import the required libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
```

In [2]:
```
data = pd.read_csv('/home/tamanna/Downloads/Housing.csv')
```

Understanding the data

In [3]:
```
data.head()
```

Out[3]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotw
0	13300000	7420	4	2	3	yes	no	no	
1	12250000	8960	4	4	4	yes	no	no	
2	12250000	9960	3	2	2	yes	no	yes	
3	12215000	7500	4	2	2	yes	no	yes	
4	11410000	7420	4	1	2	yes	yes	yes	

In [4]:
```
data.tail()
```

Out[4]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hot
540	1820000	3000	2	1	1	yes	no	yes	
541	1767150	2400	3	1	1	no	no	no	
542	1750000	3620	2	1	1	yes	no	no	
543	1750000	2910	3	1	1	no	no	no	
544	1750000	3850	3	1	2	yes	no	no	

In [5]:
```
data.shape
```

Out[5]:
```
(545, 13)
```

In [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   price             545 non-null    int64 
 1   area              545 non-null    int64 
 2   bedrooms          545 non-null    int64 
 3   bathrooms         545 non-null    int64 
 4   stories           545 non-null    int64 
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           545 non-null    int64 
 11  prefarea          545 non-null    object
 12  furnishingstatus  545 non-null    object
dtypes: int64(6), object(7)
memory usage: 55.5+ KB
```

In [7]:
```
data.columns
```

Out[7]:
```
Index(['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',
       'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
       'parking', 'prefarea', 'furnishingstatus'],
      dtype='object')
```

In [8]:
```
data.describe(include = 'all')
```

Out[8]:

	price	area	bedrooms	bathrooms	stories	mainroad	g
count	5.450000e+02	545.000000	545.000000	545.000000	545.000000	545	
unique	NaN	NaN	NaN	NaN	NaN	2	
top	NaN	NaN	NaN	NaN	NaN	yes	
freq	NaN	NaN	NaN	NaN	NaN	468	
mean	4.766729e+06	5150.541284	2.965138	1.286239	1.805505	NaN	
std	1.870440e+06	2170.141023	0.738064	0.502470	0.867492	NaN	
min	1.750000e+06	1650.000000	1.000000	1.000000	1.000000	NaN	
25%	3.430000e+06	3600.000000	2.000000	1.000000	1.000000	NaN	
50%	4.340000e+06	4600.000000	3.000000	1.000000	2.000000	NaN	
75%	5.740000e+06	6360.000000	3.000000	2.000000	2.000000	NaN	
max	1.330000e+07	16200.000000	6.000000	4.000000	4.000000	NaN	

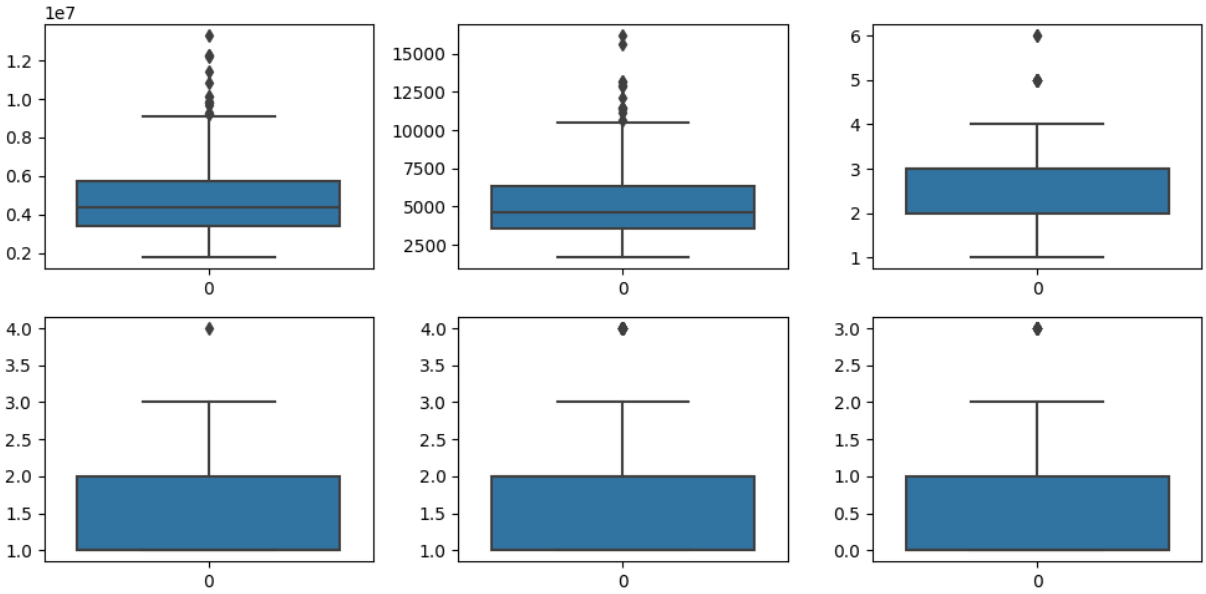
checking for the null values

In [9]:
```
data.isnull().sum()
```

Out[9]:
```
price             0
area              0
bedrooms          0
bathrooms         0
stories           0
mainroad          0
guestroom         0
basement          0
hotwaterheating   0
airconditioning   0
parking           0
prefarea          0
furnishingstatus  0
dtype: int64
```

checking for the outliers

In [10]:
```
fig, axs = plt.subplots(2,3, figsize = (10,5))
plt1 = sns.boxplot(data['price'], ax = axs[0,0])
plt2 = sns.boxplot(data['area'], ax = axs[0,1])
plt3 = sns.boxplot(data['bedrooms'], ax = axs[0,2])
plt1 = sns.boxplot(data['bathrooms'], ax = axs[1,0])
plt2 = sns.boxplot(data['stories'], ax = axs[1,1])
plt3 = sns.boxplot(data['parking'], ax = axs[1,2])

plt.tight_layout()
```



price and area have outliers

detecting and removing outliers of price column

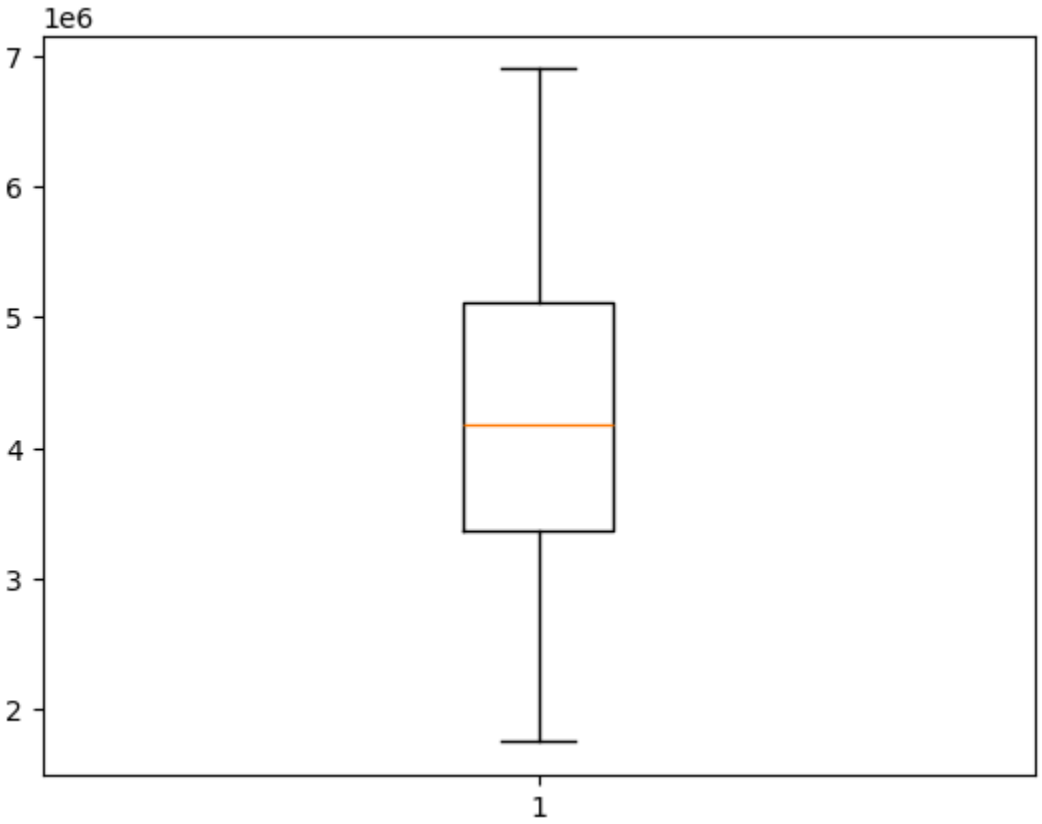
In [11]:
```
q1, q3 = np.percentile(data['price'], [25,75])
```

In [12]:
```
iqr_value = q3 - q1
```

```
lower_bound = q1 - (1.5 * iqr_value)
upper_bound = q1 + (1.5 * iqr_value)

data = data[(data['price']>= lower_bound) & (data['price'] <= upper_bound)]
#outliers removed from price column
```

In [13]:
```
plt.boxplot(data.price)
```

Out[13]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x7f90098fcb50>,
  <matplotlib.lines.Line2D at 0x7f90098fd590>],
 'caps': [<matplotlib.lines.Line2D at 0x7f90098fdf10>,
  <matplotlib.lines.Line2D at 0x7f90098fe7d0>],
 'boxes': [<matplotlib.lines.Line2D at 0x7f90098fc3d0>],
 'medians': [<matplotlib.lines.Line2D at 0x7f90098ff050>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f90098ff8d0>],
 'means': []}
```

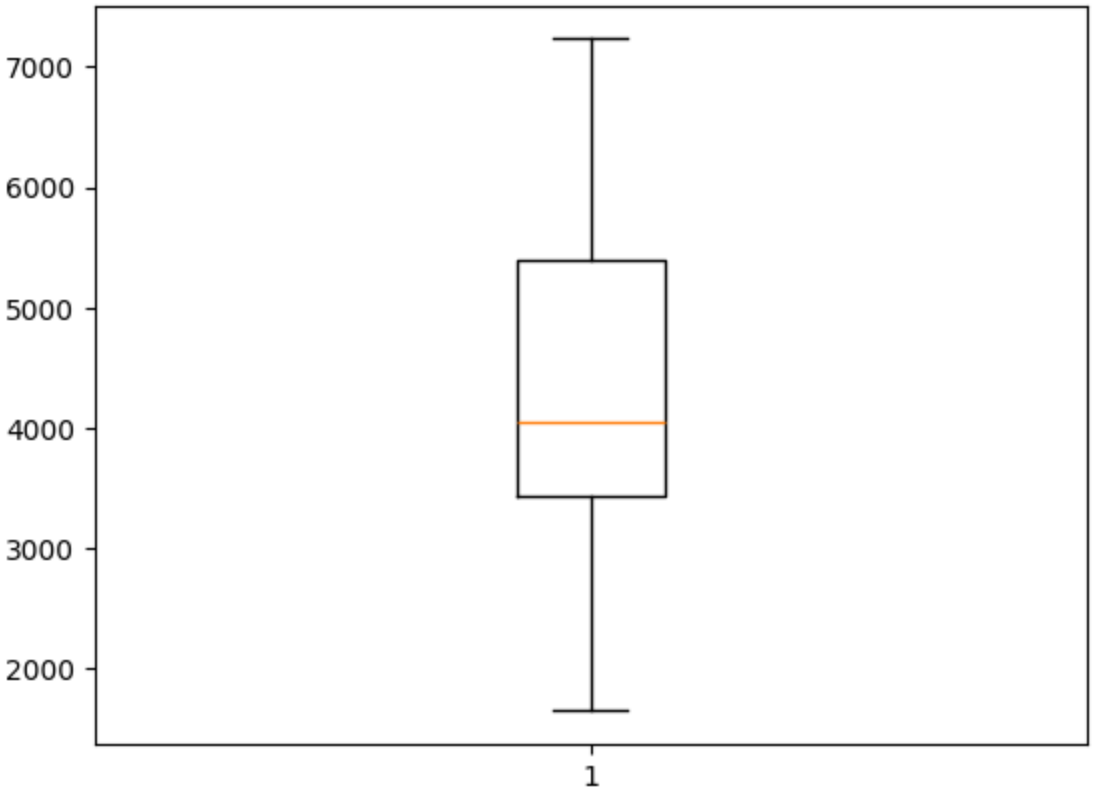


detecting and removing outlier of area column

In [14]:
```
q1, q3 = np.percentile(data['area'], [25, 75])

iqr_value = q3 - q1

lower_bound = q1 - (1.5 * iqr_value)
upper_bound = q1 + (1.5 * iqr_value)

data = data[(data['area'] >= lower_bound) & (data['area'] <= upper_bound)]
# outliers removed from area column
```

In [15]:
```
plt.boxplot(data.area)
```

Out[15]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x7f900995b610>,
  <matplotlib.lines.Line2D at 0x7f9009968210>],
 'caps': [<matplotlib.lines.Line2D at 0x7f9009968d90>,
  <matplotlib.lines.Line2D at 0x7f9009969950>],
 'boxes': [<matplotlib.lines.Line2D at 0x7f906412f510>],
 'medians': [<matplotlib.lines.Line2D at 0x7f900996a4d0>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f900996afd0>],
 'means': []}
```



In [16]:
```
# handling binary categorical variables

# List of categorical columns containing 'yes' and 'no' values
categorical_col =  ['mainroad', 'guestroom', 'basement', 'hotwaterheating',
data[categorical_col]
```

Out[16]:

	mainroad	guestroom	basement	hotwaterheating	airconditioning	prefarea
68	yes	no	no	no	yes	no
70	yes	no	yes	no	yes	yes
71	yes	no	no	no	yes	no
72	yes	no	no	no	yes	yes
73	yes	no	yes	no	no	yes
...	...	...	...	...	...	...
540	yes	no	yes	no	no	no
541	no	no	no	no	no	no
542	yes	no	no	no	no	no
543	no	no	no	no	no	no
544	yes	no	no	no	no	no

428 rows × 6 columns

In [17]:
```
def binary_map(x):
    return x.map({'yes' : 1, 'no' : 0})
```

In [18]:
```
data[categorical_col] = data[categorical_col].apply(binary_map)

data[categorical_col]
```

Out[18]:

	mainroad	guestroom	basement	hotwaterheating	airconditioning	prefarea
68	1	0	0	0	1	0
70	1	0	1	0	1	1
71	1	0	0	0	1	0
72	1	0	0	0	1	1
73	1	0	1	0	0	1
...	...	...	...	...	...	...
540	1	0	1	0	0	0
541	0	0	0	0	0	0
542	1	0	0	0	0	0
543	0	0	0	0	0	0
544	1	0	0	0	0	0

428 rows × 6 columns

In [19]:
```
# handling categorical columns with dummy variables
dummy_var = pd.get_dummies(data['furnishingstatus'], drop_first = True)

dummy_var.head()
```

Out[19]:

	semi-furnished	unfurnished
68	False	False
70	True	False
71	False	True
72	False	True
73	False	False

In [20]:
```
data = pd.concat([data, dummy_var], axis = 1)

data.head()
```

Out[20]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotv
68	6860000	6000	3	1	1	1	0	0	
70	6790000	4000	3	2	2	1	0	1	
71	6755000	6000	4	2	4	1	0	0	
72	6720000	5020	3	1	4	1	0	0	
73	6685000	6600	2	2	4	1	0	1	

In [21]:
```
data['semi-furnished'] = data['semi-furnished'].astype(int)
```

In [22]:
```
data['unfurnished'] = data['unfurnished'].astype(int)
```

In [23]:
```
data.head()
```

Out[23]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotv
68	6860000	6000	3	1	1	1	0	0	
70	6790000	4000	3	2	2	1	0	1	
71	6755000	6000	4	2	4	1	0	0	
72	6720000	5020	3	1	4	1	0	0	
73	6685000	6600	2	2	4	1	0	1	

In [24]:
```
data.drop(['furnishingstatus'], axis = 1, inplace = True)
data.head()
```

Out[24]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotv
68	6860000	6000	3	1	1	1	0	0	
70	6790000	4000	3	2	2	1	0	1	
71	6755000	6000	4	2	4	1	0	0	
72	6720000	5020	3	1	4	1	0	0	
73	6685000	6600	2	2	4	1	0	1	

In [25]:
```
data.columns
```

Out[25]:
```
Index(['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',
       'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
       'parking', 'prefarea', 'semi-furnished', 'unfurnished'],
      dtype='object')
```

Splitting the data into training and testing dataset

In [26]:
```
train_data, test_data = train_test_split(data, train_size = 0.7, test_size =
```

In [27]:
```
train_data.head()
```

Out[27]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hot
342	3850000	7152	3	1	2	1	0	0	
262	4445000	3750	2	1	1	1	1	1	
271	4340000	1905	5	1	2	0	0	1	
83	6580000	6000	3	2	4	1	0	0	
441	3220000	4370	3	1	2	1	0	0	

In [28]:
```
train_data.shape
```

Out[28]:
```
(299, 14)
```

In [29]:
```
test_data.head()
```

Out[29]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hot
93	6300000	7200	3	2	1	1	0	1	
414	3423000	4040	2	1	1	1	0	0	
385	3570000	3640	2	1	1	1	0	0	
297	4200000	3640	3	2	2	1	0	1	
168	5250000	4260	4	1	2	1	0	1	

In [30]:
```
test_data.shape
```

Out[30]: (129, 14)

Scaling training data

In [31]:
```
scaler = MinMaxScaler()
```

In [32]:
```
col_to_scale = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking', 'pric
```

In [33]:
```
train_data[col_to_scale] = scaler.fit_transform(train_data[col_to_scale])
```

In [34]:
```
train_data.head()
```

Out[34]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	baseme
342	0.410959	0.985717	0.4	0.0	0.333333	1	0	
262	0.527397	0.370638	0.2	0.0	0.000000	1	1	
271	0.506849	0.037064	0.8	0.0	0.333333	0	0	
83	0.945205	0.777436	0.4	0.5	1.000000	1	0	
441	0.287671	0.482734	0.4	0.0	0.333333	1	0	

training the model

In [35]:
```
x_train = train_data
y_train = train_data.pop('price')
```

In [36]:
```
# target variable in training set
y_train.head()
```

Out[36]:
```
342    0.410959
262    0.527397
271    0.506849
83     0.945205
441    0.287671
Name: price, dtype: float64
```

In [37]:
```
model = LinearRegression()
```

In [38]:
```
model.fit(x_train, y_train)
```

Out[38]:
```
▾ LinearRegression

LinearRegression()
```

In [39]:
```
# cofficients of linear regression model

coeff = model.coef_
print(coeff)
```

```
[ 0.28821155  0.03461766  0.17244098  0.22735911  0.06252967  0.07010271
  0.07566646  0.15782631  0.09612357  0.11184875  0.09691074  0.03176785
 -0.06208333]
```

In [40]:
```
score = model.score(x_train, y_train)

print(score)
```

```
0.6229583190886906
```

scaling the test data

In [41]:
```
scaler = MinMaxScaler()
```

In [42]:
```
col_to_scale = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking', 'pric
```

In [43]:
```
test_data[col_to_scale] = scaler.fit_transform(test_data[col_to_scale])
```

In [44]:
```
test_data.head()
```

Out[44]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	baseme
93	0.909091	1.000000	0.4	1.0	0.000000	1	0	
414	0.334266	0.430631	0.2	0.0	0.000000	1	0	
385	0.363636	0.358559	0.2	0.0	0.000000	1	0	
297	0.489510	0.358559	0.4	1.0	0.333333	1	0	
168	0.699301	0.470270	0.6	0.0	0.333333	1	0	

Model testing

In [45]:
```
# Separate the target variable from the testing subset
y_test = test_data.pop('price')

# Extract the remaining features as the testing data
x_test = test_data
```

In [46]:
```
prediction = model.predict(x_test)
```

comparing the actual and predicted values

In [47]:
```
# Get the shape of y_test
y_test.shape

# Reshape y_test to a matrix with a single column
y_test_matrix = y_test.values.reshape(-1, 1)
```

In [48]:
```
# Creating a DataFrame with actual and predicted values
data_frame = pd.DataFrame({'actual': y_test_matrix.flatten(), 'predicted': p
```

In [49]:
```
# Display the first 10 rows of the DataFrame
data_frame.head(10)
```

Out[49]:

	actual	predicted
0	0.909091	0.947038
1	0.334266	0.226085
2	0.363636	0.205313
3	0.489510	0.536130
4	0.699301	0.561016
5	0.000000	0.173881
6	0.489510	0.737574
7	0.419580	0.310069
8	0.881119	0.848146
9	0.167832	0.203235

plotting the graph between actual and predicted values

In [50]:
```
# Create a new figure
fig = plt.figure()

# Scatter plot of actual versus predicted values
plt.scatter(y_test, prediction)
```

```
# Set the title and labels for the plot
plt.title('Actual vs Prediction')
plt.xlabel('Actual', fontsize=15)
plt.ylabel('Predicted', fontsize=15)
```

Out[50]: Text(0, 0.5, 'Predicted')