0	1	0	
1	0	0	0	1	1	0	0	0	
2	0	0	0	0	1	0	1	0	
3	1	0	0	0	0	1	1	0	
4	0	0	0	0	1	1	0	0	
5	0	0	0	1	0	0	0	0	
6	1	1	0	0	0	0	0	0	
7	0	0	0	0	0	0	1	0	
8	0	0	0	0	1	0	0	0	
9	1	1	0	0	0	0	0	1	

```
d_genres.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4226 entries, 0 to 4225
Data columns (total 22 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Action                 4226 non-null   int64
 1   Adventure              4226 non-null   int64
 2   Animation              4226 non-null   int64
 3   Comedy                 4226 non-null   int64
 4   Crime                  4226 non-null   int64
 5   Documentary            4226 non-null   int64
 6   Drama                  4226 non-null   int64
 7   Fantasy                4226 non-null   int64
 8   Foreign/International  4226 non-null   int64
 9   Horror                 4226 non-null   int64
 10  Independent            4226 non-null   int64
 11  Kids & Family          4226 non-null   int64
 12  Music                  4226 non-null   int64
 13  Musicals               4226 non-null   int64
 14  Mystery                4226 non-null   int64
 15  Romance                4226 non-null   int64
 16  Sci-Fi                 4226 non-null   int64
 17  Sport                  4226 non-null   int64
 18  Thriller               4226 non-null   int64
 19  War                    4226 non-null   int64
 20  Western                4226 non-null   int64
 21  Misc_genres            4226 non-null   int64
```

```
dtypes: int64(22)
memory usage: 726.5 KB
```

release_year

```
TV['release_year'].quantile([0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9])
```

```
0.100   1974.000
0.200   1991.000
0.300   2001.000
0.400   2006.000
0.500   2008.000
0.600   2010.000
0.700   2012.000
0.800   2013.000
0.900   2014.000
Name: release_year, dtype: float64
```

Look for the quantile to find the years to divide them into 10 pieces.

```
# bin release_year and convert into dummies
bin_year = [1916, 1974, 1991, 2001, 2006, 2008, 2010, 2012, 2013, 2014, 2017]
year_range = ['1916-1974', '1974-1991', '1991-2001', '2001-2006','2006-2008','2008-
              '2013-2014','2014-2017']
year_bin = pd.cut(TV['release_year'], bin_year, labels=year_range)
d_year = pd.get_dummies(year_bin).astype(np.int64)
```

```
# new dataframe, drop the previous categorical features, add new dummy variables, c

temp_tv=TV.drop(['import_id', 'mpaa','awards','genres', 'release_year'], axis=1)

newTV = pd.concat([temp_tv, d_import_id, d_mpaa, d_awards, d_genres, d_year], axis=
print(newTV.head())
```

```
2    361899   256165.867                              1
3    308314   196622.721                              3
4    307201   159841.652                              1

   weighted_horizontal_poition  imdb_votes    budget  boxoffice  imdb_rating
0                            3       69614  15000000   42930462        6.500
1                            3       46705  15000000    3301046        6.500
2                            3      197596  26000000   37397291        7.300
3                            4      356339  15000000   15700000        7.600
4                            3       46720  27220000    8551228        6.400

   duration_in_mins  metacritic_score  star_category  lionsgate  mgm  other  \
0           112.301                51          1.710          1    0      0
1            94.983                41          3.250          1    0      0
2           115.764                58          2.647          0    0      1
3           130.704                94          1.667          1    0      0

4           105.546                37          3.067          1    0      0

   paramount  G  NC-17  NotRated  PG  PG-13  R  BAFTA  Golden Globe  Oscar  \
```

```
0          0  0      0          0   0       1  0       0               0       0
1          0  0      0          0   0       0  1       0               0       0
2          0  0      0          0   0       0  1       0               0       0
3          0  0      0          0   0       0  1       0               0       1
4          0  0      0          0   0       0  1       0               0       0

   no award  other award  Action  Adventure  Animation  Comedy  Crime  \
0         0            1       1          0          0       0      0
1         1            0       0          0          0       1      1
2         0            1       0          0          0       0      1
3         0            0       1          0          0       0      0
4         0            1       0          0          0       0      1

   Documentary  Drama  Fantasy  Foreign/International  Horror  Independent  \
0            0      1        0                      0       0            0
1            0      0        0                      0       0            0
2            0      1        0                      0       0            0
3            1      1        0                      0       0            0
4            1      0        0                      0       0            0

   Kids & Family  Music  Musicals  Mystery  Romance  Sci-Fi  Sport  Thriller
0              0      0         0        0        0       0      0         1
1              0      0         0        0        0       0      0         1
2              0      0         0        0        0       0      0         0
3              0      0         0        1        0       0      0         1
4              0      0         0        1        0       0      0         1

   War  Western  Misc_genres  1916-1974  1974-1991  1991-2001  2001-2006  \
0    0        0            0          0          0          0          0
1    0        0            0          0          0          0          0
2    0        0            0          0          0          0          0
3    1        0            0          0          0          0          0
4    0        0            0          0          0          0          0

   2006-2008  2008-2010  2010-2012  2012-2013  2013-2014  2014-2017
0          0          0          0          1          0          0
1          0          0          0          1          0          0
2          0          0          1          0          0          0
3          1          0          0          0          0          0
4          0          0          0          1          0          0
```

2.2 Missing data

Among the 10 numerical features (not include video_id), 4 features have over 25% of missing values (shown as '0', which is not possible in reality): budget, boxoffice, metacritic_score, star_category. 2 features have less than 10% of missing data: imdb_votes, imdb_rating.

There are 3242 samples have at least one missing data.

Right Now we have to deal with the missing data. According to the data info, there is no Null value in our dataset. That's good, but we have to be catious, cause zero value can be a very good candidate for missing data. So we have the check the ratio of zero value in our numeric feature

```
newTV[['budget','boxoffice','metacritic_score', 'star_category','imdb_votes', 'imdb
 = newTV[['budget','boxoffice','metacritic_score', 'star_category','imdb_votes', 'i
```

```
print(newTV.info())
```

```
 2   weighted_categorical_position  4226 non-null   int64
 3   weighted_horizontal_poition    4226 non-null   int64
 4   imdb_votes                     3882 non-null   float64
 5   budget                         1772 non-null   float64
 6   boxoffice                      1032 non-null   float64
 7   imdb_rating                    3882 non-null   float64
 8   duration_in_mins               4226 non-null   float64
 9   metacritic_score               1214 non-null   float64
 10  star_category                  2380 non-null   float64
 11  lionsgate                      4226 non-null   int64
 12  mgm                            4226 non-null   int64
 13  other                          4226 non-null   int64
 14  paramount                      4226 non-null   int64
 15  G                              4226 non-null   int64
 16  NC-17                          4226 non-null   int64
 17  NotRated                       4226 non-null   int64
 18  PG                             4226 non-null   int64
 19  PG-13                          4226 non-null   int64
 20  R                              4226 non-null   int64
 21  BAFTA                          4226 non-null   int64
 22  Golden Globe                   4226 non-null   int64
 23  Oscar                          4226 non-null   int64
 24  no award                       4226 non-null   int64
 25  other award                    4226 non-null   int64
 26  Action                         4226 non-null   int64
 27  Adventure                      4226 non-null   int64
 28  Animation                      4226 non-null   int64
 29  Comedy                         4226 non-null   int64
 30  Crime                          4226 non-null   int64
 31  Documentary                    4226 non-null   int64
 32  Drama                          4226 non-null   int64
 33  Fantasy                        4226 non-null   int64
 34  Foreign/International          4226 non-null   int64
 35  Horror                         4226 non-null   int64
 36  Independent                    4226 non-null   int64
 37  Kids & Family                  4226 non-null   int64
 38  Music                          4226 non-null   int64
 39  Musicals                       4226 non-null   int64
 40  Mystery                        4226 non-null   int64
 41  Romance                        4226 non-null   int64
 42  Sci-Fi                         4226 non-null   int64
 43  Sport                          4226 non-null   int64
 44  Thriller                       4226 non-null   int64
 45  War                            4226 non-null   int64
 46  Western                        4226 non-null   int64
 47  Misc_genres                    4226 non-null   int64
 48  1916-1974                      4226 non-null   int64
 49  1974-1991                      4226 non-null   int64
 50  1991-2001                      4226 non-null   int64
 51  2001-2006                      4226 non-null   int64
 52  2006-2008                      4226 non-null   int64
 53  2008-2010                      4226 non-null   int64
 54  2010-2012                      4226 non-null   int64
 55  2012-2013                      4226 non-null   int64
 56  2013-2014                      4226 non-null   int64
 57  2014-2017                      4226 non-null   int64
dtypes: float64(8), int64(50)
memory usage: 1.9 MB
None
```

Filling missing data with mean value

```
newTV1=newTV.copy()
newTV1['boxoffice']=newTV1['boxoffice'].fillna(newTV1['boxoffice'].mean())
newTV1['metacritic_score']=newTV1['metacritic_score'].fillna(newTV1['metacritic_sco
newTV1['star_category']=newTV1['star_category'].fillna(newTV1['star_category'].mean
newTV1['imdb_votes']=newTV1['imdb_votes'].fillna(newTV1['imdb_votes'].mean())
newTV1['imdb_rating']=newTV1['imdb_rating'].fillna(newTV1['imdb_rating'].mean())
newTV1['budget']=newTV1['budget'].fillna(newTV1['budget'].mean())
print(newTV1.info())
```

```
 2   weighted_categorical_position  4226 non-null   int64
 3   weighted_horizontal_poition    4226 non-null   int64
 4   imdb_votes                     4226 non-null   float64
 5   budget                         4226 non-null   float64
 6   boxoffice                      4226 non-null   float64
 7   imdb_rating                    4226 non-null   float64
 8   duration_in_mins               4226 non-null   float64
 9   metacritic_score               4226 non-null   float64
 10  star_category                  4226 non-null   float64
 11  lionsgate                      4226 non-null   int64
 12  mgm                            4226 non-null   int64
 13  other                          4226 non-null   int64
 14  paramount                      4226 non-null   int64
 15  G                              4226 non-null   int64
 16  NC-17                          4226 non-null   int64
 17  NotRated                       4226 non-null   int64
 18  PG                             4226 non-null   int64
 19  PG-13                          4226 non-null   int64
 20  R                              4226 non-null   int64
 21  BAFTA                          4226 non-null   int64
 22  Golden Globe                   4226 non-null   int64
 23  Oscar                          4226 non-null   int64
 24  no award                       4226 non-null   int64
 25  other award                    4226 non-null   int64
 26  Action                         4226 non-null   int64
 27  Adventure                      4226 non-null   int64
 28  Animation                      4226 non-null   int64
 29  Comedy                         4226 non-null   int64
 30  Crime                          4226 non-null   int64
 31  Documentary                    4226 non-null   int64
 32  Drama                          4226 non-null   int64
 33  Fantasy                        4226 non-null   int64
 34  Foreign/International          4226 non-null   int64
 35  Horror                         4226 non-null   int64
 36  Independent                    4226 non-null   int64
 37  Kids & Family                  4226 non-null   int64
 38  Music                          4226 non-null   int64
 39  Musicals                       4226 non-null   int64
 40  Mystery                        4226 non-null   int64
 41  Romance                        4226 non-null   int64
 42  Sci-Fi                         4226 non-null   int64
 43  Sport                          4226 non-null   int64
 44  Thriller                       4226 non-null   int64
 45  War                            4226 non-null   int64
 46  Western                        4226 non-null   int64
 47  Misc_genres                    4226 non-null   int64
 48  1916-1974                      4226 non-null   int64
```

```
 49  1974-1991                       4226 non-null   int64
 50  1991-2001                       4226 non-null   int64
 51  2001-2006                       4226 non-null   int64
 52  2006-2008                       4226 non-null   int64
 53  2008-2010                       4226 non-null   int64
 54  2010-2012                       4226 non-null   int64
 55  2012-2013                       4226 non-null   int64
 56  2013-2014                       4226 non-null   int64
 57  2014-2017                       4226 non-null   int64
dtypes: float64(8), int64(50)
memory usage: 1.9 MB
None
```

There are two most common used scaling method: normalization and standardscaler If there are no specific requirement for the range of output, we choose to use standardscaler

2.3 Feature scaling

The impact of different scaling methods on the model performance is small. In the following model training and selections, the standard scaling (sc) data is used.

2.3.1 Standard scaling

```
#Standard scaling
scale_lst = ['weighted_categorical_position', 'weighted_horizontal_poition', 'budge
             'imdb_votes','imdb_rating','duration_in_mins', 'metacritic_score','sta
newTV_sc = newTV1.copy()

sc_scale = preprocessing.StandardScaler().fit(newTV_sc[scale_lst])

sc_scale
```

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

```
newTV_sc[scale_lst] = sc_scale.transform(newTV_sc[scale_lst])
newTV_sc.head(10)
```

	video_id	cvt_per_day	weighted_categorical_position	weighted_horizontal_p
0	385504	307127.606	-1.106	
1	300175	270338.426	-1.106	
2	361899	256165.867	-1.106	
3	[illegible]	[illegible]	[illegible]	
5	389496	135076.610	-1.106	

2.3.2 MinMax scaling

```
# MinMax scaling
newTV_mm = newTV.copy()
mm_scale = preprocessing.MinMaxScaler().fit(newTV_mm[scale_lst])
newTV_mm[scale_lst] = mm_scale.transform(newTV_mm[scale_lst])
newTV_mm.head(10)
```

	video_id	cvt_per_day	weighted_categorical_position	weighted_horizontal_p
0	385504	307127.606	0.000	
1	300175	270338.426	0.000	
2	361899	256165.867	0.000	
3	308314	196622.721	0.050	
4	307201	159841.652	0.000	
5	389496	135076.610	0.000	
6	385507	134155.740	0.000	
7	380517	116906.008	0.000	
8	369857	116871.122	0.025	
9	393463	111565.597	0.025	

2.3.3 Robust scaling

```
# Robust scaling
newTV_rs = newTV.copy()
rs_scale = preprocessing.RobustScaler().fit(newTV_mm[scale_lst])
newTV_rs[scale_lst] = rs_scale.transform(newTV_rs[scale_lst])
newTV_rs.head(10)
```

	video_id	cvt_per_day	weighted_categorical_position	weighted_horizontal_p
0	385504	307127.606	7.000	
1	300175	270338.426	7.000	
2	361899	256165.867	7.000	
3	308314	196622.721	23.000	
4	307201	159841.652	7.000	
5	389496	135076.610	7.000	
6	385507	134155.740	7.000	
7	380517	116906.008	7.000	

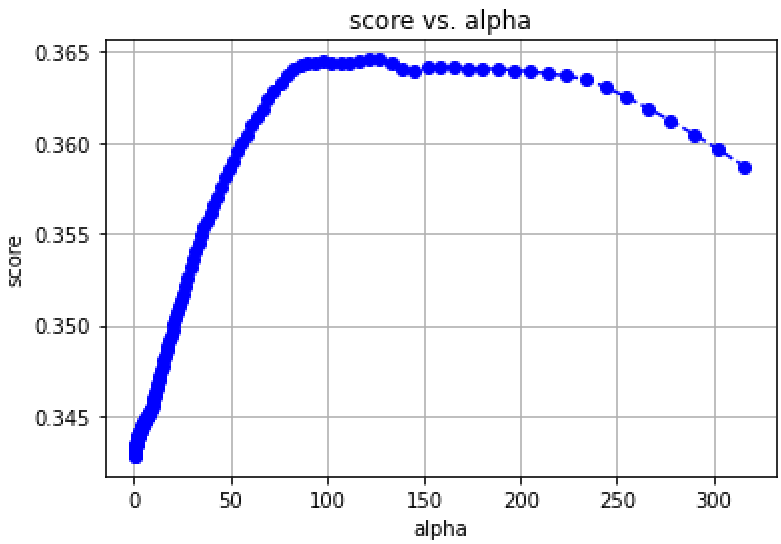
Part 3: Model Training

```
train, test = train_test_split(newTV_sc, test_size=0.15, random_state = 3)
model_train_x = train.drop(['video_id', 'cvt_per_day'], axis = 1)
model_test_x = test.drop(['video_id', 'cvt_per_day'], axis = 1)
model_train_y = train['cvt_per_day']
model_test_y = test['cvt_per_day']
```

3.1 Lasso linear regression

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.metrics import mean_squared_error, r2_score
from math import sqrt

lr_train, lr_validate = train_test_split(train, test_size=0.15, random_state = 0)

lr_train_x = lr_train.drop(['video_id', 'cvt_per_day'], axis = 1)
lr_validate_x = lr_validate.drop(['video_id', 'cvt_per_day'], axis = 1)
lr_train_y = lr_train['cvt_per_day']
lr_validate_y = lr_validate['cvt_per_day']

alphas = np.logspace (-0.3, 2.5, num=150)
# alphas= [0.000000001]
scores = np.empty_like(alphas)
opt_a = float('-inf')
max_score = float('-inf')
for i, a in enumerate(alphas):
    lasso = Lasso()
    lasso.set_params(alpha = a)
    lasso.fit(lr_train_x, lr_train_y)
    scores[i] = lasso.score(lr_validate_x, lr_validate_y)
    if scores[i] > max_score:
        max_score = scores[i]
```

```
        opt_a = a
        lasso_save = lasso
plt.plot(alphas, scores, color='b', linestyle='dashed', marker='o',markerfacecolor=
plt.xlabel('alpha')
plt.ylabel('score')
plt.grid(True)
plt.title('score vs. alpha')
plt.show()
model1_para = opt_a
print ('The optimaized alpha and score of Lasso linear is: '), opt_a, max_score
```




```
The optimaized alpha and score of Lasso linear is:
(None, 122.06107238906554, 0.36457853302954246)
```

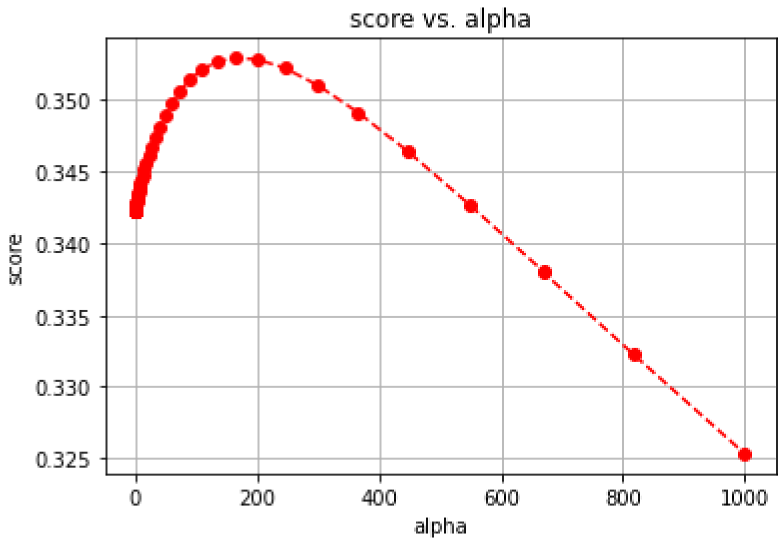
```
# combine the validate data and training data, use the optimal alpha, re-train the
lasso_f = Lasso()
lasso_f.set_params(alpha = opt_a)
lasso_f.fit(model_train_x, model_train_y)
```

```
Lasso(alpha=122.06107238906554, copy_X=True, fit_intercept=True, max_iter=100(
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

3.2 Ridge linear regression

```
# Use the same training data set as Lasso (linear features)
lr_train, lr_validate = train_test_split(train, test_size=0.15, random_state = 0)

alphas = np.logspace (-10, 3, num=150)
# alphas= [0.000000001]
scores = np.empty_like(alphas)
opt_a = float('-inf')
max_score = float('-inf')
for i, a in enumerate(alphas):
    ridge = Ridge()
    ridge.set_params(alpha = a)
```

```
    ridge.fit(lr_train_x, lr_train_y)
    scores[i] = ridge.score(lr_validate_x, lr_validate_y)
    if scores[i] > max_score:
        max_score = scores[i]
        opt_a = a
        ridge_save = ridge
plt.plot(alphas, scores, color='r', linestyle='dashed', marker='o',markerfacecolor=
plt.xlabel('alpha')
plt.ylabel('score')
plt.grid(True)
plt.title('score vs. alpha')
plt.show()
model3_para = opt_a
print ('The optimaized alpha and score of Ridge linear is: '), opt_a, max_score
```




```
The optimaized alpha and score of Ridge linear is:
(None, 163.97026580002054, 0.35296043098491625)
```

```
# add the 15% validate data, use the optimal alpha, re-train the model

ridge_f = Ridge()
ridge_f.set_params(alpha = opt_a)
ridge_f.fit(model_train_x, model_train_y)

# ridge_f is the Ridge model (linear feature), to be tested with test data.
```

```
Ridge(alpha=163.97026580002054, copy_X=True, fit_intercept=True, max_iter=None
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

3.3 Random Forest

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
rf=RandomForestRegressor(random_state=2,max_features='sqrt',n_jobs=-1)
param_grid={'n_estimators':[55,56,57,58,59,60,61,62,63,64,65],'max_depth':[15,16,17
clf=GridSearchCV(estimator=rf,param_grid=param_grid,cv=5,refit=True,n_jobs=-1,pre_d
clf.fit(model_train_x,model_train_y)
```

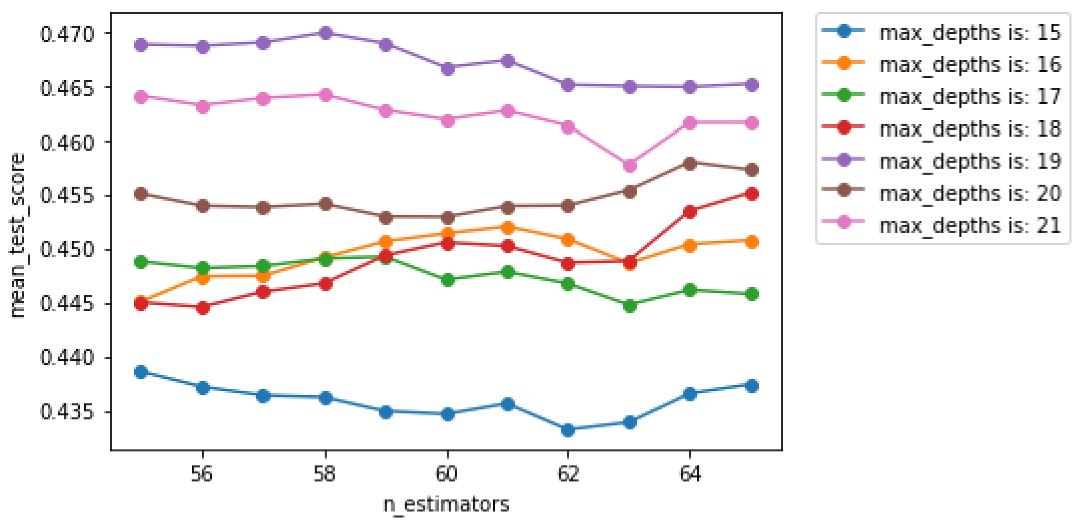
```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestRegressor(bootstrap=True, ccp_alpha=0.0,
                                             criterion='mse', max_depth=None,
                                             max_features='sqrt',
                                             max_leaf_nodes=None,
                                             max_samples=None,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
                                             min_samples_split=2,
                                             min_weight_fraction_leaf=0.0,
                                             n_estimators=100, n_jobs=-1,
                                             oob_score=False, random_state=2,
                                             verbose=0, warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'max_depth': [15, 16, 17, 18, 19, 20, 21],
                         'n_estimators': [55, 56, 57, 58, 59, 60, 61, 62, 63,
                                          64, 65]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
result=clf.cv_results_
print(result)
```

```
{'mean_fit_time': array([0.47757721, 0.42598939, 0.44056273, 0.43351402, 0.437
       0.41357117, 0.44808807, 0.44912872, 0.45942993, 0.45676956,
       0.45864143, 0.40283813, 0.44836869, 0.42409859, 0.44806843,
       0.42783947, 0.4517674 , 0.44990182, 0.46053739, 0.49713426,
       0.54971185, 0.52849622, 0.53228941, 0.55107732, 0.55231943,
       0.52121758, 0.45497069, 0.46019745, 0.52137389, 0.54528894,
       0.55122976, 0.59291906, 0.5893177 , 0.48426347, 0.52969685,
       0.54703422, 0.54467463, 0.52267203, 0.53134522, 0.54112301,
       0.5355134 , 0.54608774, 0.53617721, 0.52714334, 0.49025741,
       0.54225206, 0.54766593, 0.52618065, 0.53278103, 0.49857893,
       0.49431486, 0.52485676, 0.55420289, 0.5251668 , 0.53133631,
       0.48707099, 0.44400434, 0.49238219, 0.47847977, 0.47847638,
       0.51751757, 0.53525953, 0.52141371, 0.5534308 , 0.55240345,
       0.53698211, 0.47044544, 0.47378397, 0.46309233, 0.51365352,
       0.49258962, 0.51054635, 0.54652386, 0.52652063, 0.54407268,
       0.55739865, 0.55835481]), 'std_fit_time': array([0.06164261, 0.03412747
       0.05570594, 0.01688874, 0.01066637, 0.00560508, 0.00605744,
       0.00784283, 0.04632903, 0.01077802, 0.02759404, 0.01575712,
       0.03066517, 0.00948142, 0.00629531, 0.00562764, 0.04989976,
       0.01304167, 0.03816046, 0.04223529, 0.00117747, 0.00656398,
       0.04255394, 0.01732812, 0.00966481, 0.04553089, 0.00622447,
       0.05191692, 0.03831679, 0.0489063 , 0.04320658, 0.04281781,
       0.00746868, 0.01623398, 0.03667103, 0.0364212 , 0.01381227,
       0.03251518, 0.01529672, 0.03804058, 0.02509052, 0.0482586 ,
       0.01257513, 0.00522159, 0.02583448, 0.04939438, 0.04183682,
       0.05052129, 0.04546947, 0.01642429, 0.03261688, 0.05419846,
       0.05207288, 0.00804495, 0.04795563, 0.03703092, 0.04140358,
       0.052463  , 0.04326349, 0.03133214, 0.01204775, 0.01631887,
       0.03506841, 0.04642412, 0.03925411, 0.04216655, 0.05438394,
       0.0540297 , 0.05534793, 0.01255433, 0.04510276, 0.04542791,
       0.00754578, 0.01158356]), 'mean_score_time': array([0.1044929 , 0.10767
       0.1089354 , 0.11033363, 0.10786161, 0.10960598, 0.11134558,
       0.10848355, 0.1104898 , 0.11061625, 0.10974145, 0.11124349,
       0.11036396, 0.10869336, 0.11082916, 0.10810575, 0.10761023,
       0.10670924, 0.10715117, 0.10450702, 0.10424061, 0.10489559,
```

```
       0.1087852 , 0.11067448, 0.1089438 , 0.10839686, 0.10644011,
       0.10408044, 0.10424924, 0.10382066, 0.10848875, 0.10453858,
       0.10396228, 0.1046546 , 0.11049037, 0.1053916 , 0.10779719,
       0.10806265, 0.11009145, 0.10675073, 0.11019616, 0.1056704 ,
       0.10551953, 0.1079114 , 0.10661273, 0.10821838, 0.11164227,
       0.11104507, 0.1090579 , 0.10909724, 0.10839295, 0.11009898,
       0.10710597, 0.10783529, 0.11297274, 0.11017389, 0.10773077,
       0.10834208, 0.10927806, 0.10900574, 0.11130843, 0.1106493 ,
       0.1133296 , 0.1111629 , 0.10699282, 0.10978117, 0.11137486,
       0.11055732, 0.10761223, 0.11238141, 0.11296945, 0.10938077,
       0.1119267 , 0.10866466]), 'std_score_time': array([0.00104327, 0.004499
       0.00310245, 0.00254537, 0.00122788, 0.00253408, 0.00107547,
       0.00343908, 0.00653774, 0.00325935, 0.00445841, 0.00359862,
       0.00543624, 0.00220464, 0.00394087, 0.00273557, 0.00415199,
       0.00312379, 0.00399037, 0.00118361, 0.00086153, 0.0012608 ,
       0.00585731, 0.00555695, 0.00245776, 0.0041662 , 0.00228817,
       0.00090991, 0.00097014, 0.00012404, 0.00266233, 0.00060998,
       0.00029335, 0.00093424, 0.00562334, 0.00187267, 0.00480332,
       0.00400568, 0.0025587 , 0.00193983, 0.00449854, 0.00207882,
       0.00233357, 0.00492   , 0.00296914, 0.00306066, 0.00261617,
       0.0020381 , 0.0033122 , 0.00330732, 0.0030537 , 0.0030173 ,
       0.00323271, 0.00337628, 0.00217179, 0.0035193 , 0.00275587,
       0.0030973 , 0.00244521, 0.00340447, 0.00443212, 0.00360001,
       0.00308771, 0.00453479, 0.00291074, 0.00467101, 0.0050209 ,
```

```
max_depth=[15,16,17,18,19,20,21]
n_estimators=[55,56,57,58,59,60,61,62,63,64,65]
scores=clf.cv_results_['mean_test_score'].reshape(len(max_depth),len(n_estimators))
plt.figure(1)
plt.subplot(1,1,1)
for i,j in enumerate(max_depth):
  plt.plot(n_estimators,scores[i],'-o',label='max_depths is: '+str(j))
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.xlabel('n_estimators')
plt.ylabel('mean_test_score')
plt.show()
print('the best parameter for max_depth is: '+str(clf.best_params_['max_depth']))
print('the best parameter for n_estimators is: '+str(clf.best_params_['n_estimators
```




```
the best parameter for max_depth is: 19
the best parameter for n_estimators is: 58
```

Part 4: Model Evaluation

4.1: Evaluate all models

```
train_x = model_train_x
train_y = model_train_y
test_x = model_test_x
test_y = model_test_y
```

4.1.1 Evaluate Lasso regression model

```
#For lasso
from sklearn.linear_model import Lasso
from sklearn.metrics import  mean_squared_error
lasso=Lasso(alpha=model1_para)
lasso.fit(train_x,train_y)
pred_y=lasso.predict(test_x)
lasso_score=lasso.score(test_x,test_y)
MSE_lasso=mean_squared_error(test_y,pred_y)
RMSE_lasso=np.sqrt(MSE_lasso)
print ('lasso score: ', lasso_score)
print ('Mean square error of lasso: ', MSE_lasso)
print ('Root mean squared error of lasso:', RMSE_lasso)
```

```
lasso score:  0.09954927178753703
Mean square error of lasso:  238953191.99910036
Root mean squared error of lasso: 15458.110880670392
```

4.1.2 Evaluate Ridge Regression model

```
#for ridge
from sklearn.metrics import  mean_squared_error
ridge=Ridge(alpha=model3_para)
ridge.fit(train_x,train_y)
pred_y=ridge.predict(test_x)
ridge_score=ridge.score(test_x,test_y)
MSE_ridge=mean_squared_error(test_y,pred_y)
RMSE_ridge=np.sqrt(MSE_ridge)
print ('ridge score: ', ridge_score)
print ('Mean square error of ridge: ', MSE_ridge)
print ('Root mean squared error of ridge:', RMSE_ridge)
```

```
ridge score:  0.11371374943726809
Mean square error of ridge:  235194355.4060952
Root mean squared error of ridge: 15336.047580980414
```

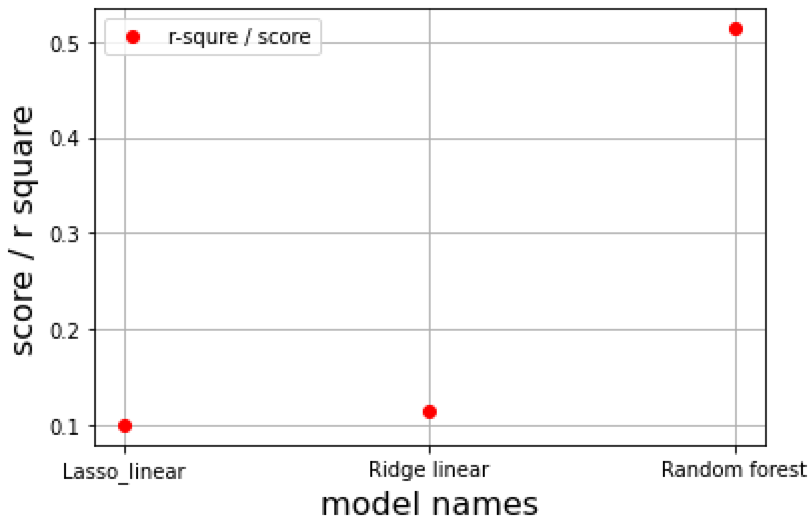
4.1.3 Evaluate Randomforest model

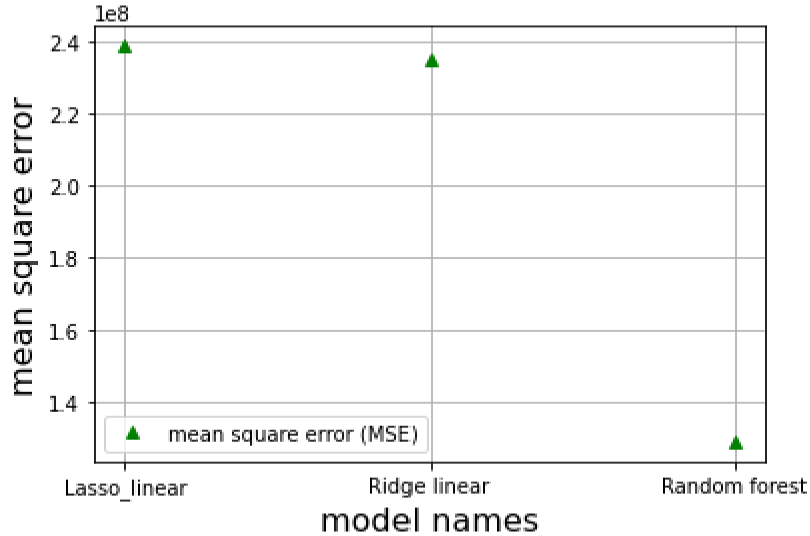
```
#For randomforest regression
from sklearn.ensemble import RandomForestRegressor
rf=RandomForestRegressor(n_estimators=clf.best_params_['n_estimators'],max_depth=cl
rf.fit(train_x,train_y)
pred_y=rf.predict(test_x)
rf_score=rf.score(test_x,test_y)
MSE_rf=mean_squared_error(test_y,pred_y)
RMSE_rf=np.sqrt(MSE_rf)
print ('rf score: ', rf_score)
print ('Mean square error of rf: ', MSE_rf)
print ('Root mean squared error of rf:', RMSE_rf)
```

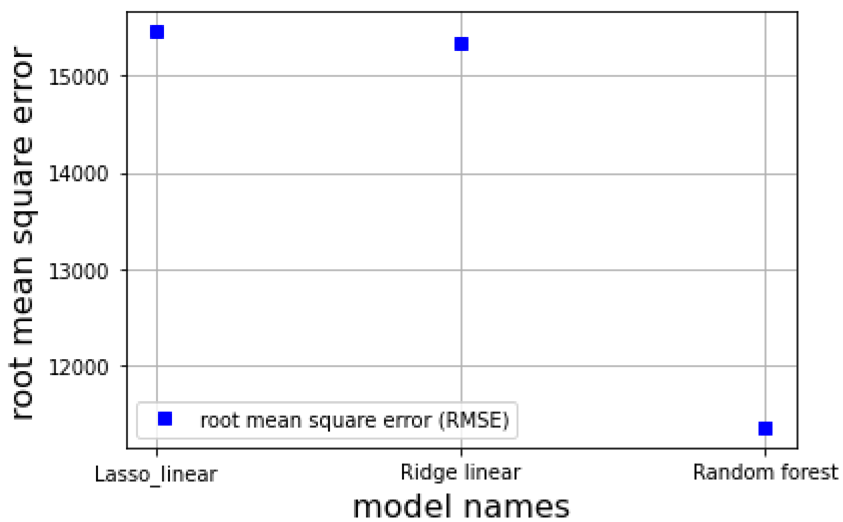
```
rf score:  0.5139461304918905
Mean square error of rf:  128984429.64563024
Root mean squared error of rf: 11357.131224285042
```

4.2 Model comparison

```
lst_score = [lasso_score, ridge_score, rf_score]
MSE_lst =  [MSE_lasso, MSE_ridge, MSE_rf]
RMSE_lst =  [RMSE_lasso, RMSE_ridge, RMSE_rf]
model_lst = ['Lasso_linear', 'Ridge linear', 'Random forest']

plt.figure(1)
plt.plot(model_lst, lst_score, 'ro')
plt.legend(['r-squre / score'])
plt.xlabel('model names',fontsize =16)
plt.ylabel('score / r square', fontsize =16)
plt.grid(True)
plt.show()

plt.figure(2)
plt.plot(model_lst, MSE_lst, 'g^')
plt.legend(['mean square error (MSE)'])
plt.xlabel('model names', fontsize =16)
plt.ylabel('mean square error', fontsize =16)
plt.grid(True)
plt.show()

plt.figure(3)
plt.plot(model_lst, RMSE_lst, 'bs')
plt.legend(['root mean square error (RMSE)'])
plt.xlabel('model names', fontsize =16)
plt.ylabel('root mean square error', fontsize =16)
plt.grid(True)
plt.show()
```







4.3 Feature importance

According to MSE,RMSE and R square, the Random Forest Regression has the best performance

```
importances = rf.feature_importances_
feature_name = train_x.columns.values
indices = np.argsort(importances)[::-1]
plt.figure(1)
```

```
plt.bar(feature_name[indices[:20]], importances[indices[:20]])
plt.xticks(rotation=90)
plt.show()
```