Objective:

To study the real estate data in order to understand the impact of input variables Price and predict the price of a house based on given inputs.

Here: Price is the target variable (ie.y)

Steps to be followed in a Linear Regression Project

1) Import required libraries (i.e. pandas, matplotlib, seaborn, sklearn) 2) Import data 3) Check shape, columns, dtypes, head, tail, describe, etc 4) Check if any of the numeric variable is stored as object (string). If yes then change the data type 5) Check if there are missing values (NaN) in the data 6) Perform missing value treatment, if required 7) Check outliers 8) Treat outliers (if exists) 9) Perform EDA (Exploratory Data Analysis) to check data distribution, data mix, and correlation 10) Check spelling differences, errors etc in object variables and clean them 11) Dummy conversion: transform object variables to numeric 12) Create final data by combining all numeric variables and dummies 13) Create X (with all independent variables) and Y (with the target variable) 14) Random sampling: create training and test samples 15) Instantiate LinearRegression class 16) Bulid training model 17) check accuracy of training model 18) Test model: Predict y using test sample 19) check accuracy of Test model 20) Validate the project output 21) Deploy

In [12]:
```
#data preparation and analysis library
import pandas as pd

#ploting libraries
import matplotlib.pyplot as plt
import seaborn as sns

#library for creating random samples
from sklearn.model_selection import train_test_split

#library for building linear regression model
from sklearn.linear_model import LinearRegression

#feature selection (to selet significant variable)
from sklearn.feature_selection import SelectKBest, f_regression
```

In [13]:
```
#load data
df=pd.read_csv(r"C:\Users\GIRISH\Desktop\INTROTALLENT\PYTHON\ML PROJECT\104380_Python an
```

In [14]:
```
df.head(2)
```

Out[14]:

	Home	Price	SqFt	Bedrooms	Bathrooms	Offers	Brick	Neighborhood
0	1	114300	1790.0	2	2	2	No	East
1	2	114200	2030.0	4	2	3	No	East

In [15]:
```
#in this data "home" column contains serial numbers
#so we can drp home  column  as it is of no use

df=df.drop("Home",axis=1)
```

In [16]:
```
df.head()
```

Out[16]:

	Price	SqFt	Bedrooms	Bathrooms	Offers	Brick	Neighborhood
0	114300	1790.0	2	2	2	No	East
1	114200	2030.0	4	2	3	No	East
2	114800	1740.0	3	2	1	No	East
3	94700	1980.0	3	2	3	No	East
4	119800	2130.0	3	3	3	No	East

In [17]:
```
#print row  and column count
df.shape
```

Out[17]:
```
(128, 7)
```

In [18]:
```
#check data types of variables
df.dtypes
```

Out[18]:
```
Price             int64
SqFt            float64
Bedrooms          int64
Bathrooms         int64
Offers            int64
Brick            object
Neighborhood     object
dtype: object
```

In [19]:
```
#Bedrooms, Bthrooms, and offers are categorical variables stored as int.
#change the data type of these variables to object
df["Bedrooms"]=df["Bedrooms"].astype("object")
df["Bathrooms"]=df["Bathrooms"].astype("object")
df["Offers"]=df["Offers"].astype("object")
```

In [20]:
```
df.dtypes
```

Out[20]:
```
Price             int64
SqFt            float64
Bedrooms         object
Bathrooms        object
Offers           object
Brick            object
Neighborhood     object
dtype: object
```

In [21]:
```
#Feature Engeneering -[chcek and input missing values, if any]
df.isnull().sum()
```

Out[21]:
```
Price           0
SqFt            1
Bedrooms        0
Bathrooms       0
Offers          0
Brick           0
Neighborhood    1
dtype: int64
```

In [22]:
```
#impute SqFt with meadian
df["SqFt"]=df["SqFt"].fillna(df["SqFt"].median())
```

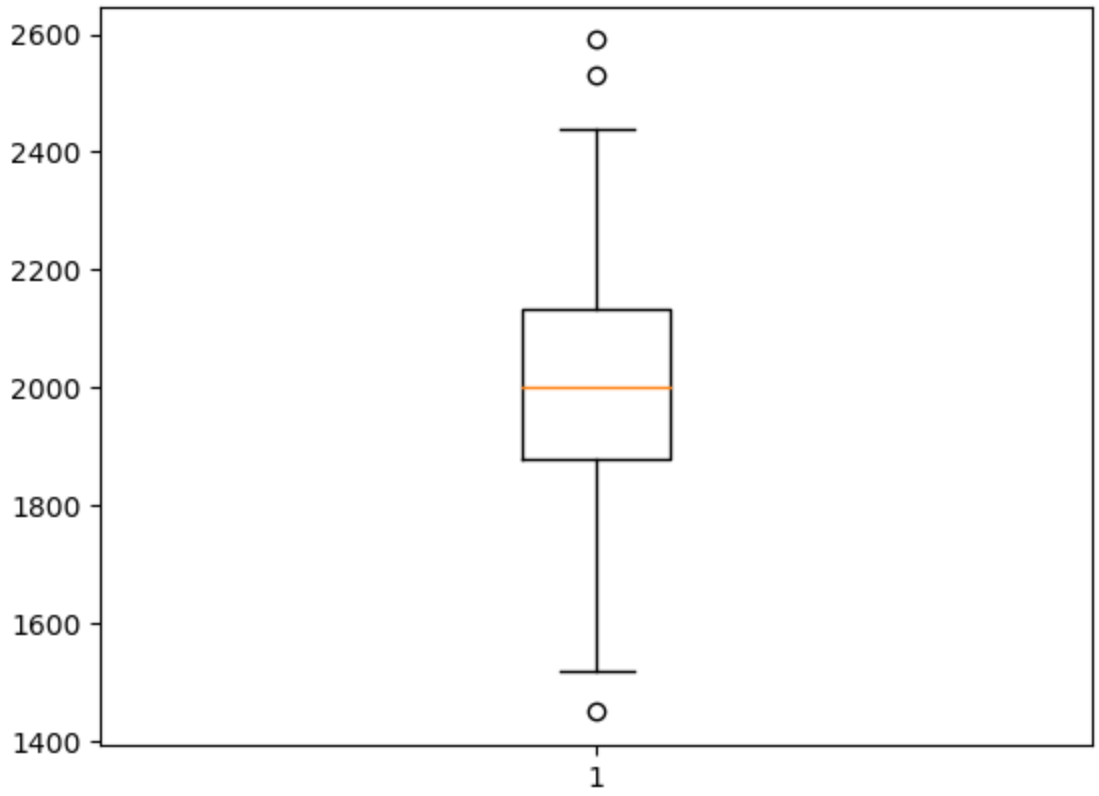
In [23]:
```
#drop nan rows
df=df.dropna(axis=0)
```

In [24]:
```
df.isnull().sum()
```

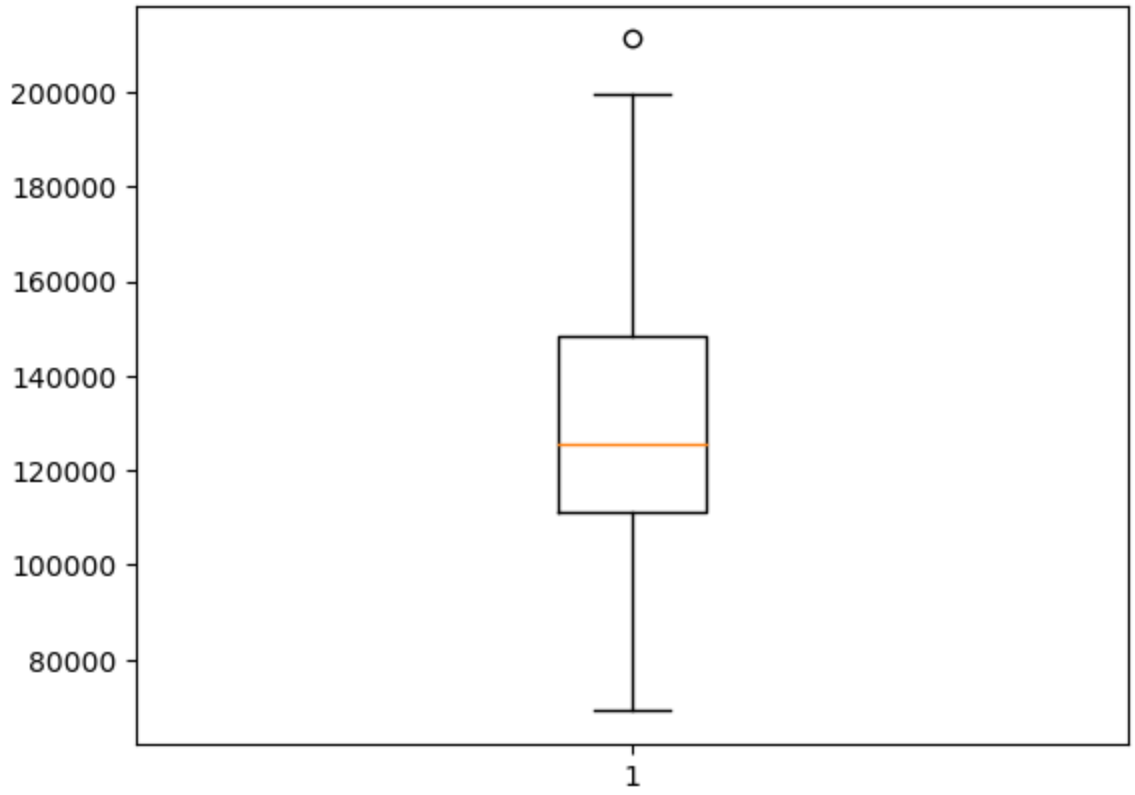
Out[24]:
```
Price           0
SqFt            0
Bedrooms        0
Bathrooms       0
Offers          0
Brick           0
Neighborhood    0
dtype: int64
```

In [25]:
```
#Feature Engineering - Outlier treatment
```

In [26]:
```
#outlier test in SqFt variable
plt.boxplot(df["SqFt"]) #has outliers
plt.show()
```

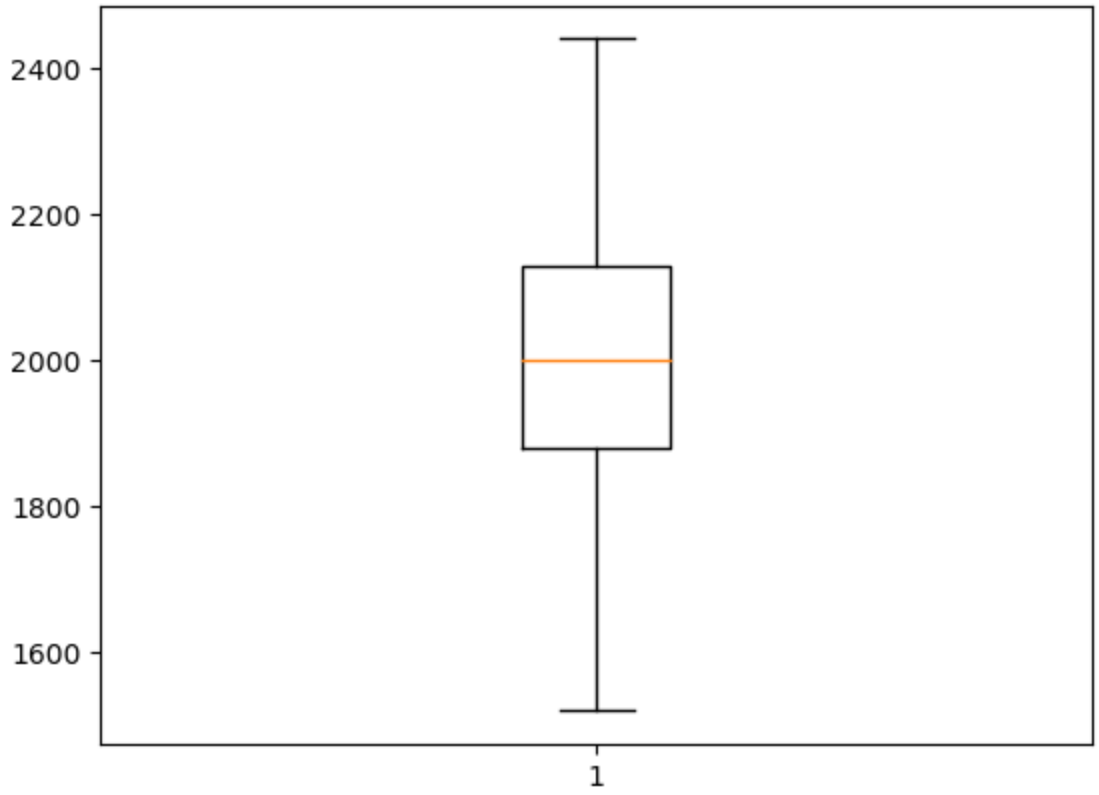


In [27]:
```
#outlier test in price variable
plt.boxplot(df["Price"]) #has outliers
plt.show()
```

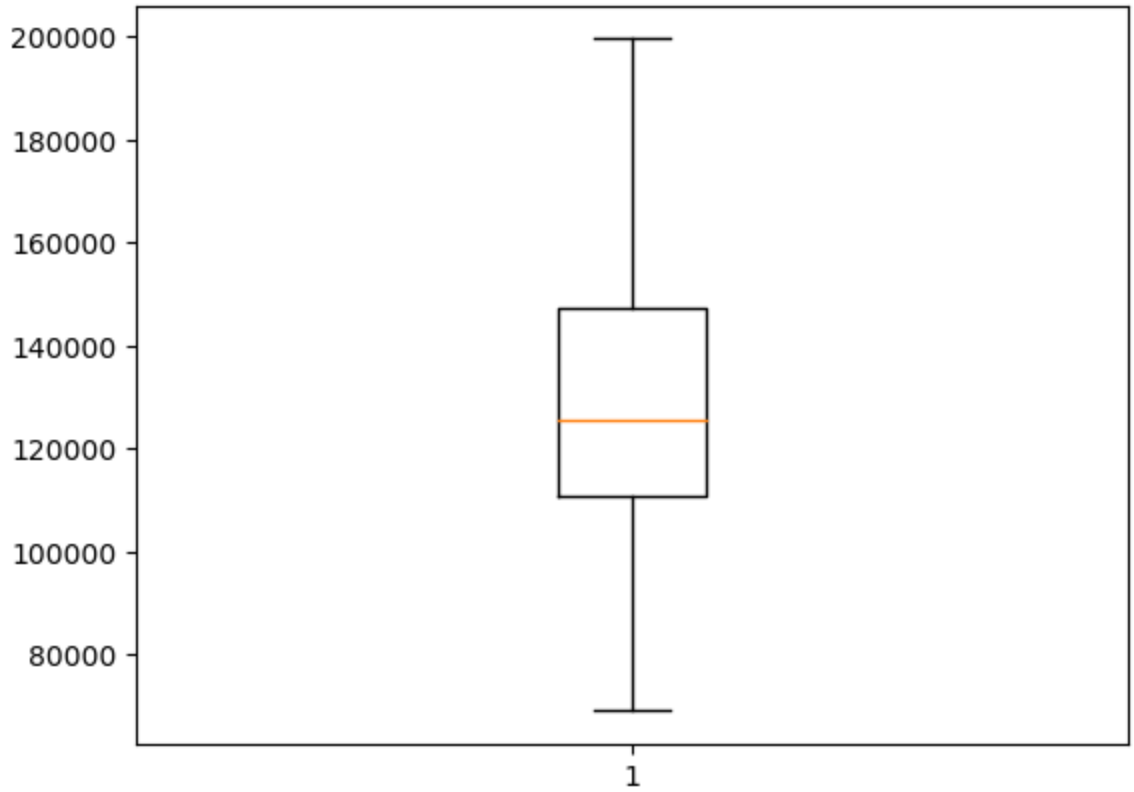


In [28]:
```
#User defined function to remove outliers
def remove_outlier(d,c):
    #find q1 and q3
    q1 = d[c].quantile(0.25)
    q3 = d[c].quantile(0.75)

    #IQR
    iqr = q3-q1

    #upper and lower bond
    ub = q3+1.5*iqr
    lb = q1-1.5*iqr

    #remove outliers and store good data in result
    result=d[(d[c]>=lb) & (d[c]<=ub)]
    return result
```

In [29]:
```
#remove outliers from SqFt variable

df=remove_outlier(df,'SqFt')
plt.boxplot(df['SqFt'])
plt.show()
```



In [30]:
```
#remove outliers from price variable

df=remove_outlier(df,'Price')
plt.boxplot(df['Price'])
plt.show()
```



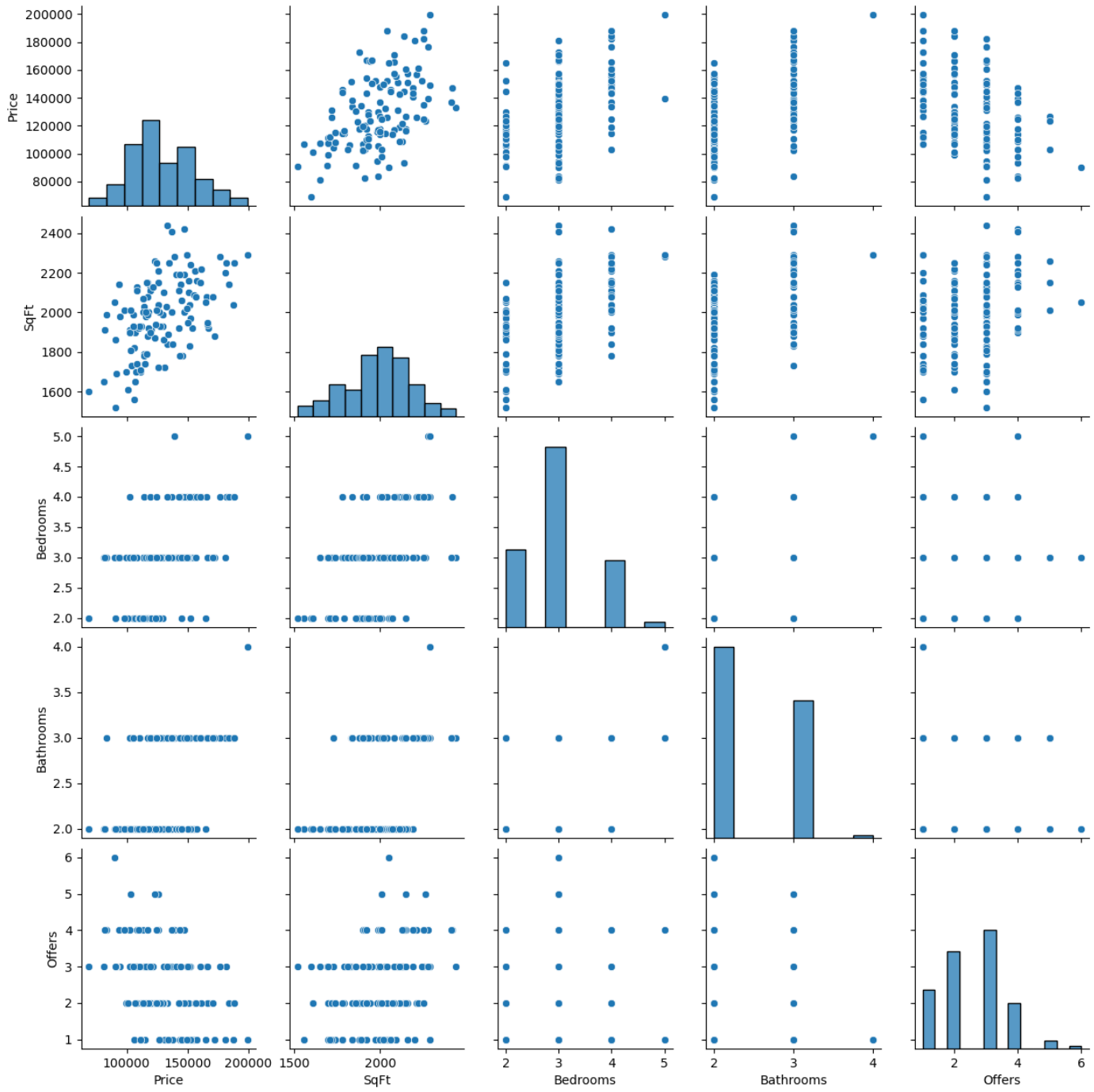
EDA(Exploratory Data Analysis)

In [31]:
```
#Create paitplot
```

```
sns.pairplot(df)
```

Out[31]:

```
<seaborn.axisgrid.PairGrid at 0x2610db1ead0>
```

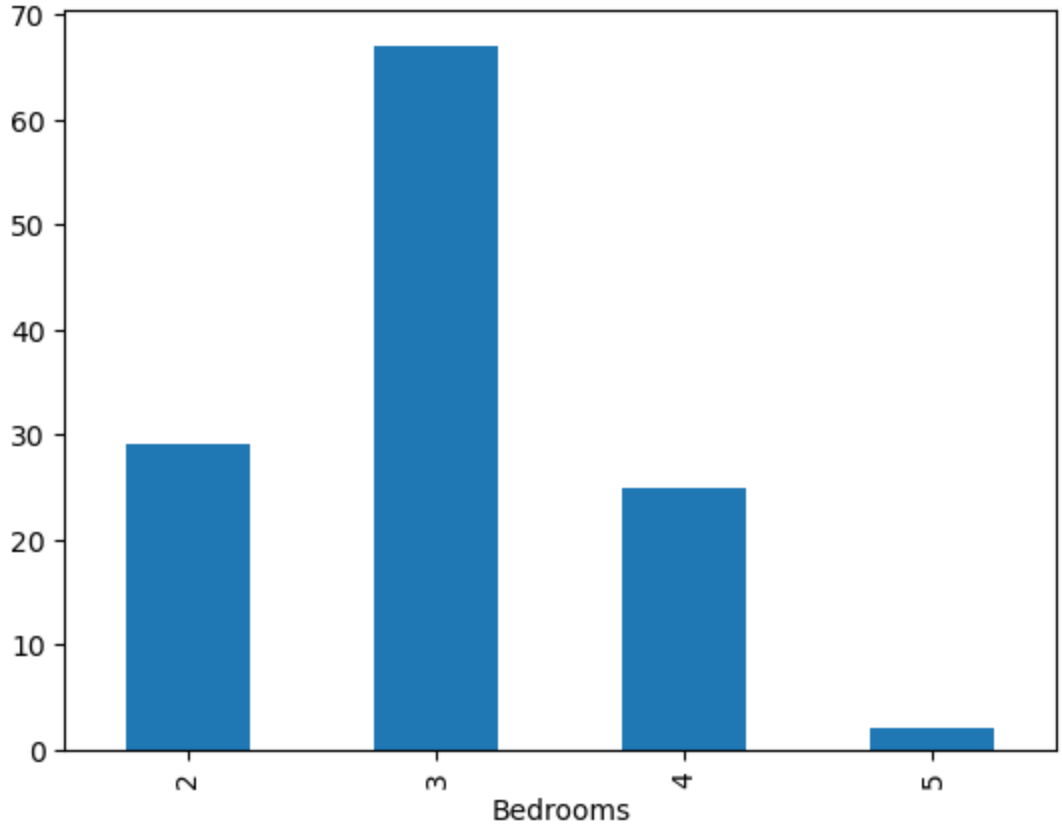


In [32]:

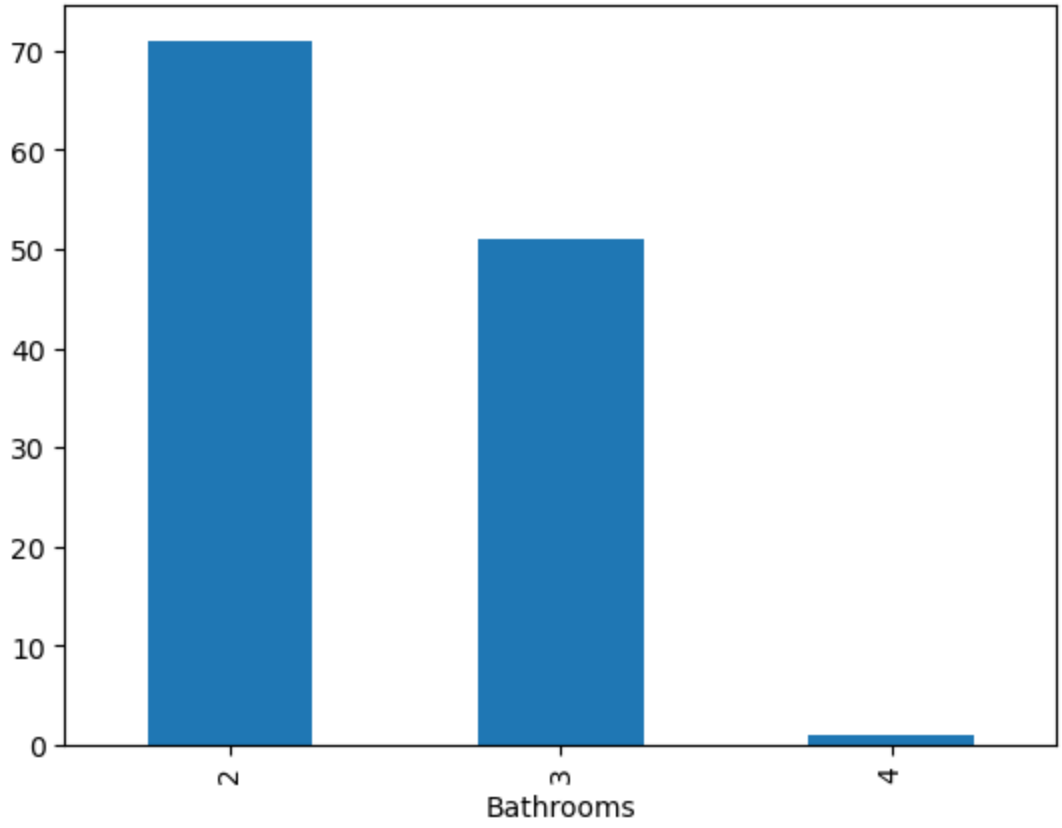
```
#data mix
#"Bedrooms", "Bathrooms", "offers", "Brick", "Neighbrhood"
```

In [33]:

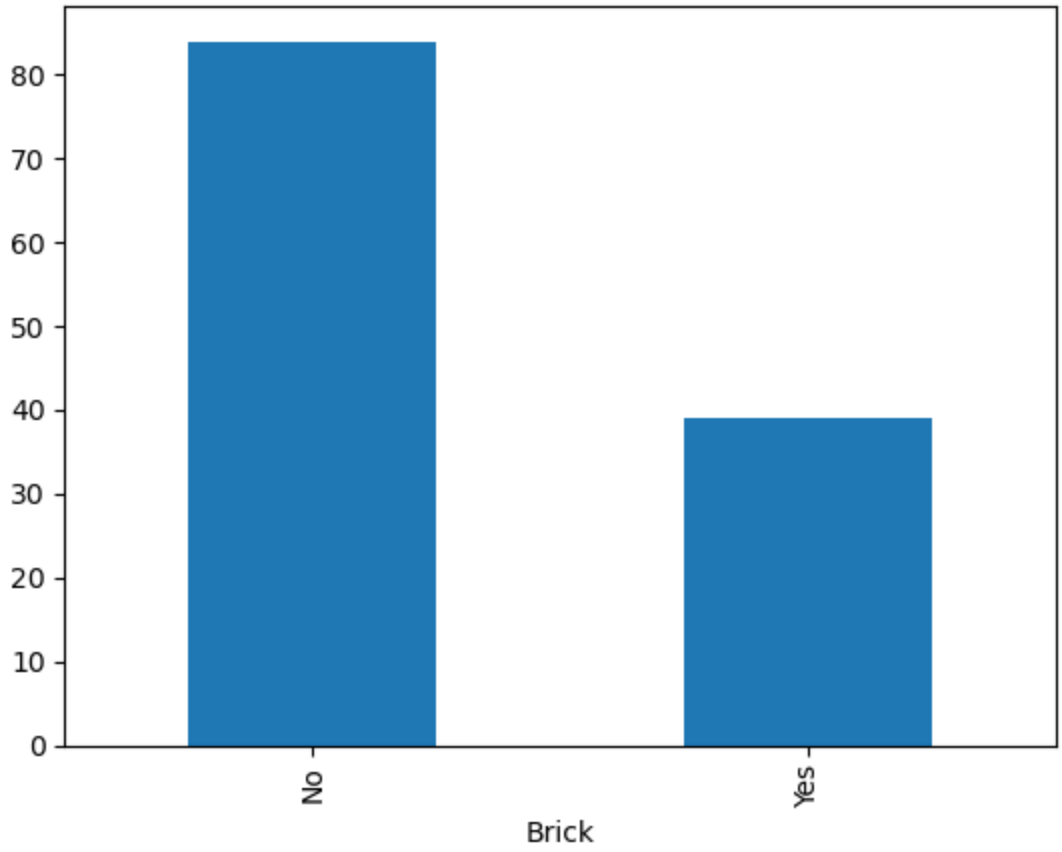
```
df.groupby("Bedrooms")["Bedrooms"].count().plot(kind="bar")
plt.show()
```



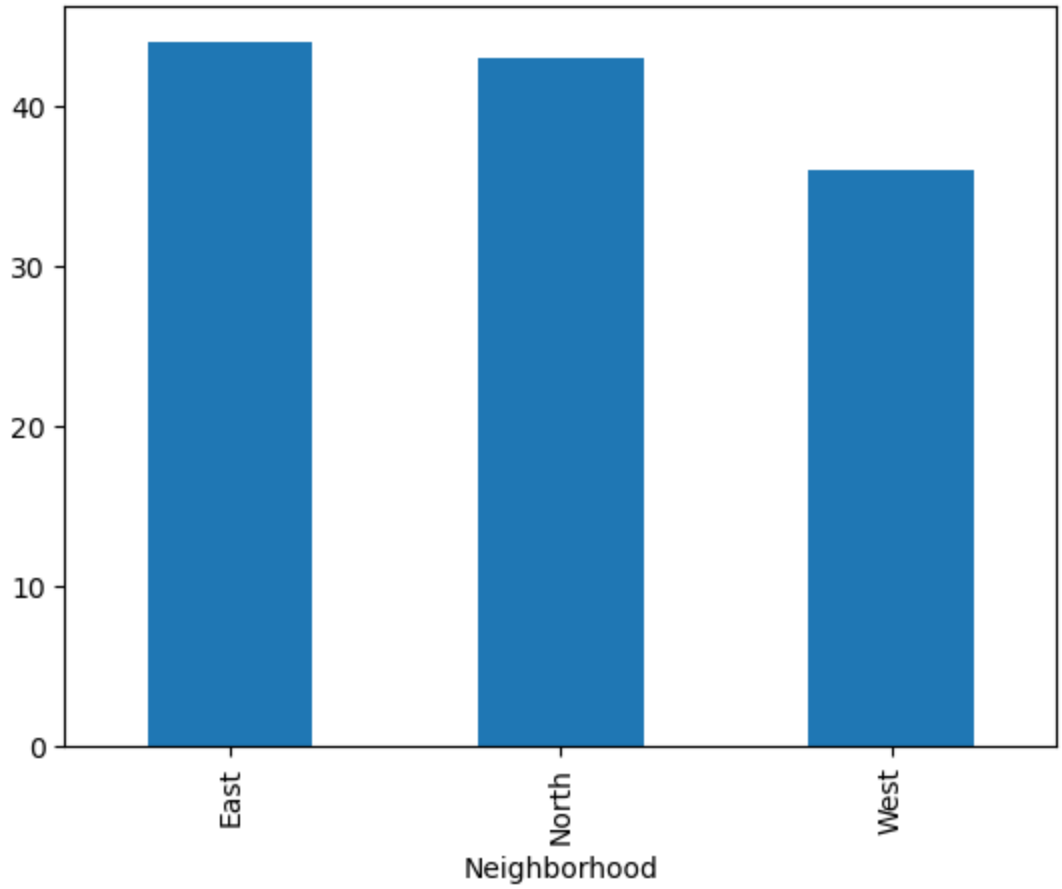
In [34]:
```
df.groupby("Bathrooms")["Bathrooms"].count().plot(kind="bar")
plt.show()
```



In [35]:
```
df.groupby("Brick")["Brick"].count().plot(kind="bar")
plt.show()
```

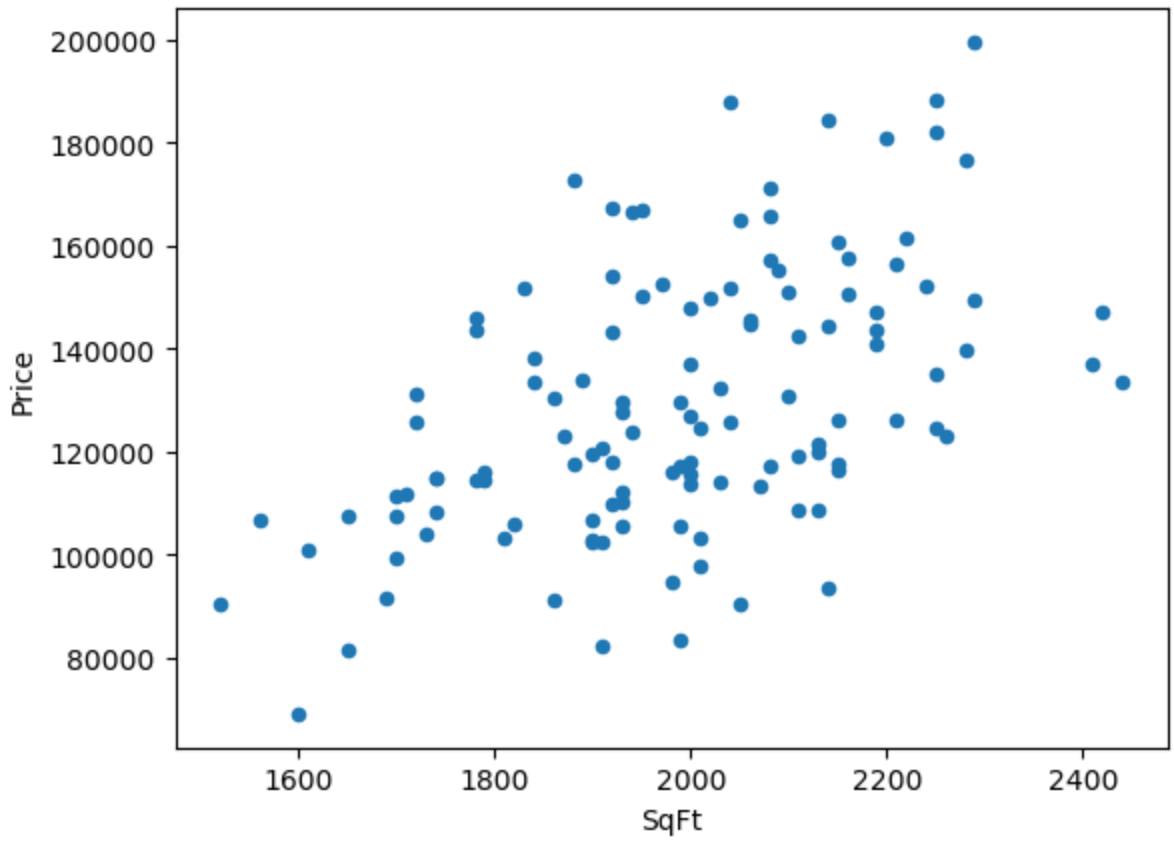


In [36]:
```
df.groupby("Neighborhood")["Neighborhood"].count().plot(kind="bar")
plt.show()
```

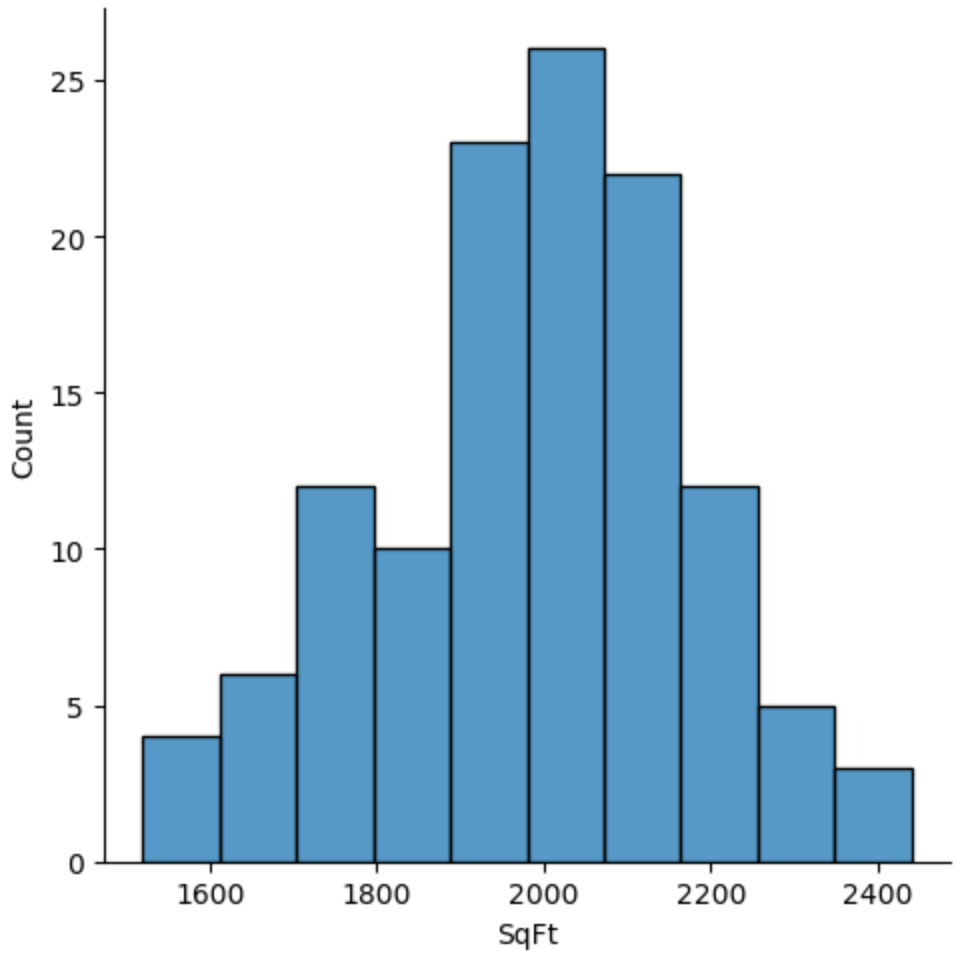


In [37]:
```
#check correlation
df.plot(kind="scatter", x="SqFt" ,y="Price")
```

Out[37]:
```
<Axes: xlabel='SqFt', ylabel='Price'>
```



In [38]:
```
#distribution
sns.displot(df["SqFt"])
plt.show()
```

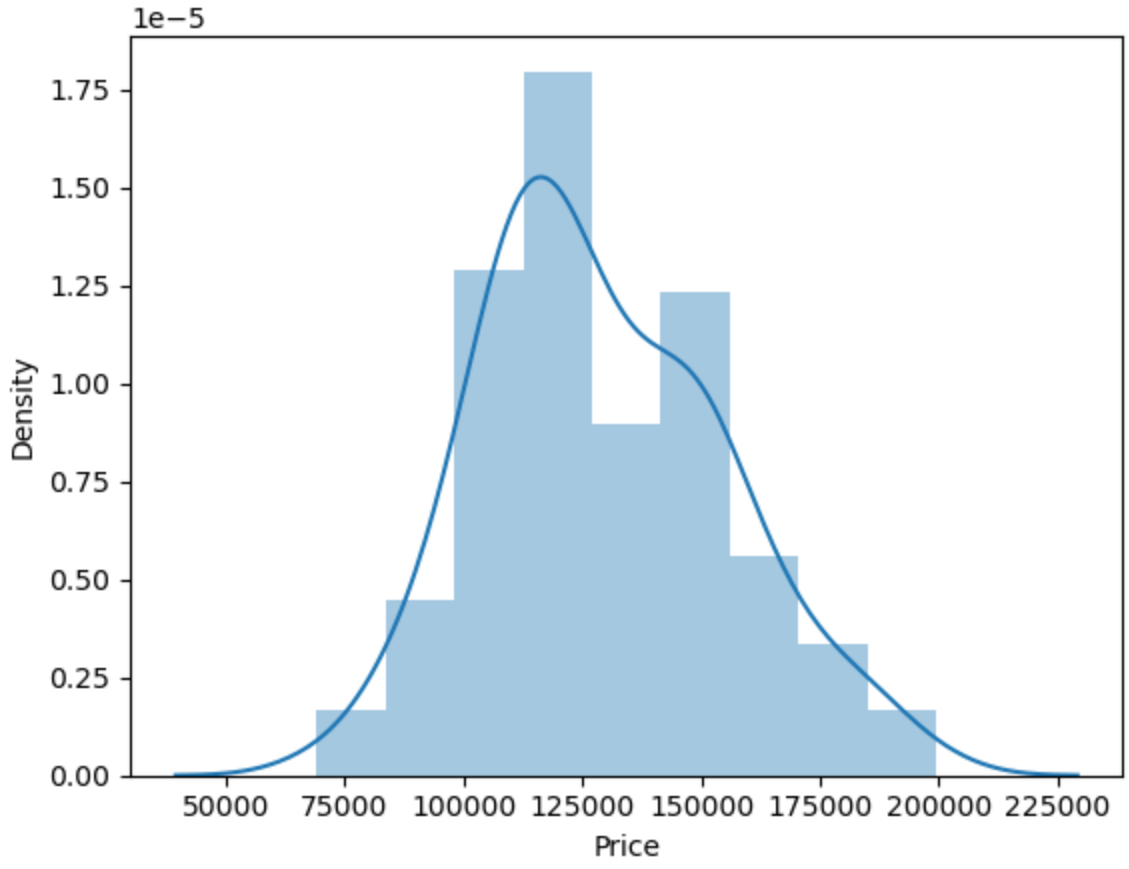


what is kurtosis? kurtosis is the peakiness in the data .

- if the plot has tailer peak then kurtosis is 1.
- if the plot forms a bell-shaped curve then kurtosis is 0
- if the plot has wider peak then kurtosis is -1

In [39]:
```
sns.distplot(df["Price"])
plt.show()
```

```
C:\Users\GIRISH\AppData\Local\Temp\ipykernel_28648\3178995481.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df["Price"])
```



In [40]:
```
#check object variable for spelling difference, and redundant data
#Bedrooms           object
#Bathrooms          object
#offers             object
#Brick              object
#Neighborhood       object
```

In [41]:
```
df["Bedrooms"].unique()
```

Out[41]:
```
array([2, 4, 3, 5], dtype=object)
```

In [42]:
```
df["Bathrooms"].unique()
```
Out[42]:
```
array([2, 3, 4], dtype=object)
```
In [43]:
```
df["Offers"].unique()
```
Out[43]:
```
array([2, 3, 1, 4, 5, 6], dtype=object)
```
In [44]:
```
df["Brick"].unique()
```
Out[44]:
```
array(['No', 'Yes'], dtype=object)
```
In [45]:
```
df["Neighborhood"].unique()
```
Out[45]:
```
array(['East', 'North', 'West'], dtype=object)
```

Feature Engineering: one-hot-encoding(dummy conversion)

In [46]:
```
#store all categorical variables in a new DataFrame
df_catergorical=df.select_dtypes(include=["object"])
```
In [47]:
```
df_catergorical.dtypes
```
Out[47]:
```
Bedrooms        object
Bathrooms       object
Offers          object
Brick           object
Neighborhood    object
dtype: object
```
In [48]:
```
#create dummy
dummy=pd.get_dummies(df_catergorical, drop_first=True)
```
```
C:\Users\GIRISH\AppData\Local\Temp\ipykernel_28648\3414680821.py:2: FutureWarning: In a future version, the Index constructor will not infer numeric dtypes when passed object-dtype sequences (matching Series behavior)
  dummy=pd.get_dummies(df_catergorical, drop_first=True)
C:\Users\GIRISH\AppData\Local\Temp\ipykernel_28648\3414680821.py:2: FutureWarning: In a future version, the Index constructor will not infer numeric dtypes when passed object-dtype sequences (matching Series behavior)
  dummy=pd.get_dummies(df_catergorical, drop_first=True)
C:\Users\GIRISH\AppData\Local\Temp\ipykernel_28648\3414680821.py:2: FutureWarning: In a future version, the Index constructor will not infer numeric dtypes when passed object-dtype sequences (matching Series behavior)
  dummy=pd.get_dummies(df_catergorical, drop_first=True)
```
In [49]:
```
dummy.head()
```
Out[49]:

	Bedrooms_3	Bedrooms_4	Bedrooms_5	Bathrooms_3	Bathrooms_4	Offers_2	Offers_3	Offers_4	Offers_5	Of
0	0	0	0	0	0	1	0	0	0	
1	0	1	0	0	0	0	1	0	0	
2	1	0	0	0	0	0	0	0	0	
3	1	0	0	0	0	0	1	0	0	
4	1	0	0	1	0	0	1	0	0	

In [50]:
```
#combine numeric columns from df with dummy columns to create the first data
df_numeric=df.select_dtypes(include=["int64","float64"])
```

In [51]:
```
df_master=pd.concat([df_numeric,dummy], axis=1)
```

In [52]:
```
df_master.head()
```

Out[52]:

	Price	SqFt	Bedrooms_3	Bedrooms_4	Bedrooms_5	Bathrooms_3	Bathrooms_4	Offers_2	Offers_3	Offers
0	114300	1790.0	0	0	0	0	0	1	0	
1	114200	2030.0	0	1	0	0	0	0	1	
2	114800	1740.0	1	0	0	0	0	0	0	
3	94700	1980.0	1	0	0	0	0	0	1	
4	119800	2130.0	1	0	0	1	0	0	1	

In [53]:
```
#expert final data to excel
#df_master to_excel(r"c:\user\Mukesh\Desktop\final.xlsx")
```

create x(with all independent variables) and y(with the target variable)

In [54]:
```
x=df_master.drop("Price", axis=1)
```

In [55]:
```
y=df_master["Price"]
```

Random sampling: create training and test samples

In [56]:
```
#create training and test samples
xtrain, xtest, ytrain, ytest=train_test_split(x,y,train_size=0.7,random_state=0)
```

Feature Selection

In [57]:
```
#create key_features object to slect the top k features
#key_features = selectKBest(score_func=f_regression, k="all")

key_features = SelectKBest(score_func=f_regression, k=5)
#to select 5 significant features

# Fit the key_features to the training data and transform it
xtrain_selected = key_features.fit_transform(xtrain, ytrain)

# Get the indives of the selected features
selected_indices = key_features.get_support(indices=True)

# Get the names of the selected features
selected_features = xtrain.columns[selected_indices]
```

In [58]:
```
selected_features
```

Out[58]:
```
Index(['SqFt', 'Bedrooms_4', 'Brick_Yes', 'Neighborhood_North',
       'Neighborhood_West'],
      dtype='object')
```

Instantiate Linear Regression

In [59]:
```
linreg=LinearRegression()
```

Model 1; Build training model using all features

In [60]:
```
#train your model
linreg.fit(xtrain, ytrain)
```

Out[60]:
```
▾ LinearRegression
LinearRegression()
```

In [61]:
```
#check the accuracyof trained model
linreg.score(xtrain, ytrain)
#Accuracy: 0.888445199238056
```

Out[61]:
```
0.888445199238056
```

In [62]:
```
#test your models learning
#predict house price using test sample
predicted_price=linreg.predict(xtest)
```

In [63]:
```
#check r-squarred (accuracy score)
linreg.score(xtest, ytest)
#Accuracy:- 0.7824520988380175
```

Out[63]:
```
0.7824520988380175
```

Model 2: buld model using KBest selected features

In [64]:
```
#store KBest columns from xtrain to xtarain_kbest
xtrain_kbest=xtrain[selected_features]
```

In [65]:
```
xtrain_kbest.head()
```

Out[65]:

	SqFt	Bedrooms_4	Brick_Yes	Neighborhood_North	Neighborhood_West
6	1830.0	0	1	0	1
110	1710.0	0	0	1	0
49	1700.0	0	1	1	0
95	1970.0	0	1	0	1
111	1740.0	0	0	1	0

In [66]:
```
#train your model
linreg.fit(xtrain_kbest, ytrain)
```

Out[66]:
```
▾ LinearRegression
LinearRegression()
```

In [67]:
```
linreg.score(xtrain_kbest, ytrain)
```

Out[67]:
```
0.8190376688489006
```

In [69]:
```
#test model
```

In [70]:
```
#store KBest coluns from xtrain to xtrain_kbest
xtest_kbest=xtest[selected_features]
```

In [72]:
```
pred_y=linreg.predict(xtest_kbest)
```

In [73]:
```
linreg.score(xtest_kbest, ytest)
```

Out[73]:
```
0.6450366270077347
```

conclusion:

- Model 2 (with 5 key features)has much lower accuracy and hence we need more data to train the model properly.
- If there no avaliability of additional data then we will continue with model 1 (including all features)

Implementation:

- Ask client to share new data(that bovioselt wont have the target variable)
- Load new data in pandas(say df_new) and perform akk the steps to clean and prepare the data
- donot create training and test samples
- directly parse new_df to linreg.predict and store the predicted output in a new variable(say new_prd_y)
- Export the predicted ooutput to excel. Combine the predicted ouput column to new_df
- share the output file with your client and stakeholders

In [ ]:
```
```