Micro Project

Tittle:Movie ticket pricing

Yogehswari.G--225229148

In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [3]:
```
df=pd.read_csv("cinemaTicket.csv")
df.head()
```

Out[3]:

	film_code	cinema_code	total_sales	tickets_sold	tickets_out	show_time	occu_perc	ticket_p
0	1492	304	3900000	26	0	4	4.26	1500
1	1492	352	3360000	42	0	5	8.08	800
2	1492	489	2560000	32	0	4	20.00	800
3	1492	429	1200000	12	0	1	11.01	1000
4	1492	524	1200000	15	0	3	16.67	800

In [4]:
```
df.shape
```

Out[4]:
```
(142524, 14)
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 142524 entries, 0 to 142523
Data columns (total 14 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   film_code     142524 non-null  int64  
 1   cinema_code   142524 non-null  int64  
 2   total_sales   142524 non-null  int64  
 3   tickets_sold  142524 non-null  int64  
 4   tickets_out   142524 non-null  int64  
 5   show_time     142524 non-null  int64  
 6   occu_perc     142399 non-null  float64
 7   ticket_price  142524 non-null  float64
 8   ticket_use    142524 non-null  int64  
 9   capacity      142399 non-null  float64
 10  date          142524 non-null  object 
 11  month         142524 non-null  int64  
 12  quarter       142524 non-null  int64  
 13  day           142524 non-null  int64  
dtypes: float64(3), int64(10), object(1)
memory usage: 15.2+ MB
```

Filling null values with mean

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

Out[6]:
```
film_code         0
cinema_code       0
total_sales       0
tickets_sold      0
tickets_out       0
show_time         0
occu_perc       125
ticket_price      0
ticket_use        0
capacity        125
date              0
month             0
quarter           0
day               0
dtype: int64
```

In [7]:
```
df['occu_perc'] = df['occu_perc'].fillna((df['occu_perc'].mean()))
df['capacity'] = df['capacity'].fillna((df['capacity'].mean()))
df.isnull().sum()
```

Out[7]:
```
film_code       0
cinema_code     0
total_sales     0
tickets_sold    0
tickets_out     0
show_time       0
occu_perc       0
ticket_price    0
ticket_use      0
capacity        0
date            0
month           0
quarter         0
day             0
dtype: int64
```

In [8]:
```
df.describe()
```

Out[8]:

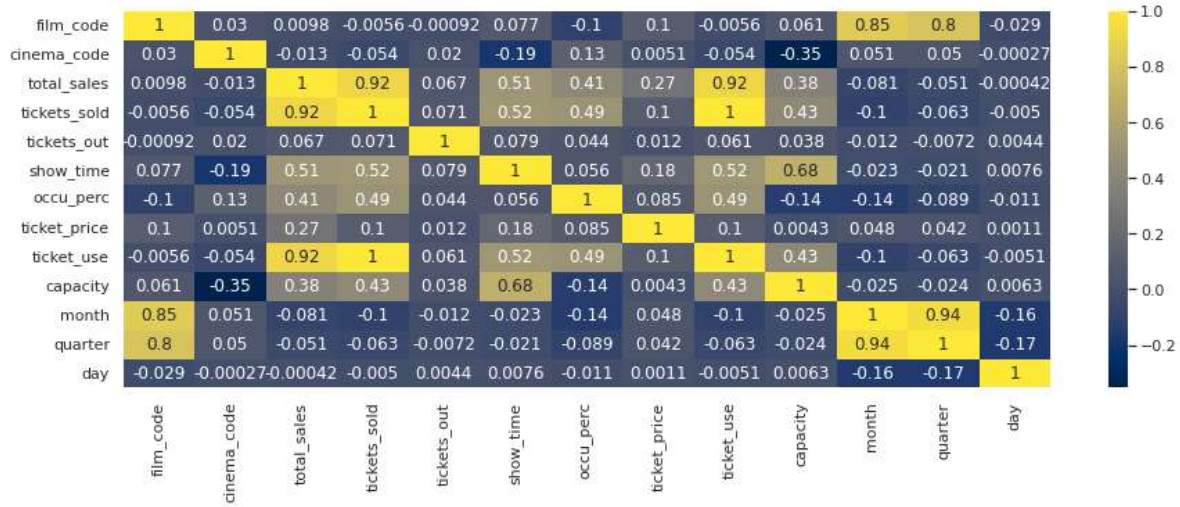
	film_code	cinema_code	total_sales	tickets_sold	tickets_out	show_tim
count	142524.000000	142524.000000	1.425240e+05	142524.000000	142524.000000	142524.00000
mean	1518.985111	320.378427	1.234728e+07	140.137570	0.237413	3.93210
std	36.184450	159.701229	3.065486e+07	279.758733	2.923206	3.05627
min	1471.000000	32.000000	2.000000e+04	1.000000	0.000000	1.00000
25%	1485.000000	181.000000	1.260000e+06	18.000000	0.000000	2.00000
50%	1498.000000	324.000000	3.720000e+06	50.000000	0.000000	3.00000
75%	1556.000000	474.000000	1.110000e+07	143.000000	0.000000	5.00000
max	1589.000000	637.000000	1.262820e+09	8499.000000	311.000000	60.00000

Correlation plot of the dataset

Some of the features show strong correlation with the target such as ticket sold and total sales which showed strong correlation in addition show-time and total sales show good correlaticy also capacity of the cinema and the total sales show correlation.

In [9]:
```
sns.set_theme(color_codes=True)
sns.set_theme(style="darkgrid")
plt.figure(figsize = (15,5))
sns.heatmap(df.corr()  ,annot = True , cmap="cividis")
```

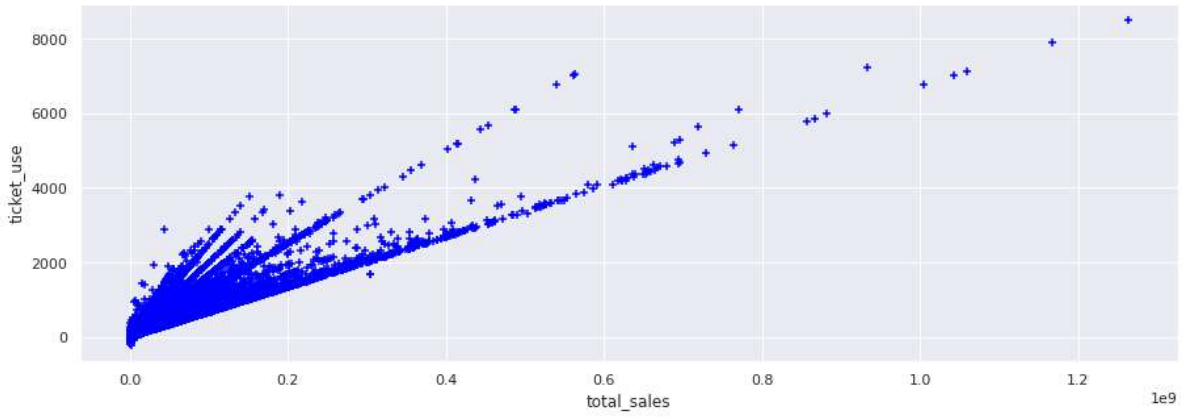
Out[9]: <AxesSubplot:>



Investigating features that have high correlation with the target

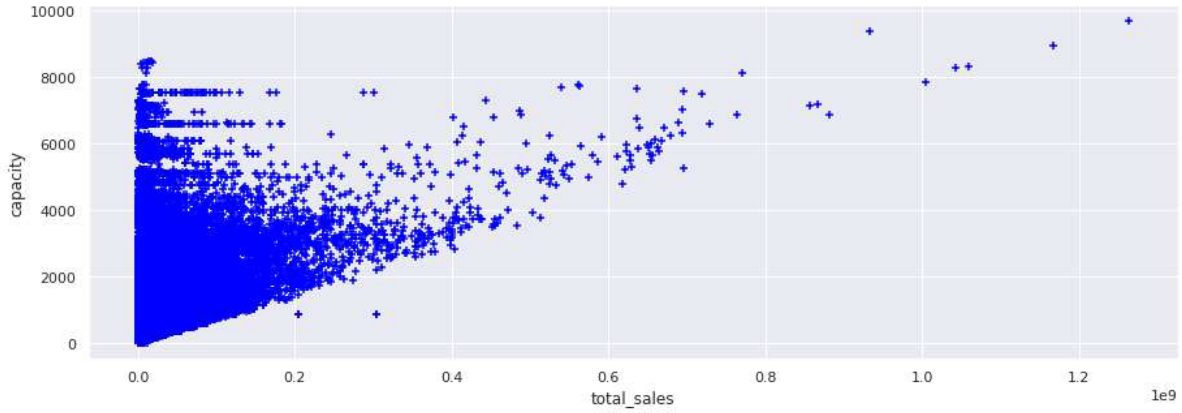
In [10]:
```
%matplotlib inline
sns.set_theme(color_codes=True)
sns.set_theme(style="darkgrid")
plt.figure(figsize = (15,5))
plt.xlabel('total_sales')
plt.ylabel('ticket_use')
plt.scatter(df.total_sales,df.ticket_use,color='blue',marker='+')
```

Out[10]: <matplotlib.collections.PathCollection at 0x7f13db0f7fd0>



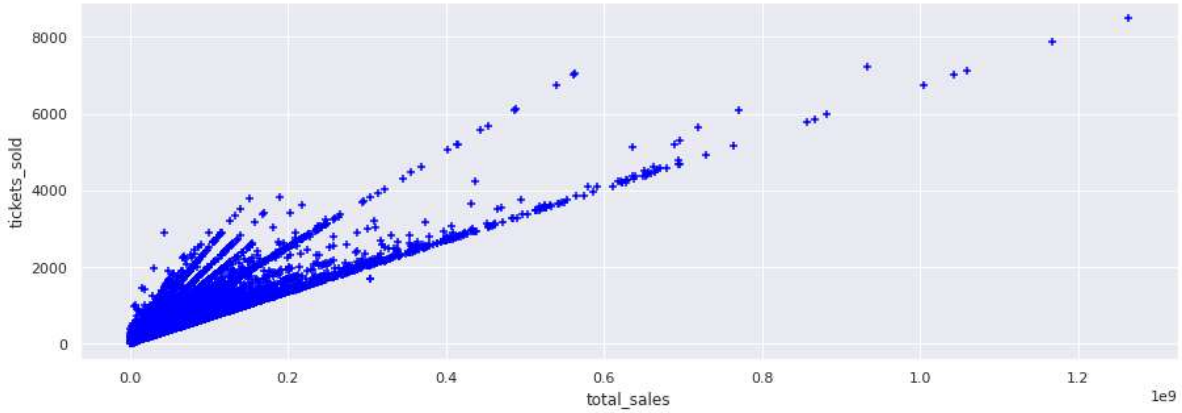
In [11]:
```
%matplotlib inline
sns.set_theme(color_codes=True)
sns.set_theme(style="darkgrid")
plt.figure(figsize = (15,5))
plt.xlabel('total_sales')
plt.ylabel('capacity')
plt.scatter(df.total_sales,df.capacity,color='blue',marker='+')
```

Out[11]: <matplotlib.collections.PathCollection at 0x7f13da886750>



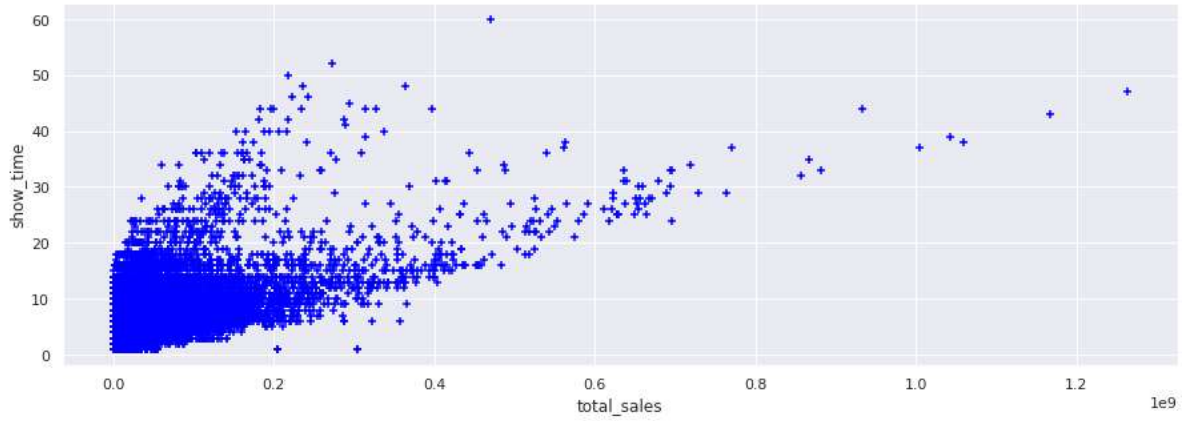
In [12]:
```
%matplotlib inline
sns.set_theme(color_codes=True)
sns.set_theme(style="darkgrid")
plt.figure(figsize = (15,5))
plt.xlabel('total_sales')
plt.ylabel('tickets_sold')
plt.scatter(df.total_sales,df.tickets_sold,color='blue',marker='+')
```

Out[12]: <matplotlib.collections.PathCollection at 0x7f13da804990>



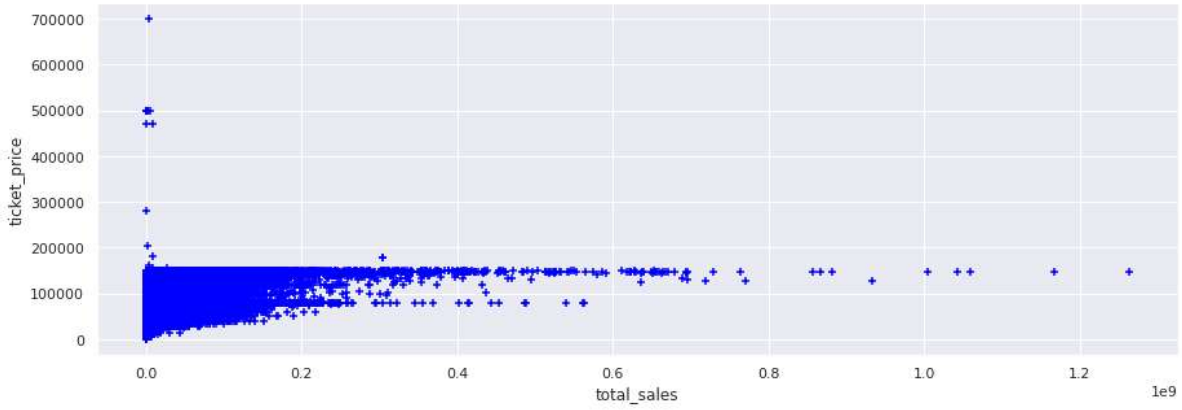
In [13]:
```
%matplotlib inline
sns.set_theme(color_codes=True)
sns.set_theme(style="darkgrid")
plt.figure(figsize = (15,5))
plt.xlabel('total_sales')
plt.ylabel('show_time')
plt.scatter(df.total_sales,df.show_time,color='blue',marker='+')
```

Out[13]: <matplotlib.collections.PathCollection at 0x7f13da785550>



In [14]:
```
%matplotlib inline
sns.set_theme(color_codes=True)
sns.set_theme(style="darkgrid")
plt.figure(figsize = (15,5))
plt.xlabel('total_sales')
plt.ylabel('ticket_price')
plt.scatter(df.total_sales,df.ticket_price,color='blue',marker='+')
```

Out[14]: <matplotlib.collections.PathCollection at 0x7f13da704c90>



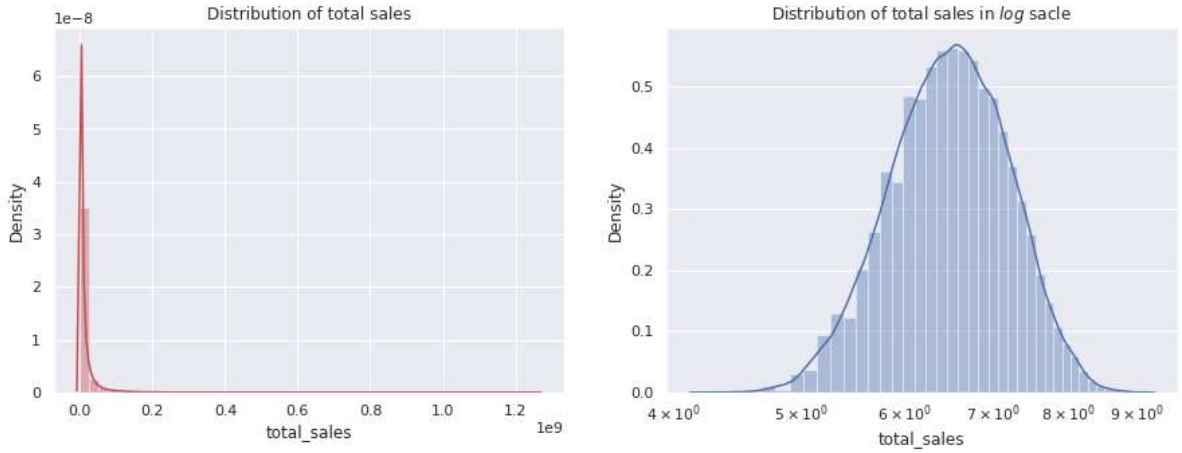
Target normalization

In [15]:
```
f= plt.figure(figsize=(15,5))

ax=f.add_subplot(121)
sns.distplot(df['total_sales'],bins=50,color='r',ax=ax)
ax.set_title('Distribution of total sales')

ax=f.add_subplot(122)
sns.distplot(np.log10(df['total_sales']),bins=40,color='b',ax=ax)
ax.set_title('Distribution of total sales in $log$ sacle')
ax.set_xscale('log');
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histog
rams).
  warnings.warn(msg, FutureWarning)
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histog
rams).
  warnings.warn(msg, FutureWarning)
```



Spliting the data into training and testing

In [16]:
```
from sklearn.model_selection import train_test_split
X = df[['tickets_sold','occu_perc',
             'capacity', 'show_time','ticket_price','ticket_use']]
y = np.log10(df['total_sales'])
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.30)
```

Data normalization

In [17]:
```
from sklearn.preprocessing import MinMaxScaler
scaling = MinMaxScaler(feature_range=(-1,1)).fit(X_train)
x_train = scaling.transform(X_train)
x_test = scaling.transform(X_test)
```

Linear regression

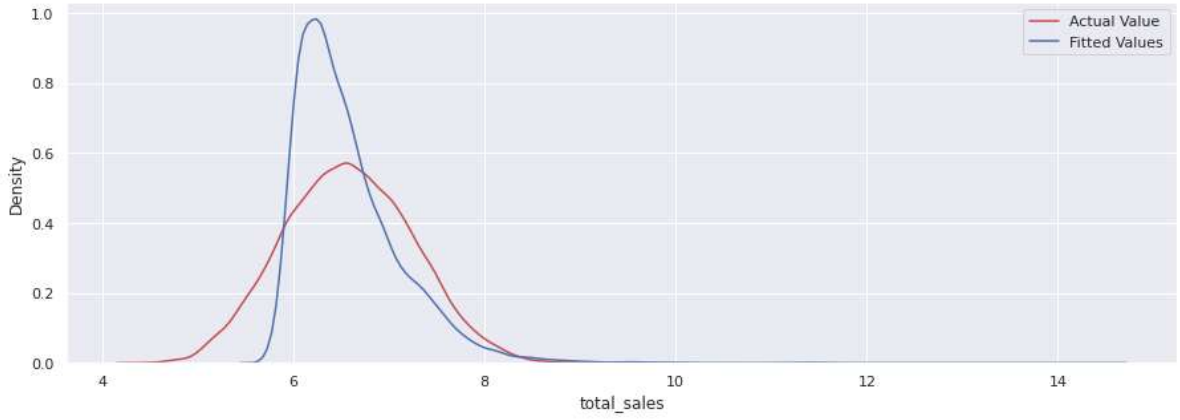
In [18]:
```
from sklearn import linear_model
# Create linear regression object
reg = linear_model.LinearRegression()
reg.fit(x_train,y_train)
```

Out[18]:
```
LinearRegression()
```

In [19]:
```
y_pred = reg.predict(x_test)
```

In [20]:
```
sns.set_theme(color_codes=True)
sns.set_theme(style="darkgrid")
plt.figure(figsize = (15,5))
ax1 = sns.distplot(y_test, hist=False, color='r', label='Actual Value')
sns.distplot(y_pred, hist=False, color='b', label='Fitted Values', ax=ax1)
ax1.ticklabel_format(style='plain')
plt.legend(loc='best')
plt.show()
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).
  warnings.warn(msg, FutureWarning)
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).
  warnings.warn(msg, FutureWarning)
```



Model evaluation

In [21]:
```
import sklearn.metrics as metrics
mae = metrics.mean_absolute_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse) # or mse**(0.5)
r2 = metrics.r2_score(y_test,y_pred)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
Results of sklearn.metrics:
MAE: 0.29685893717309403
MSE: 0.1406712694905156
RMSE: 0.3750616875802107
R-Squared: 0.6876061921102878
```

the predicted values and the actual values are qiute close

In [22]:
```
df_actual_pre ={'Actual': y_test, 'Predicted': y_pred}
df1_actual_pre= pd.DataFrame(df_actual_pre)
df1_actual_pre.head(10)
```

Out[22]:

	Actual	Predicted
138130	5.748188	6.062517
4669	6.133539	6.100696
125828	6.666518	6.399685
48176	6.494155	6.374911
21179	6.292256	6.238935
131891	6.577492	6.235690
22956	5.845098	5.984222
139850	7.445604	6.953395
12246	7.256718	6.698770
25124	6.722634	6.541335

Non linear regression (knn)

In [23]:
```
from sklearn import neighbors
n_neighbors = 3

for i, weights in enumerate(["uniform", "distance"]):
    knn = neighbors.KNeighborsRegressor(n_neighbors, weights=weights, leaf_siz
    knn.fit(x_train, y_train)
    y_pred2 = knn.predict(x_test)
```

In [24]:
```
knn
```

Out[24]:
```
KNeighborsRegressor(leaf_size=3, n_neighbors=3, weights='distance')
```

model evaluation

In [25]:
```
import sklearn.metrics as metrics
import numpy as np
mae = metrics.mean_absolute_error(y_test, y_pred2)
mse = metrics.mean_squared_error(y_test, y_pred2)
rmse = np.sqrt(mse) # or mse**(0.5)
r2 = metrics.r2_score(y_test,y_pred2)
print("Results of sklearn.metrics:")
print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
Results of sklearn.metrics:
MAE: 0.021981452527660237
MSE: 0.002870756983343082
RMSE: 0.05357944553038116
R-Squared: 0.9936248054858636
```

In [26]:
```
df_actual_pre2 ={'Actual': y_test, 'Predicted': y_pred2}
df1_actual_pre2= pd.DataFrame(df_actual_pre2)
df1_actual_pre2.head(10)
```

Out[26]:

	Actual	Predicted
138130	5.748188	5.748188
4669	6.133539	6.134966
125828	6.666518	6.660067
48176	6.494155	6.441416
21179	6.292256	6.373247
131891	6.577492	6.606941
22956	5.845098	6.018286
139850	7.445604	7.435003
12246	7.256718	7.275293
25124	6.722634	6.721689

Yogeshwari.G--225229148