SVM Regression

In [2]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [3]:

```
df = pd.read_csv("Movie_regression.csv", header=0)
```

In [4]:

```
df.head()
```

Out[4]:

	Marketing expense	Production expense	Multiplex coverage	Budget	Movie_length	Lead_ Actor_Rating	Lead_Actress_rating	Director_rating	Producer_rating	Crit
0	20.1264	59.62	0.462	36524.125	138.7	7.825	8.095	7.910	7.995	
1	20.5462	69.14	0.531	35668.655	152.4	7.505	7.650	7.440	7.470	
2	20.5458	69.14	0.531	39912.675	134.6	7.485	7.570	7.495	7.515	
3	20.6474	59.36	0.542	38873.890	119.3	6.895	7.035	6.920	7.020	
4	21.3810	59.36	0.542	39701.585	127.7	6.920	7.070	6.815	7.070	

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 18 columns):
Marketing expense       506 non-null float64
Production expense      506 non-null float64
Multiplex coverage      506 non-null float64
Budget                  506 non-null float64
Movie_length            506 non-null float64
Lead_ Actor_Rating      506 non-null float64
Lead_Actress_rating     506 non-null float64
Director_rating         506 non-null float64
Producer_rating         506 non-null float64
Critic_rating           506 non-null float64
Trailer_views           506 non-null int64
3D_available            506 non-null object
Time_taken              494 non-null float64
Twitter_hastags         506 non-null float64
Genre                   506 non-null object
Avg_age_actors          506 non-null int64
Num_multiplex           506 non-null int64
Collection              506 non-null int64
dtypes: float64(12), int64(4), object(2)
memory usage: 71.2+ KB
```

Missing Value Imputation

In [6]:

```
df['Time_taken'].mean()
```

Out[6]:

```
157.39149797570855
```

In [7]:

```
df['Time_taken'].fillna(value = df['Time_taken'].mean(), inplace = True)
```

In [8]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 18 columns):
Marketing expense       506 non-null float64
Production expense      506 non-null float64
Multiplex coverage      506 non-null float64
Budget                  506 non-null float64
Movie_length            506 non-null float64
Lead_ Actor_Rating      506 non-null float64
Lead_Actress_rating     506 non-null float64
Director_rating         506 non-null float64
Producer_rating         506 non-null float64
Critic_rating           506 non-null float64
Trailer_views           506 non-null int64
3D_available            506 non-null object
Time_taken              506 non-null float64
Twitter_hastags         506 non-null float64
Genre                   506 non-null object
Avg_age_actors          506 non-null int64
Num_multiplex           506 non-null int64
Collection              506 non-null int64
dtypes: float64(12), int64(4), object(2)
memory usage: 71.2+ KB
```

Dummy Variable Creation

In [9]:

```
df.head()
```

Out[9]:

	Marketing expense	Production expense	Multiplex coverage	Budget	Movie_length	Lead_ Actor_Rating	Lead_Actress_rating	Director_rating	Producer_rating	Crit
0	20.1264	59.62	0.462	36524.125	138.7	7.825	8.095	7.910	7.995	
1	20.5462	69.14	0.531	35668.655	152.4	7.505	7.650	7.440	7.470	
2	20.5458	69.14	0.531	39912.675	134.6	7.485	7.570	7.495	7.515	
3	20.6474	59.36	0.542	38873.890	119.3	6.895	7.035	6.920	7.020	
4	21.3810	59.36	0.542	39701.585	127.7	6.920	7.070	6.815	7.070	

In [10]:

```
df = pd.get_dummies(df,columns = ["3D_available","Genre"],drop_first = True)
```

In [11]:

```
df.head()
```

Out[11]:

	Marketing expense	Production expense	Multiplex coverage	Budget	Movie_length	Lead_ Actor_Rating	Lead_Actress_rating	Director_rating	Producer_rating	Crit
0	20.1264	59.62	0.462	36524.125	138.7	7.825	8.095	7.910	7.995	

	Marketing expense	Production expense	Multiplex coverage	Budget	Movie_length	Lead_ Actor_Rating	Lead_Actress_rating	Director_rating	Producer_rating	Crit
1	20.5462	69.14	0.531	35668.655	152.4	7.505	7.650	7.440	7.470	
2	20.5458	69.14	0.531	39912.675	134.6	7.485	7.570	7.495	7.515	
3	20.6474	59.36	0.542	38873.890	119.3	6.895	7.035	6.920	7.020	
4	21.3810	59.36	0.542	39701.585	127.7	6.920	7.070	6.815	7.070	

X-y split

In [12]:

```
X = df.loc[:,df.columns!="Collection"]
type(X)
```

Out[12]:

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

In [13]:

```
X.head()
```

Out[13]:

	Marketing expense	Production expense	Multiplex coverage	Budget	Movie_length	Lead_ Actor_Rating	Lead_Actress_rating	Director_rating	Producer_rating	Crit
0	20.1264	59.62	0.462	36524.125	138.7	7.825	8.095	7.910	7.995	
1	20.5462	69.14	0.531	35668.655	152.4	7.505	7.650	7.440	7.470	
2	20.5458	69.14	0.531	39912.675	134.6	7.485	7.570	7.495	7.515	
3	20.6474	59.36	0.542	38873.890	119.3	6.895	7.035	6.920	7.020	
4	21.3810	59.36	0.542	39701.585	127.7	6.920	7.070	6.815	7.070	

In [14]:

```
X.shape
```

Out[14]:

```
(506, 19)
```

In [15]:

```
y = df["Collection"]
type(y)
```

Out[15]:

```
pandas.core.series.Series
```

In [16]:

```
y.head()
```

Out[16]:

```
0    48000
1    43200
2    69400
3    66800
4    72400
Name: Collection, dtype: int64
```

In [17]:

```
y.shape
```

Out[17]:

```
(506,)
```

Test-Train Split

In [18]:

```
from sklearn.model_selection import train_test_split
```

In [19]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2,random_state=0)
```

In [20]:

```
X_train.head()
```

Out[20]:

	Marketing expense	Production expense	Multiplex coverage	Budget	Movie_length	Lead_ Actor_Rating	Lead_Actress_rating	Director_rating	Producer_rating	C
220	27.1618	67.40	0.493	38612.805	162.0	8.485	8.640	8.485	8.670	
71	23.1752	76.62	0.587	33113.355	91.0	7.280	7.400	7.290	7.455	
240	22.2658	64.86	0.572	38312.835	127.8	6.755	6.935	6.800	6.840	
6	21.7658	70.74	0.476	33396.660	140.1	7.065	7.265	7.150	7.400	
417	538.8120	91.20	0.321	29463.720	162.6	9.135	9.305	9.095	9.165	

In [21]:

```
X_train.shape
```

Out[21]:

```
(404, 19)
```

In [22]:

```
X_test.shape
```

Out[22]:

```
(102, 19)
```

Standardizing Data

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

In [23]:

```
from sklearn.preprocessing import StandardScaler
```

In [24]:

```
sc = StandardScaler().fit(X_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py:625:
DataConversionWarning: Data with input dtype uint8, int64, float64 were all converted to float64 b
y StandardScaler.
  return self.partial_fit(X, y)
```

In [25]:

```
X_train_std = sc.transform(X_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: DataConversionWarning: Data wi
th input dtype uint8, int64, float64 were all converted to float64 by StandardScaler.
  """Entry point for launching an IPython kernel.
```

In [26]:

```
X_test_std = sc.transform(X_test)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: DataConversionWarning: Data wi
th input dtype uint8, int64, float64 were all converted to float64 by StandardScaler.
  """Entry point for launching an IPython kernel.
```

In [27]:

```
X_test_std
```

Out[27]:

```
array([[-0.40835869, -1.12872913,  0.83336883, ...,  1.50268577,
        -0.48525664, -0.75225758],
       [ 0.71925111,  0.9988844 , -0.65283979, ...,  1.50268577,
        -0.48525664, -0.75225758],
       [-0.40257488,  0.39610829,  0.05115377, ...,  1.50268577,
        -0.48525664, -0.75225758],
       ...,
       [-0.3982601 , -0.85812418,  0.89420778, ..., -0.66547513,
        -0.48525664,  1.3293319 ],
       [-0.39934279, -0.07637654,  0.58132175, ...,  1.50268577,
        -0.48525664, -0.75225758],
       [-0.40088071, -0.36702631,  0.31189212, ..., -0.66547513,
        -0.48525664, -0.75225758]])
```

In [28]:

```
X_test
```

Out[28]:

	Marketing expense	Production expense	Multiplex coverage	Budget	Movie_length	Lead_ Actor_Rating	Lead_Actress_rating	Director_rating	Producer_rating	C
329	21.3448	61.48	0.540	35179.815	90.7	7.320	7.460	7.275	7.515	
371	204.6460	91.20	0.369	34529.880	173.5	9.310	9.525	9.320	9.505	
219	22.2850	82.78	0.450	35402.015	165.9	8.175	8.375	8.315	8.405	
403	516.0340	91.20	0.307	29713.695	169.5	9.125	9.310	9.060	9.100	
78	21.1292	80.66	0.563	34618.760	127.2	7.330	7.500	7.450	7.690	
15	32.5478	71.28	0.462	32407.870	130.0	7.700	7.825	7.640	7.840	
487	116.7134	91.20	0.417	32802.275	126.7	8.320	8.575	8.420	8.385	
340	21.2302	65.38	0.485	33152.240	132.0	7.470	7.620	7.515	7.770	
310	72.7096	74.80	0.456	27625.015	111.3	8.685	8.860	8.685	8.735	
102	24.5752	72.12	0.480	35579.775	158.9	8.485	8.725	8.635	8.725	
418	1490.6820	91.20	0.321	33091.135	173.5	9.020	9.155	9.075	9.150	
411	301.0140	91.20	0.403	36979.635	173.5	9.220	9.295	9.170	9.260	
446	145.7614	91.20	0.260	35224.255	169.9	8.885	8.980	8.890	9.105	
386	507.8760	91.20	0.300	25841.860	173.5	9.145	9.320	9.140	9.460	
162	56.6754	94.16	0.395	43340.110	171.7	8.870	9.120	8.945	8.980	
299	21.1122	59.48	0.600	39112.755	83.5	6.080	6.095	5.945	6.220	
480	136.4802	91.20	0.468	34674.310	138.2	8.260	8.310	8.140	8.445	

	Marketing expense	Production expense	Multiplex coverage	Budget	Movie_length	Lead_ Actor_Rating	Lead_Actress_rating	Director_rating	Producer_rating	C
196	20.8022	58.04	0.596	40479.285	107.6	6.210	6.400	6.245	6.525	
175	21.3328	63.10	0.490	36363.030	106.6	8.390	8.545	8.395	8.410	
37	21.6028	66.92	0.501	32496.750	115.0	8.010	8.125	7.940	8.055	
320	23.3520	69.76	0.507	35696.430	125.8	7.630	7.840	7.615	7.830	
171	66.2780	94.16	0.395	32663.400	170.8	8.690	8.820	8.725	8.985	
107	22.6234	72.12	0.480	34035.485	158.7	8.825	9.025	8.915	8.990	
278	21.5956	67.82	0.553	36007.510	105.6	7.890	7.960	7.800	8.075	
45	23.4284	68.82	0.552	31563.510	107.3	7.385	7.525	7.395	7.490	
367	290.4440	91.20	0.369	21458.965	173.5	9.235	9.305	9.230	9.215	
21	37.0408	71.28	0.462	33135.575	162.7	7.945	8.140	7.830	8.060	
153	62.9836	94.16	0.129	31713.495	172.0	9.110	9.320	9.050	9.275	
97	22.4166	60.78	0.555	44823.295	149.5	8.150	8.340	8.215	8.305	
113	24.4424	75.02	0.453	33841.060	168.9	8.705	8.875	8.725	8.600	
...	...	...	...	...	...	...	...	...	...	
65	20.7168	61.74	0.602	34940.950	91.3	6.690	6.785	6.575	6.725	
344	20.6098	62.56	0.516	38185.070	101.6	6.645	6.910	6.665	6.850	
481	134.1636	91.20	0.468	37496.250	148.4	8.280	8.370	8.310	8.380	
387	471.9420	91.20	0.300	27775.000	163.0	9.105	9.260	9.115	9.480	
233	26.6294	67.40	0.493	45812.085	143.9	8.090	8.255	8.130	8.225	
206	24.5938	76.18	0.511	35140.930	126.0	7.685	7.880	7.765	7.955	
90	20.9368	61.82	0.511	35646.435	139.6	8.430	8.535	8.425	8.425	
497	25.3676	74.38	0.415	32185.670	144.1	8.480	8.705	8.470	8.560	
239	21.8504	64.86	0.572	36696.330	115.7	6.875	6.945	6.795	7.005	
137	27.0466	98.78	0.376	35851.970	171.9	8.925	9.235	8.915	9.225	
407	259.0220	91.20	0.341	31152.440	173.5	9.270	9.460	9.305	9.390	
224	26.3066	67.40	0.496	45917.630	151.8	8.520	8.635	8.435	8.620	
225	30.5386	67.40	0.496	48467.375	156.5	8.435	8.695	8.525	8.560	
326	26.0694	69.76	0.507	35063.160	102.4	7.265	7.455	7.145	7.300	
96	22.3008	60.78	0.555	34235.465	143.1	8.210	8.325	8.230	8.245	
426	264.9440	91.20	0.416	32424.535	133.2	8.865	9.165	8.905	9.070	
159	48.5004	94.16	0.129	36163.050	173.5	9.080	9.155	9.010	9.220	
391	125.8610	91.20	0.300	33613.305	156.0	8.915	9.045	8.760	8.940	
54	20.2720	63.00	0.590	32707.840	121.1	6.300	6.365	6.200	6.500	
435	243.2080	91.20	0.260	36824.095	168.1	8.925	9.015	8.930	8.880	
254	20.9638	62.28	0.608	33929.940	105.5	5.305	5.495	5.345	5.420	
300	20.8834	59.48	0.600	38168.405	120.9	5.955	6.160	6.055	6.175	
505	20.9482	78.86	0.427	33496.650	154.3	8.640	8.880	8.680	8.790	
246	26.7966	66.72	0.569	33929.940	108.4	5.815	6.110	5.810	6.155	
374	389.9640	91.20	0.332	22986.590	173.5	9.270	9.480	9.425	9.550	
56	20.4110	56.48	0.590	35457.565	109.2	5.340	5.535	5.285	5.465	
455	115.0474	91.20	0.287	36246.375	160.0	8.695	8.790	8.630	9.015	
60	22.9864	65.26	0.547	31891.255	139.7	6.335	6.420	6.235	6.560	
213	22.8104	76.18	0.511	35413.125	105.8	7.945	8.040	7.910	8.215	
108	22.5604	72.12	0.480	35963.070	170.6	8.640	8.910	8.730	8.850	

102 rows × 19 columns

Training Regression Tree

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html

In [37]:

In [37]:

```
from sklearn.svm import SVR
svr = SVR(kernel='linear', C = 3000)
```

In [38]:

```
svr.fit(X_train_std, y_train)
```

Out[38]:

```
SVR(C=3000, cache_size=200, coef0=0.0, degree=3, epsilon=0.1,
  gamma='auto_deprecated', kernel='linear', max_iter=-1, shrinking=True,
  tol=0.001, verbose=False)
```

Predict values using trained model

In [39]:

```
y_test_pred = svr.predict(X_test_std)
y_train_pred = svr.predict(X_train_std)
```

In [40]:

```
y_test_pred
```

Out[40]:

```
array([54149.04985486, 42108.34471214, 47533.48916051, 17335.56708281,
       48583.82292107, 39306.44775797, 33505.9535726 , 43171.76493611,
       29476.26499757, 46980.75731107, 12645.08472006, 38648.42247605,
       38048.23936642,  8415.03656142, 67457.69091145, 62224.20392732,
       38975.54675176, 65880.00221637, 56742.36900527, 42713.82149416,
       53525.49453611, 39264.6242224 , 41193.07225572, 58209.33323513,
       42495.88201229, 12132.62441561, 40418.388771  , 29850.91523034,
       75048.6007429 , 43839.06117389, 36786.86517154, 37330.36790834,
       37565.19845685, 40538.47149966, 51853.9891749 , 35081.953653  ,
       25483.31757363, 36025.27585637, 36413.96001026, 36164.40005112,
       47058.89224017, 45705.55298319, 44210.84051282, 27455.32262481,
       50732.91083479, 46541.4643388 , 32853.69351744, 40270.99818314,
       15510.53146758, 52029.40170025, 41465.42768853, 38769.99820302,
       47214.74691352, 68388.92527648, 25170.79907635, 41175.69832861,
       40899.41103489, 34095.61876232, 20977.9636645 , 38804.33453059,
       41129.5063814 , 41083.49505688, 65425.92618456, 64734.65527104,
       28281.73281763, 63138.57252568, 37568.72857469, 40005.32896684,
       42164.20110828, 46623.41251417, 46160.02195304, 49356.9966042 ,
       56247.16388058, 60030.58078437, 49366.96696328, 10862.91367165,
       74305.04922386, 44623.1870363 , 59050.95988709, 37530.90896347,
       50647.30689612, 43058.5467422 , 31784.52915153, 81982.17233807,
       79971.19671076, 47402.01934049, 50372.39131487, 29749.23771566,
       50294.74817081, 37816.36662392, 33719.52821022, 34498.87865074,
       42984.72407916, 59337.27083148, 43237.0100136 , 42010.15292021,
       -5090.77801027, 52414.30389763, 37462.28212695, 32115.02376172,
       49809.79502581, 48308.98132474])
```

Model Performance

In [41]:

```
from sklearn.metrics import mean_squared_error, r2_score
```

In [42]:

```
mean_squared_error(y_test, y_test_pred)
```

Out[42]:

```
161383675.95784867
```

In [43]:

```
r2_score(y_train, y_train_pred)
```

Out[43]:

```
0.7110100333340962
```

In [44]:

```
r2_score(y_test, y_test_pred)
```

Out[44]:

```
0.49869112364165125
```

In [ ]: