In [4]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from statsmodels.api import add_constant,OLS
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler,MinMaxScaler,RobustScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.stats.diagnostic import het_breuschpagan
from statsmodels.graphics.gofplots import qqplot
from sklearn.linear_model import LinearRegression,Ridge,Lasso,ElasticNet,SGDRegressor
from mlxtend.feature_selection import SequentialFeatureSelector as sfs
from sklearn.feature_selection import SequentialFeatureSelector as sfs,RFE
from sklearn.metrics import r2_score, mean_absolute_error,mean_squared_error,mean_absc
from sklearn.model_selection import train_test_split, GridSearchCV,RandomizedSearchCV
import warnings
warnings.filterwarnings('ignore')
from statsmodels.stats.diagnostic import linear_rainbow,het_breuschpagan
import statsmodels.api as sma
```

In [5]:

```
! pip install mlxtend
```

Case Study

Data Exploration

```
In [ ]: df=pd.read_csv(r"C:\Users\M.JOTHI\OneDrive\Desktop\bridge course\ML\week 2\casestudy\B
        df.head()
```

```
In [ ]: print('No. of rows:',df.shape[0])
        print('No. of columns:',df.shape[1])
```

```
In [ ]: df.dtypes
```

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

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

```
In [ ]: df.describe(include='object').T
```

```
In [ ]: # Lets treat the total_sqft first
        df['total_sqft'].unique()
```

```
In [ ]: pd.set_option('display.max_rows',None)
        df['total_sqft'].value_counts()
```

```
In [ ]: df['total_sqft']=df['total_sqft'].str.split(expand=True).loc[:,0]
```

```
In [ ]: df['total_sqft'].value_counts()
```

```
In [ ]: df['total_sqft'].str.isdigit().sum() # for numeric
```

```
In [ ]: df=df[df['total_sqft'].str.isnumeric()] # for numeric
```

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

In [ ]:
```
df['total_sqft']=df['total_sqft'].astype(float)
```

In [ ]:
```
# lets treat the size column as well
df['size'].value_counts()
```

In [ ]:
```
df['size']=df['size'].str.split(expand=True).iloc[:,0]
```

In [ ]:
```
df['size']=df['size'].astype(float)
```

In [ ]:
```
df.dtypes
```

In [ ]:
```
# Null values
df['society'].isnull().sum()/df.shape[0]
```

- There is 41 % null values are there so we drop the null values.

In [ ]:
```
df.drop(columns='society',inplace=True)
# del df['society']
```

In [ ]:
```
df.head()
```

In [ ]:
```
# lets now treat the column availability
df['availability'].unique()
```

In [ ]:
```
def availability(s):
    if s in ['Ready To Move','Immediate Possession']:
        return 'Ready To Move'
    else:
        return 'Under Construction'
```

In [ ]:
```
df['availability']=df['availability'].apply(availability)
```

In [ ]:
```
df['availability'].value_counts()
```

In [ ]:
```
df.head()
```

In [ ]:
```
# lets check the area type as well
df['area_type'].value_counts()
```

Univarite Analysis

In [ ]:
```
df.columns
```

In [ ]:
```
num_col=['size', 'total_sqft', 'bath',
       'balcony', 'price']
cat_col=['area_type','availability','location']
```

In [ ]:

```
t=1
plt.figure(figsize=(10,10))
for i in num_col:
    plt.subplot(3,3,t)
    sns.distplot(x=df[i])
    t+=1
plt.tight_layout()
plt.show()
```

In [ ]:

```
for i in df.select_dtypes(include='number'):
    sns.boxplot(x=df[i])
    plt.show()
```

In [ ]:

```
t=1
plt.figure(figsize=(10,10))
for i in cat_col:
    plt.subplot(2,2,t)
    sns.countplot(x=df[i])
    plt.xticks(rotation=90)
    t+=1
plt.tight_layout()
plt.show()
```

In [ ]:

```
df['location']=df['location'].str.lstrip(' ')
df.groupby('location')['price'].mean().sort_values(ascending=False
                                                  )
```

In [ ]:

```
top_10_location= df.groupby('location')['price'].mean().sort_values(ascending=False).h
```

In [ ]:

```
top_10_location.plot(kind='bar')
plt.title('Top_10_location as per property Price')
plt.show()
```

Bivariate Analysis

In [ ]:

```
t=1
plt.figure(figsize=(10,10))
for i in num_col:
    if i!='price':
        plt.subplot(2,2,t)
        sns.scatterplot(x=df[i],y=df['price'])
        t+=1
plt.tight_layout()
plt.show()
```

In [ ]:

```
sns.heatmap(df.select_dtypes(include=np.number).corr(),vmax=1,vmin=-1,annot=True,cmap=
plt.show()
```

In [ ]:

```
t=1
plt.figure(figsize=(10,5))
for i in cat_col:
    if i!='location':
        plt.subplot(1,2,t)
        sns.boxplot(x=df[i],y=df['price'])
        t+=1
plt.tight_layout()
plt.show()
```

Data Preprocessing

In [ ]:
```
((df.isnull().sum()/df.shape[0])*100).sort_values(ascending=False)
```

In [ ]:
```
df[df['location'].isnull()]
```

In [ ]:
```
df.drop(index=[568],inplace=True)
```

In [ ]:
```
df[df['size'].isnull()]
```

In [ ]:
```
# lets impute this with mode
df['size'].mode()
```

In [ ]:
```
df['size'].fillna(df['size'].mode()[0],inplace=True)
```

In [ ]:
```
df[df['bath'].isnull()]
```

In [ ]:
```
df.groupby('size')['bath'].median()
```

In [ ]:
```
df['size'][df['bath'].isnull()].unique()
#So only for size of 1 to 5 are null so we are replacing
```

In [ ]:
```
df['bath']=np.where(((df['bath'].isnull())&(df['size']==1)),1,df['bath'])
df['bath']=np.where(((df['bath'].isnull())&(df['size']==2)),2,df['bath'])
df['bath']=np.where(((df['bath'].isnull())&(df['size']==3)),3,df['bath'])
df['bath']=np.where(((df['bath'].isnull())&(df['size']==4)),4,df['bath'])
df['bath']=np.where(((df['bath'].isnull())&(df['size']==5)),5,df['bath'])
```

In [ ]:
```
df[df['balcony'].isnull()]
```

In [ ]:
```
df.groupby('size')['balcony'].median()
```

In [ ]:
```
df['balcony'].fillna(df['balcony'].median(),inplace=True)
```

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

In [ ]:
```
# Treat outliers
```

In [ ]:
```
t=1
plt.figure(figsize=(10,10))
for i in num_col:
    plt.subplot(3,2,t)
    sns.boxplot(x=df[i])
    t+=1
```

```
plt.tight_layout()
plt.show()
```

In [ ]:

```
# lets drop the  extreme values (values > 95 percentile)
outlier_col = ['size','total_sqft','bath','price']
```

In [ ]:

```
per_99 = df[outlier_col].quantile(0.99)
```

In [ ]:

```
per_99
```

In [ ]:

```
df=df[~(df[outlier_col]>per_99).any(axis=1)]
```

In [ ]:

```
df.shape
```

In [ ]:

```

```

In [ ]:

```
t=1
plt.figure(figsize=(10,10))
for i in num_col:
    plt.subplot(3,2,t)
    sns.boxplot(x=df[i])
    t+=1
plt.tight_layout()
plt.show()
```

Encoding

In [ ]:

```
df['availability'] =np.where (df['availability']== 'Ready To Move',1,0)
```

In [ ]:

```
df.sample(20)
```

In [ ]:

```
(df['location'].value_counts(normalize=True)*100).cumsum()
```

In [ ]:

```
top_10_properties=df.groupby('location')['price'].mean().sort_values(ascending=False).
```

In [ ]:

```
top_10_properties.index
```

In [ ]:

```
def location(s):
    if s in top_10_properties.index:
        return 1
    else:
        return 0
```

In [ ]:

```
df['location']=df['location'].apply(location)
```

In [ ]:

```
df.head(5)
```

In [ ]:

```
df=pd.get_dummies(df,drop_first=True)
```

In [ ]:

```
df
```

predictive Modelling

In [ ]:

```
def model_validation(xtrain,ytrain,xtest,ytest,model):
    global m
    m=model
    m.fit(xtrain,ytrain)
    print('Training Scores')
    pred=m.predict(xtrain)
    print('R2:',r2_score(ytrain,pred))
    print('MSE:',mean_squared_error(ytrain,pred))
    print('RMSE:',mean_squared_error(ytrain,pred)**0.5)
    print('MAPE:',mean_absolute_percentage_error(ytrain,pred))


    print('Testing Scores')
    pred=m.predict(xtest)
    print('R2:',r2_score(ytest,pred))
    print('MSE:',mean_squared_error(ytest,pred))
    print('RMSE:',mean_squared_error(ytest,pred)**0.5)
    print('MAPE:',mean_absolute_percentage_error(ytest,pred))
```

Train test split

In [ ]:

```
x=df.drop(columns=['price'],axis=1)
y=df['price']

x_train,x_test,y_train,y_test =train_test_split(x,y,train_size=0.8,random_state=1)
```

In [ ]:

```

```

Linear Regression

In [ ]:

```
model_validation(x_train,y_train,x_test,y_test,LinearRegression())
```

In [ ]:

```
# coefficients of linear regression
pd.DataFrame({'Features':x.columns,'Coef':m.coef_})
```

Lasso

In [ ]:

```
# Lasso
model_validation(x_train,y_train,x_test,y_test,Lasso(alpha=0.005))
```

In [ ]:

```
# coefficients of Lasso
pd.DataFrame({'Features':x.columns,'Coef':m.coef_})
```

Ridge

In [ ]:

```
model_validation(x_train,y_train,x_test,y_test,Ridge(alpha=0.5))# Ridge
```

In [ ]:

```
# coefficients of Ridge
pd.DataFrame({'Features':x.columns,'Coef':m.coef_})
```

Elastic Net

In [ ]:

```
# Elastic Net
model_validation(x_train,y_train,x_test,y_test,ElasticNet(alpha=0.01,l1_ratio=0.1))
```

In [ ]:

```
# coefficients of Elastic Net
pd.DataFrame({'Features':x.columns,'Coef':m.coef_})
```

Grid Search Cv

In [ ]:

```
## Using Grid Search CV
model_validation(x_train,y_train,x_test,y_test,ElasticNet(alpha=0.001,l1_ratio=0.25))
```

Feature Selection

In [ ]:

```
from mlxtend.feature_selection import SequentialFeatureSelector
```

In [ ]:

```
sfs = SequentialFeatureSelector(estimator=LinearRegression(),
                                k_features='best',
                                scoring='r2')
```

In [ ]:

```
sfs.fit(x_train,y_train)
```

In [ ]:

```
k = list(sfs.k_feature_names_)
```

In [ ]:

```
sfs.k_score_
```

In [ ]:

```
model_validation(x_train[k],y_train,x_test[k],y_test,LinearRegression())
```

In [ ]:

```

```

Improve the prediction of the model

In [ ]:

```
#different aooroach in Encoding
#different techinque of missing values Traetment
#Use some advance models for prediction
```

In [ ]:

```
# Lets take the Linear Regression Model as final model
```

In [ ]:

```
model_validation(x_train,y_train,x_test,y_test,LinearRegression())
```

In [ ]:

```
m.feature_names_in_
```

In [ ]:

```
m.predict([[1,0,3,1500,2,2,1,0,0]])
```

In [ ]:

```
df.head()
```

Save this model as trained model

In [ ]:

```
import pickle
```

In [ ]:

```
with open('model_lr.pkl','wb') as file:
    pickle.dump(m,file)

print('The file has been saved')
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: