```
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)


TV=pd.read_table('/content/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
```

```
#Part1 Data Exploration


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

```
no duplicated index
```

```
TV.info()
print(TV.drop(columns =['video_id','release_year'],axis = 1).describe(percentiles = [0.1,0.25,0.5,0.75,0.95]))
(TV==0).sum(axis = 0)/TV.shape[0]
```

```
<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
```

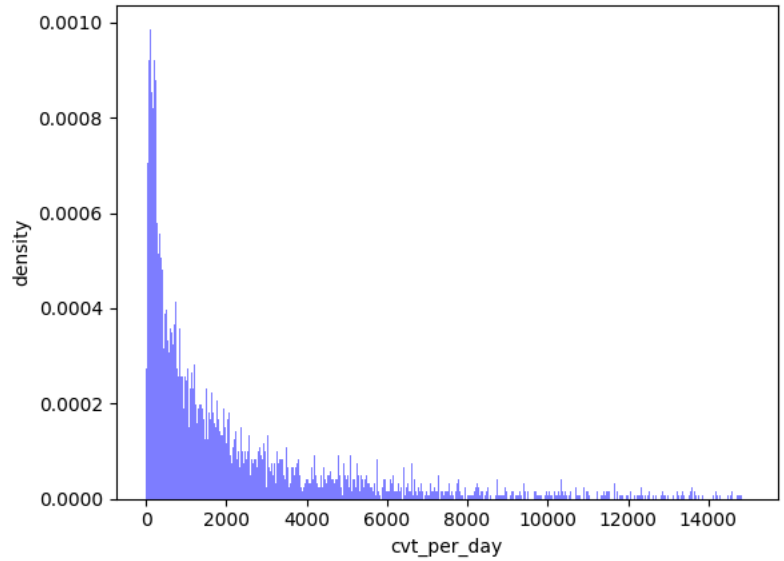
```
 13  awards                        4226 non-null   object
 14  mpaa                          4226 non-null   object
 15  star_category                 4226 non-null   float64
dtypes: float64(4), int64(8), object(4)
memory usage: 528.4+ KB
       cvt_per_day  weighted_categorical_position  \
count     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
```

```
plt.hist(TV['cvt_per_day'],bins = range(0,15000,30),color = 'blue',label='cvt_per_day',density = True,alpha = 0.5)
plt.xlabel('cvt_per_day')
plt.ylabel('density')
```

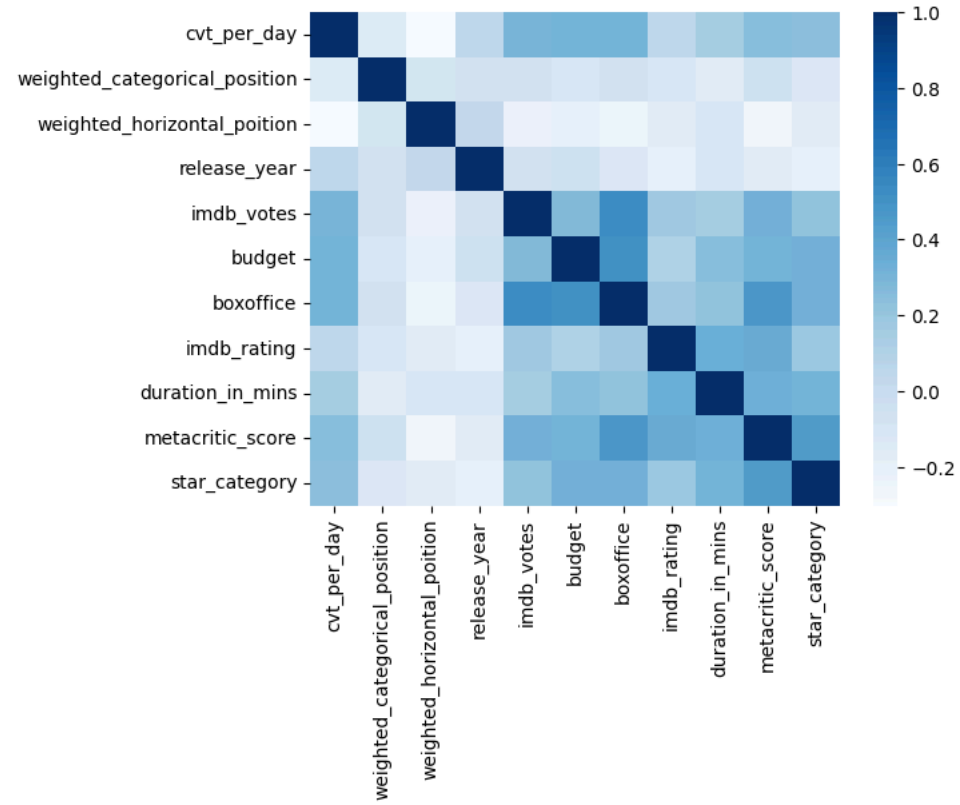
```
Text(0, 0.5, 'density')
```




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

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

```
<Axes: >
```



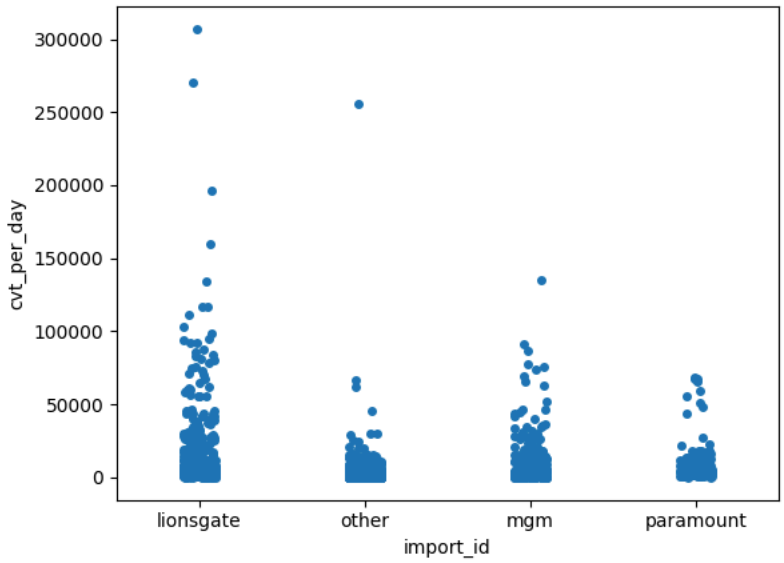

```
corr
```

	cvt_per_day	weighted_categorical_position	weighted_horizontal_poition	release_year	imdb_votes	
cvt_per_day	1.000	−0.148	−0.302	0.046	0.298	
weighted_categorical_position	−0.148	1.000	−0.084	−0.069	−0.064	
weighted_horizontal_poition	−0.302	−0.084	1.000	0.027	−0.221	
release_year	0.046	−0.069	0.027	1.000	−0.083	
imdb_votes	0.298	−0.064	−0.221	−0.083	1.000	
budget	0.316	−0.090	−0.211	−0.040	0.273	
boxoffice	0.312	−0.074	−0.245	−0.128	0.531	
imdb_rating	0.059	−0.116	−0.162	−0.199	0.163	
duration_in_mins	0.152	−0.174	−0.102	−0.097	0.156	
metacritic_score	0.249	−0.044	−0.255	−0.167	0.322	
star_category	0.247	−0.123	−0.168	−0.198	0.225	

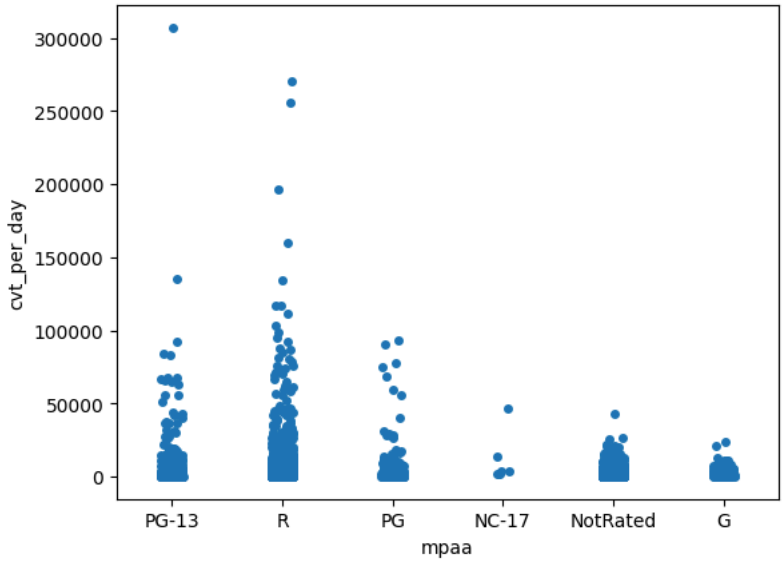
```
sns.stripplot(x='import_id',y='cvt_per_day',data = TV,jitter = True)
plt.show()
print(TV['import_id'].value_counts())

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

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



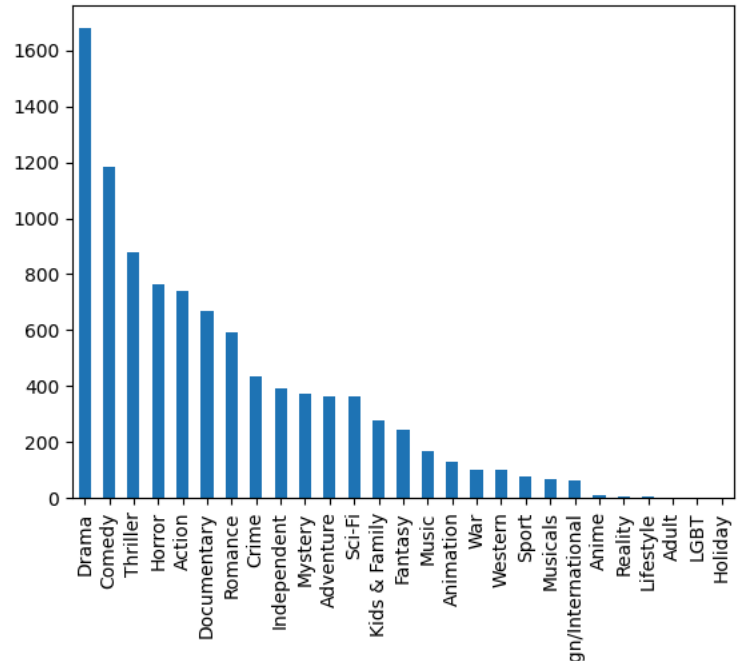

```
import_id
other        2963
lionsgate     677
mgm           445
paramount     141
Name: count, dtype: int64
```



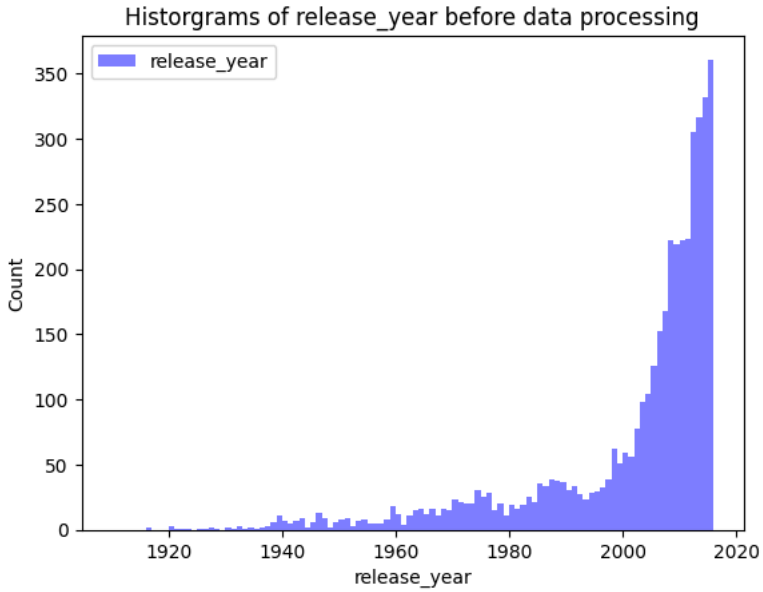

```
mpaa
```

```
gen_split = TV['genres'].str.get_dummies(sep = ',').sum()
print(gen_split)
gen_split.sort_values(ascending = False).plot.bar()
```

```
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
<Axes: >
```




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




```
#Part2: Data Preprocessing


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_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['Adult']|d_genres['LGBT']|d_genres[
d_genres.drop(['Anime', 'Reality','Lifestyle', 'Adult','LGBT','Holiday'], inplace=True, axis=1)


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
```

```
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','2008-2010','2010-2012','2012-2013',
              '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)


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=1)
print(newTV.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  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
```

```
newTV[['budget','boxoffice','metacritic_score', 'star_category','imdb_votes', 'imdb_rating']] = newTV[['budget','boxoffice','met
print(newTV.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4226 entries, 0 to 4225
Data columns (total 58 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   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
```

```
newTV1=newTV.copy()
newTV1['boxoffice']=newTV1['boxoffice'].fillna(newTV1['boxoffice'].mean())
newTV1['metacritic_score']=newTV1['metacritic_score'].fillna(newTV1['metacritic_score'].mean())
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())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4226 entries, 0 to 4225
Data columns (total 58 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   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
```

```
scale_lst = ['weighted_categorical_position', 'weighted_horizontal_poition', 'budget','boxoffice',
             'imdb_votes','imdb_rating','duration_in_mins', 'metacritic_score','star_category']
newTV_sc = newTV1.copy()
sc_scale = preprocessing.StandardScaler().fit(newTV_sc[scale_lst])
newTV_sc[scale_lst] = sc_scale.transform(newTV_sc[scale_lst])
newTV_sc.head()
```

	video_id	cvt_per_day	weighted_categorical_position	weighted_horizontal_poition	imdb_votes	budget	boxoffice	imdb_rati
0	385504	307127.606	−1.106	−2.116	1.984	1.470	4.696	0.5
1	300175	270338.426	−1.106	−2.116	1.258	1.470	−1.022	0.5
2	361899	256165.867	−1.106	−2.116	6.043	3.108	3.898	1.1
3	308314	196622.721	−0.780	−2.032	11.077	1.470	0.767	1.3
4	307201	159841.652	−1.106	−2.116	1.258	3.290	−0.265	0.4

```
#Part3:Model Training
```

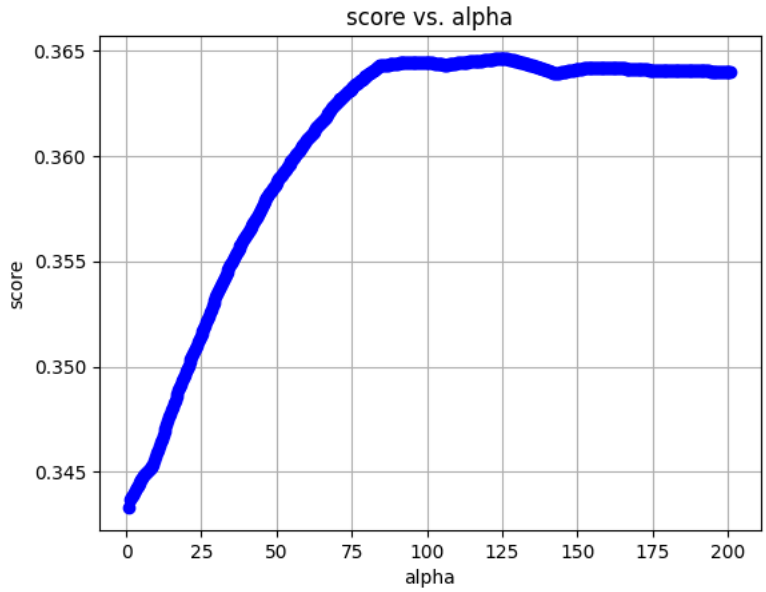
```
train,test = train_test_split(newTV_sc, test_size=0.15, random_state = 3)
model_train_x = train.drop(['video_id', 'cvt_per_day'], axis = 1)
model_test_x = test.drop(['video_id', 'cvt_per_day'], axis = 1)
model_train_y = train['cvt_per_day']
model_test_y = test['cvt_per_day']
```

```
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 #not used
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.linspace(1,201,num=500)
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='blue', markersize=6)
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
```



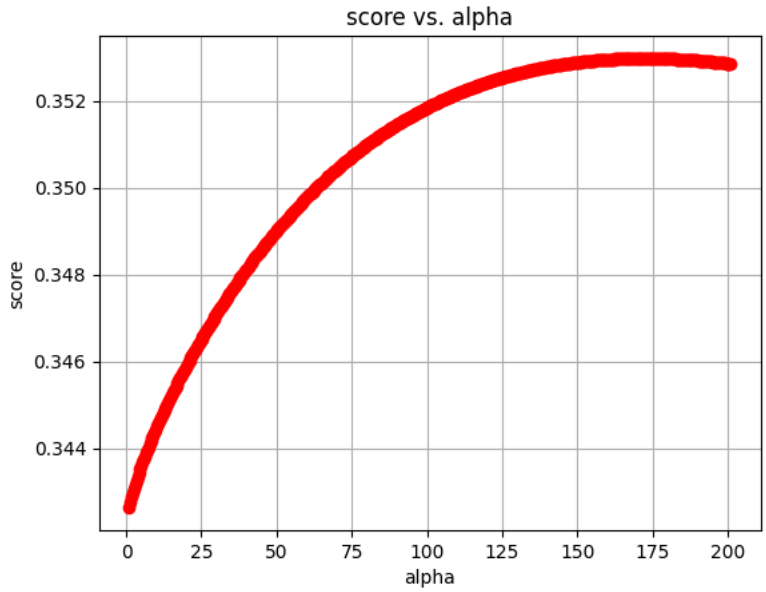

```
The optimaized alpha and score of Lasso linear is:
(None, 125.64929859719439, 0.3646194368519905)
```

```
lasso_f = Lasso()
lasso_f.set_params(alpha = opt_a)
lasso_f.fit(model_train_x,model_train_y)
```

```
Lasso
Lasso(alpha=125.64929859719439)
```

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

alphas = np.linspace (1, 201, num=500)
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 #optional code
plt.plot(alphas, scores, color='r', linestyle='dashed', marker='o',markerfacecolor='r', markersize=6)
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, 172.5430861723447, 0.3529726994066543)
```

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

```
Ridge
Ridge(alpha=172.5430861723447)
```

```
#Random Forest
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
rf=RandomForestRegressor(random_state=2,max_features='sqrt')
param_grid={'n_estimators':[55,56,57,58,59,60,61,62,63,64,65],'max_depth':[15,16,17,18,19,20,21]}
clf=GridSearchCV(estimator=rf,param_grid=param_grid,cv=5)
clf.fit(model_train_x,model_train_y)
```

```
GridSearchCV
estimator: RandomForestRegressor
RandomForestRegressor
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
rf=RandomForestRegressor(random_state=2,max_features='sqrt')
param_grid={'n_estimators':[55,56,57,58,59,60,61,62,63,64,65],'max_depth':[15,16,17,18,19,20,21]}
clf=GridSearchCV(estimator=rf,param_grid=param_grid,cv=5)
clf.fit(model_train_x,model_train_y)
```

```
GridSearchCV
estimator: RandomForestRegressor
RandomForestRegressor
```

```
result=clf.cv_results_#cross validation
print(result)
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']))
```

```
    {'mean_fit_time': array([0.48871288, 0.34127173, 0.36331453, 0.35074887, 0.2824326 ,
           0.28462601, 0.28707218, 0.2891573 , 0.29668045, 0.30391083,
           0.45797677, 0.55077024, 0.73036962, 0.36840377, 0.29132252,
           0.44167824, 0.32906651, 0.30508847, 0.31440158, 0.32778602,
           0.32528243, 0.32314711, 0.41279831, 0.33822794, 0.33572397,
           0.47798963, 0.31799636, 0.32453346, 0.37578235, 0.44385486,
           0.33858433, 0.33656759, 0.343187  , 0.29983821, 0.30618105,
           0.38463459, 0.41838102, 0.32289791, 0.32392144, 0.33134871,
           0.33867221, 0.34519706, 0.47674098, 0.38127756, 0.30237479,
           0.3110435 , 0.32417979, 0.31906018, 0.32807488, 0.50364695,
           0.33903918, 0.34099755, 0.35492692, 0.36419516, 0.36162186,
           0.39788122, 0.40192938, 0.44031811, 0.33020692, 0.34743114,
           0.34704003, 0.39519005, 0.48194208, 0.35881195, 0.36379843,
           0.36855321, 0.31787381, 0.32958012, 0.6098196 , 0.3372663 ,
           0.34464335, 0.35344071, 0.35422697, 0.35721207, 0.49845595,
           0.40733309, 0.38767047]), 'std_fit_time': array([0.05841355, 0.15013664, 0.06831524, 0.068133  , 0.00224433,
           0.00297816, 0.00359982, 0.00567222, 0.00405806, 0.00626311,
           0.0469459 , 0.11477585, 0.32297904, 0.09038757, 0.00813619,
           0.0447284 , 0.05226372, 0.00540005, 0.00574631, 0.00843265,
           0.00483612, 0.00549575, 0.06383155, 0.06809719, 0.03261244,
           0.13891813, 0.01419031, 0.00770351, 0.06661008, 0.07464703,
           0.01463415, 0.00644277, 0.00889729, 0.00657116, 0.00746336,
           0.07725396, 0.07914434, 0.00843585, 0.00444429, 0.00493231,
           0.00438061, 0.00535994, 0.07451004, 0.06460025, 0.00511131,
           0.00828891, 0.00995646, 0.00919039, 0.00860586, 0.04490847,
           0.0056474 , 0.00882243, 0.00695062, 0.01027428, 0.00424317,
           0.07441649, 0.08372043, 0.23202695, 0.00667646, 0.00772491,
           0.00474503, 0.07971608, 0.08514921, 0.00744576, 0.00341079,
           0.00915804, 0.00836826, 0.0053799 , 0.30574313, 0.00699344,
           0.00763438, 0.00596805, 0.00536745, 0.00715706, 0.08564482,
           0.06987731, 0.00484414]), 'mean_score_time': array([0.03334899, 0.01103878, 0.01339788, 0.01615286, 0.01144834,
           0.01143084, 0.01160531, 0.01191177, 0.01199665, 0.01216846,
           0.02032852, 0.0183217 , 0.03261409, 0.01355948, 0.0122786 ,
           0.0173326 , 0.01204739, 0.01228456, 0.01301117, 0.013344  ,
           0.0126895 , 0.01275282, 0.01569467, 0.0148427 , 0.01293821,
           0.02162042, 0.0136447 , 0.01334138, 0.01601725, 0.0162312 ,
           0.0141397 , 0.01362276, 0.01397142, 0.01230149, 0.01259036,
           0.01665177, 0.01599765, 0.01404052, 0.01582923, 0.01539712,
           0.01398273, 0.0136445 , 0.01926646, 0.01384301, 0.01390653,
           0.01285748, 0.01400476, 0.01307354, 0.01323481, 0.01974568,
           0.01518383, 0.01384473, 0.01600204, 0.01446428, 0.01520896,
           0.01540728, 0.01590099, 0.0150146 , 0.0138185 , 0.01476068,
           0.01488824, 0.0173924 , 0.01848407, 0.01495929, 0.01670856,
           0.01490612, 0.01360373, 0.01385655, 0.0269887 , 0.01406159,
           0.01425529, 0.01444983, 0.01505914, 0.01548953, 0.01990237,
           0.01653748, 0.01533999]), 'std_score_time': array([0.01729482, 0.00036757, 0.00173782, 0.00365241, 0.00050654,
           0.00023197, 0.00016198, 0.00048558, 0.00030423, 0.00010785,
           0.00354249, 0.0030123 , 0.01301285, 0.00258149, 0.00079863,
           0.00195687, 0.00020903, 0.00019065, 0.00066716, 0.0009301 ,
           0.00016276, 0.00012619, 0.00177489, 0.00321502, 0.00064102,
           0.01102188, 0.00186848, 0.00113549, 0.00387483, 0.00312693,
           0.00171163, 0.0006779 , 0.00056237, 0.00013693, 0.00027076,
           0.00435465, 0.00366157, 0.00180942, 0.00229355, 0.00328411,
           0.0006226 , 0.00028863, 0.00389303, 0.00031292, 0.00198435,
           0.00012574, 0.00103442, 0.00012389, 0.00023884, 0.00219899,
           0.00275881, 0.00027995, 0.00196617, 0.0006295 , 0.00084008,
           0.00198744, 0.00316104, 0.0021497 , 0.0003002 , 0.00150857,
```

```
#Part4:Model Evaluation
           0.00027932, 0.00018689, 0.00070895, 0.00114525, 0.00326565,
```

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

```
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: ', ridge_score)
print ('Mean square error of ridge: ', MSE_ridge)
print ('Root mean squared error of ridge:', RMSE_ridge)
```

```
lasso score:  0.1138238752321652
Mean square error of ridge:  235165131.2526933
Root mean squared error of ridge: 15335.094758516925
```

```
                  False, False, False, False, False, False, False, False,
```

```
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:  0.1138238752321652
Mean square error of ridge:  235165131.2526933
Root mean squared error of ridge: 15335.094758516925
```

```
      0.43135333, 0.43004306, 0.43085472, 0.43410635, 0.43377452,
```

```
from sklearn.ensemble import RandomForestRegressor
rf=RandomForestRegressor(n_estimators=clf.best_params_['n_estimators'],max_depth=clf.best_params_['max_depth'],max_features='sqr
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:  0.5169719005665712
Mean square error of rf:  128181479.08437566
Root mean squared error of rf: 11321.725976386095
```

```
#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()
```