```
# Import the numpy and pandas package

import numpy as np
import pandas as pd

# Data Visualisation

import matplotlib.pyplot as plt
import seaborn as sns


housing = pd.DataFrame(pd.read_csv("/content/Housing.csv"))
```

```
housing.head()
```

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

Next steps: Generate code with housing View recommended plots

```
housing.shape
```

```
(545, 13)
```

```
housing.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
```

```
housing.describe()
```

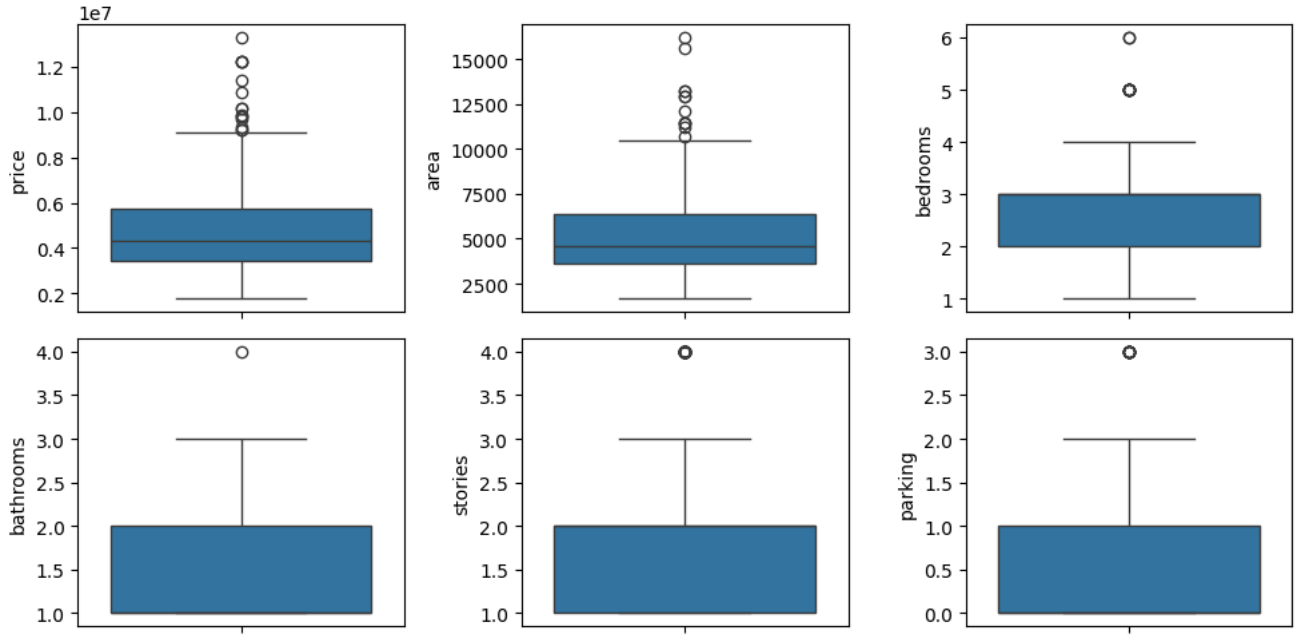
	price	area	bedrooms	bathrooms	stories	parking
count	5.450000e+02	545.000000	545.000000	545.000000	545.000000	545.000000
mean	4.766729e+06	5150.541284	2.965138	1.286239	1.805505	0.693578
std	1.870440e+06	2170.141023	0.738064	0.502470	0.867492	0.861586
min	1.750000e+06	1650.000000	1.000000	1.000000	1.000000	0.000000
25%	3.430000e+06	3600.000000	2.000000	1.000000	1.000000	0.000000
50%	4.340000e+06	4600.000000	3.000000	1.000000	2.000000	0.000000
75%	5.740000e+06	6360.000000	3.000000	2.000000	2.000000	1.000000
max	1.330000e+07	16200.000000	6.000000	4.000000	4.000000	3.000000

```
# Checking Null values
housing.isnull().sum()*100/housing.shape[0]
```

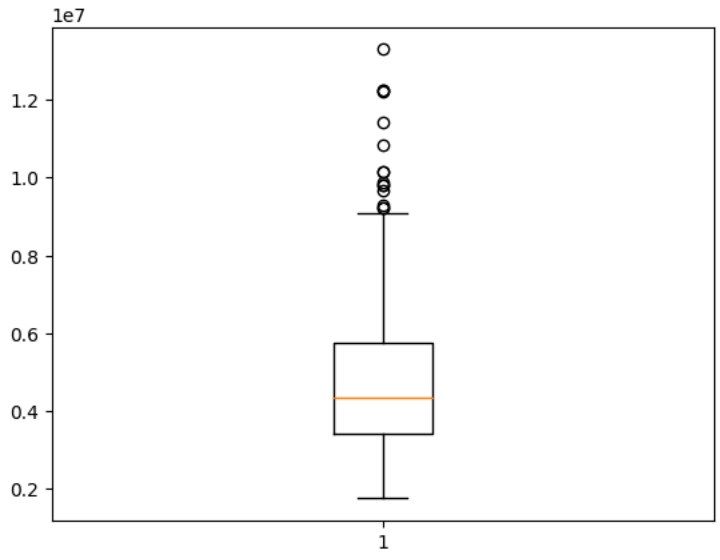
```
price             0.0
area              0.0
bedrooms          0.0
```

```
bathrooms           0.0
stories             0.0
mainroad            0.0
guestroom           0.0
basement            0.0
hotwaterheating     0.0
airconditioning     0.0
parking             0.0
prefarea            0.0
furnishingstatus    0.0
dtype: float64
```

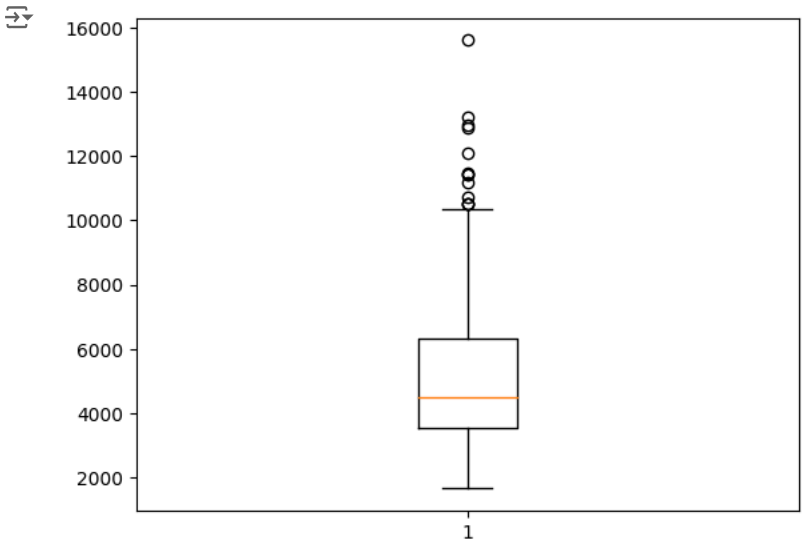
```
# Outlier Analysis
fig, axs = plt.subplots(2,3, figsize = (10,5))
plt1 = sns.boxplot(housing['price'], ax = axs[0,0])
plt2 = sns.boxplot(housing['area'], ax = axs[0,1])
plt3 = sns.boxplot(housing['bedrooms'], ax = axs[0,2])
plt1 = sns.boxplot(housing['bathrooms'], ax = axs[1,0])
plt2 = sns.boxplot(housing['stories'],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

plt.tight_layout()
```



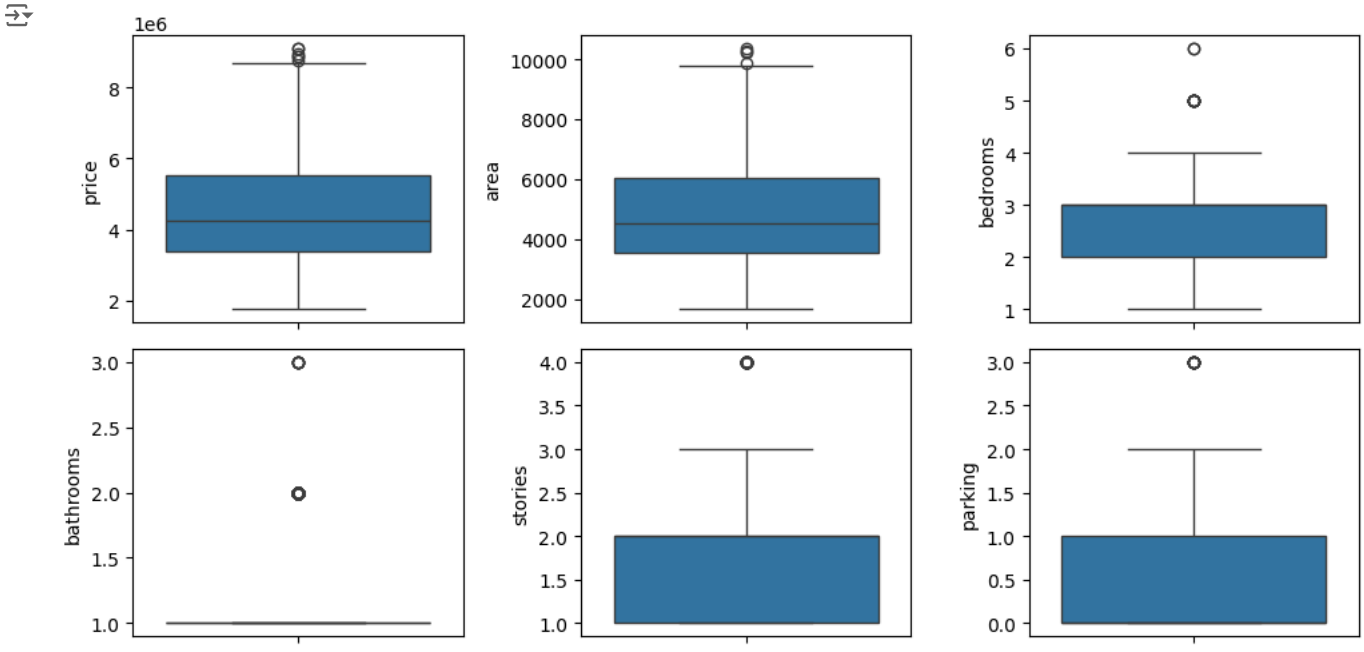

```
# outlier treatment for price
plt.boxplot(housing.price)
Q1 = housing.price.quantile(0.25)
Q3 = housing.price.quantile(0.75)
IQR = Q3 - Q1
housing = housing[(housing.price >= Q1 - 1.5*IQR) & (housing.price <= Q3 + 1.5*IQR)]
```



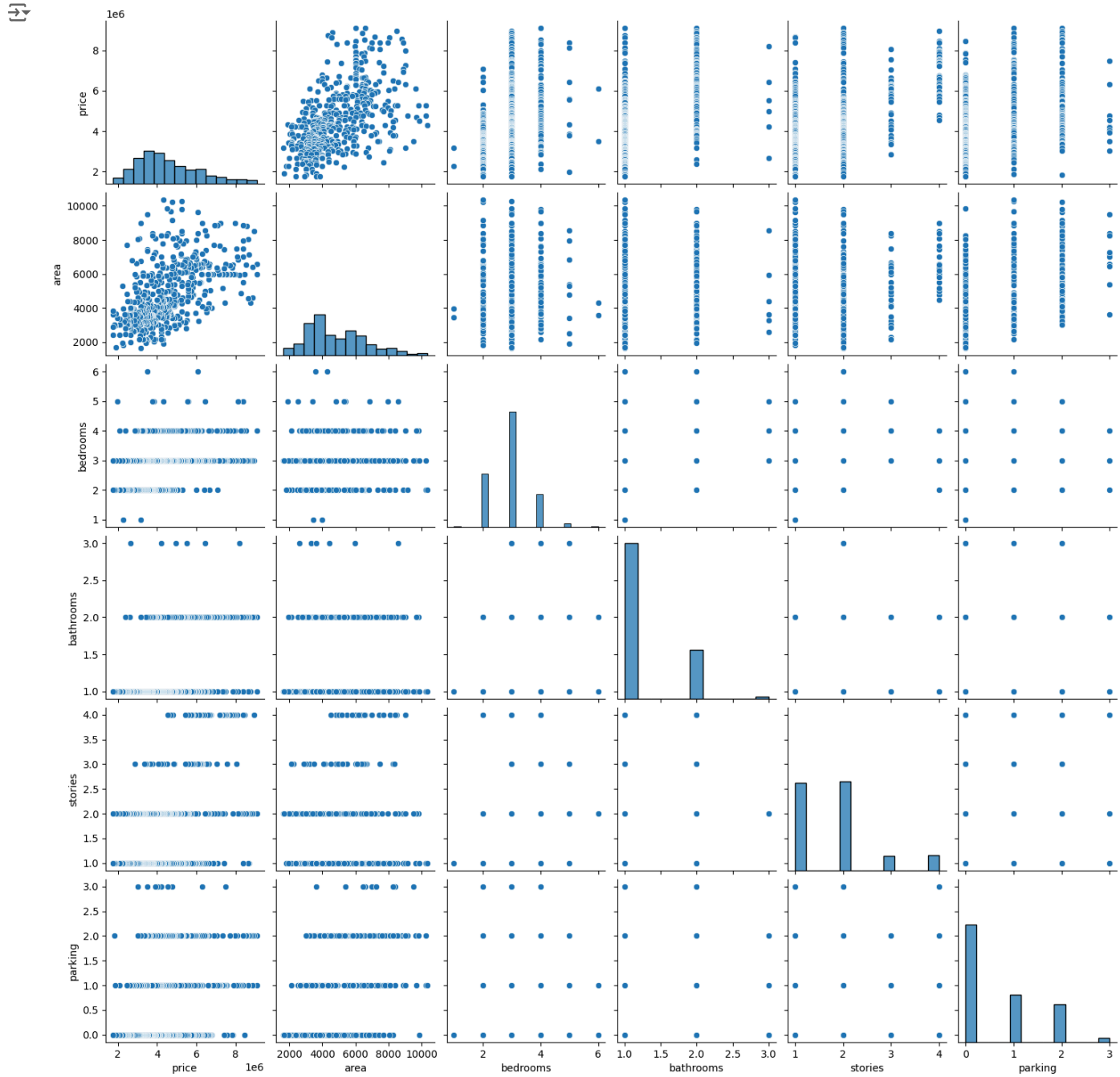
```
# outlier treatment for area
plt.boxplot(housing.area)
Q1 = housing.area.quantile(0.25)
Q3 = housing.area.quantile(0.75)
IQR = Q3 - Q1
housing = housing[(housing.area >= Q1 - 1.5*IQR) & (housing.area <= Q3 + 1.5*IQR)]
```



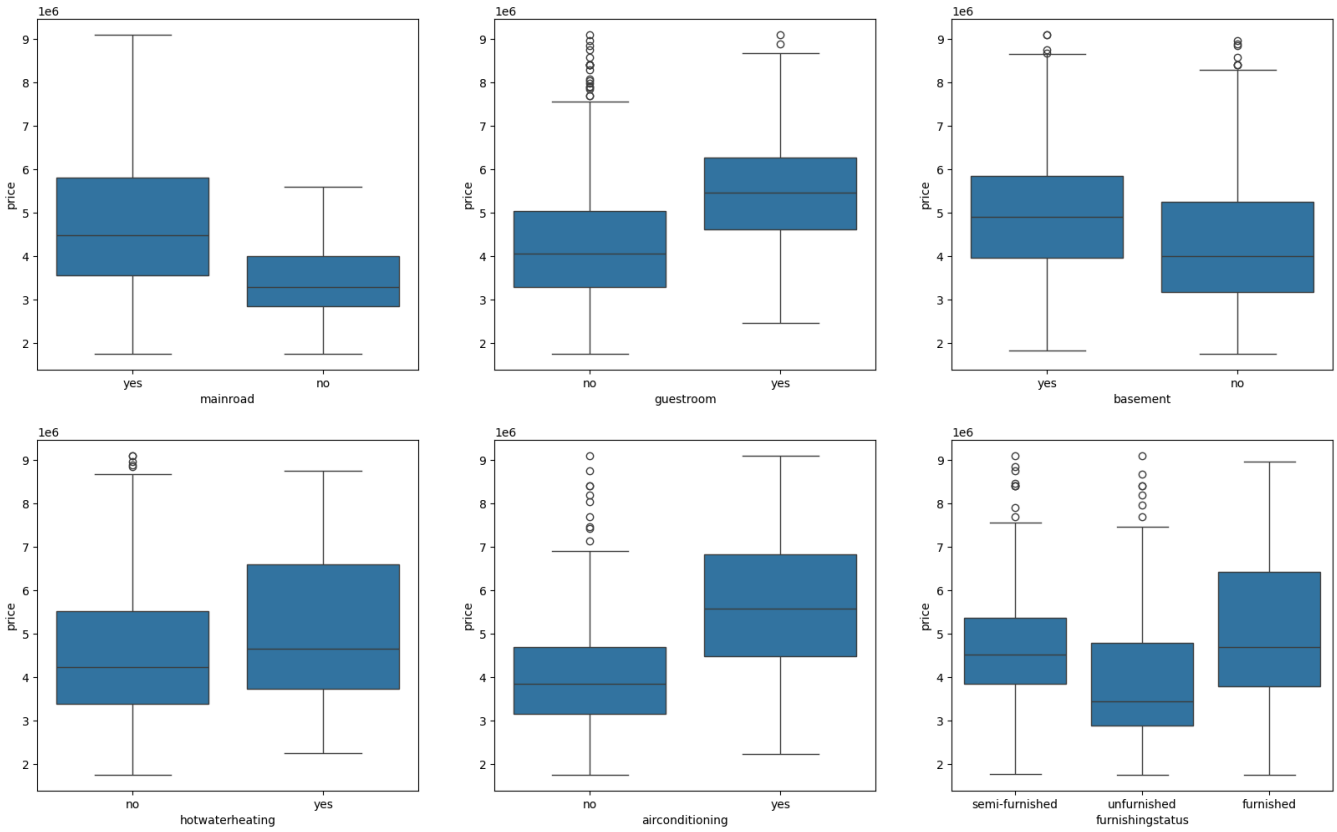

```
# Outlier Analysis
fig, axs = plt.subplots(2,3, figsize = (10,5))
plt1 = sns.boxplot(housing['price'], ax = axs[0,0])
plt2 = sns.boxplot(housing['area'], ax = axs[0,1])
plt3 = sns.boxplot(housing['bedrooms'], ax = axs[0,2])
plt1 = sns.boxplot(housing['bathrooms'], ax = axs[1,0])
plt2 = sns.boxplot(housing['stories'],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

plt.tight_layout()
```



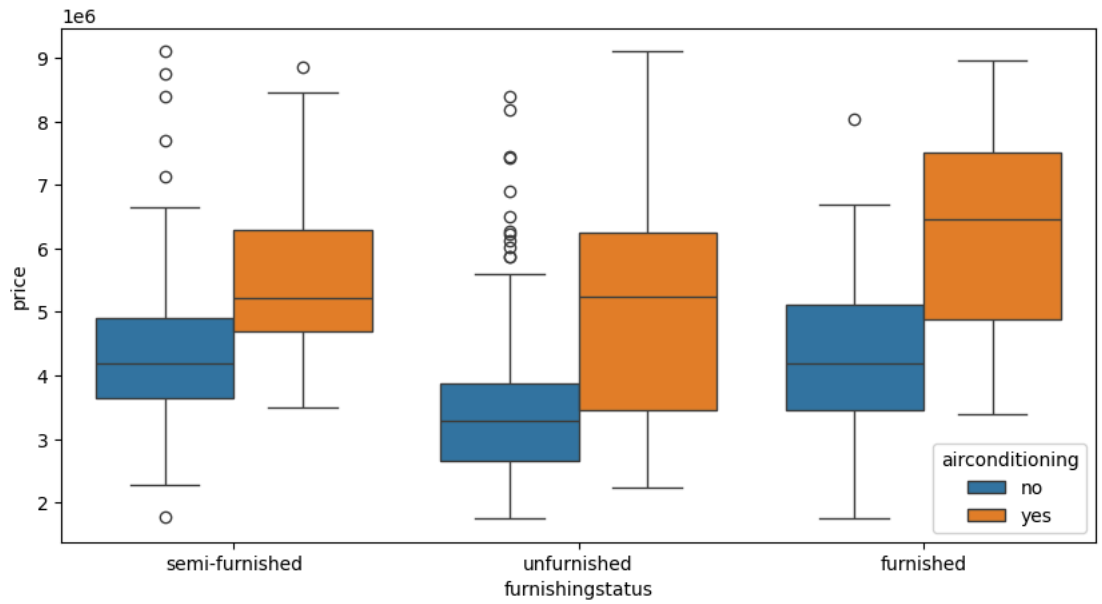

```
sns.pairplot(housing)
plt.show()
```




```
plt.figure(figsize=(20, 12))
plt.subplot(2,3,1)
sns.boxplot(x = 'mainroad', y = 'price', data = housing)
plt.subplot(2,3,2)
sns.boxplot(x = 'guestroom', y = 'price', data = housing)
plt.subplot(2,3,3)
sns.boxplot(x = 'basement', y = 'price', data = housing)
plt.subplot(2,3,4)
sns.boxplot(x = 'hotwaterheating', y = 'price', data = housing)
plt.subplot(2,3,5)
sns.boxplot(x = 'airconditioning', y = 'price', data = housing)
plt.subplot(2,3,6)
sns.boxplot(x = 'furnishingstatus', y = 'price', data = housing)
plt.show()
```




```
plt.figure(figsize = (10, 5))
sns.boxplot(x = 'furnishingstatus', y = 'price', hue = 'airconditioning', data = housing)
plt.show()
```



```
# List of variables to map

varlist =  ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'prefarea']

# Defining the map function
def binary_map(x):
    return x.map({'yes': 1, "no": 0})

# Applying the function to the housing list
housing[varlist] = housing[varlist].apply(binary_map)


# Check the housing dataframe now

housing.head()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea
15	9100000	6000	4	1	2	1	0	1	0	0	2	0
16	9100000	6600	4	2	2	1	1	1	0	1	1	1
17	8960000	8500	3	2	4	1	0	0	0	1	2	0
18	8890000	4600	3	2	2	1	1	0	0	1	2	0
19	8855000	6420	3	2	2	1	0	0	0	1	1	1

Next steps: Generate code with housing View recommended plots

```
# Get the dummy variables for the feature 'furnishingstatus' and store it in a new variable - 'status'
status = pd.get_dummies(housing['furnishingstatus'])


# Check what the dataset 'status' looks like
status.head()
```

	furnished	semi-furnished	unfurnished
15	False	True	False
16	False	False	True
17	True	False	False
18	True	False	False
19	False	True	False

Next steps: Generate code with status View recommended plots

```
# Let's drop the first column from status df using 'drop_first = True'

status = pd.get_dummies(housing['furnishingstatus'], drop_first = True)


# Add the results to the original housing dataframe

housing = pd.concat([housing, status], axis = 1)



# Now let's see the head of our dataframe.

housing.head()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea
15	9100000	6000	4	1	2	1	0	1	0	0	2	0
16	9100000	6600	4	2	2	1	1	1	0	1	1	1
17	8960000	8500	3	2	4	1	0	0	0	1	2	0
18	8890000	4600	3	2	2	1	1	0	0	1	2	0
19	8855000	6420	3	2	2	1	0	0	0	1	1	1

Next steps: Generate code with housing View recommended plots

```
# Drop 'furnishingstatus' as we have created the dummies for it

housing.drop(['furnishingstatus'], axis = 1, inplace = True)


housing.head()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea
15	9100000	6000	4	1	2	1	0	1	0	0	2	0
16	9100000	6600	4	2	2	1	1	1	0	1	1	1
17	8960000	8500	3	2	4	1	0	0	0	1	2	0
18	8890000	4600	3	2	2	1	1	0	0	1	2	0
19	8855000	6420	3	2	2	1	0	0	0	1	1	1

Next steps: Generate code with housing View recommended plots

```
from sklearn.model_selection import train_test_split

# We specify this so that the train and test data set always have the same rows, respectively
np.random.seed(0)
df_train, df_test = train_test_split(housing, train_size = 0.7, test_size = 0.3, random_state = 100)


from sklearn.preprocessing import MinMaxScaler


scaler = MinMaxScaler()


# Apply scaler() to all the columns except the 'yes-no' and 'dummy' variables
num_vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking','price']

df_train[num_vars] = scaler.fit_transform(df_train[num_vars])


df_train.head()
```

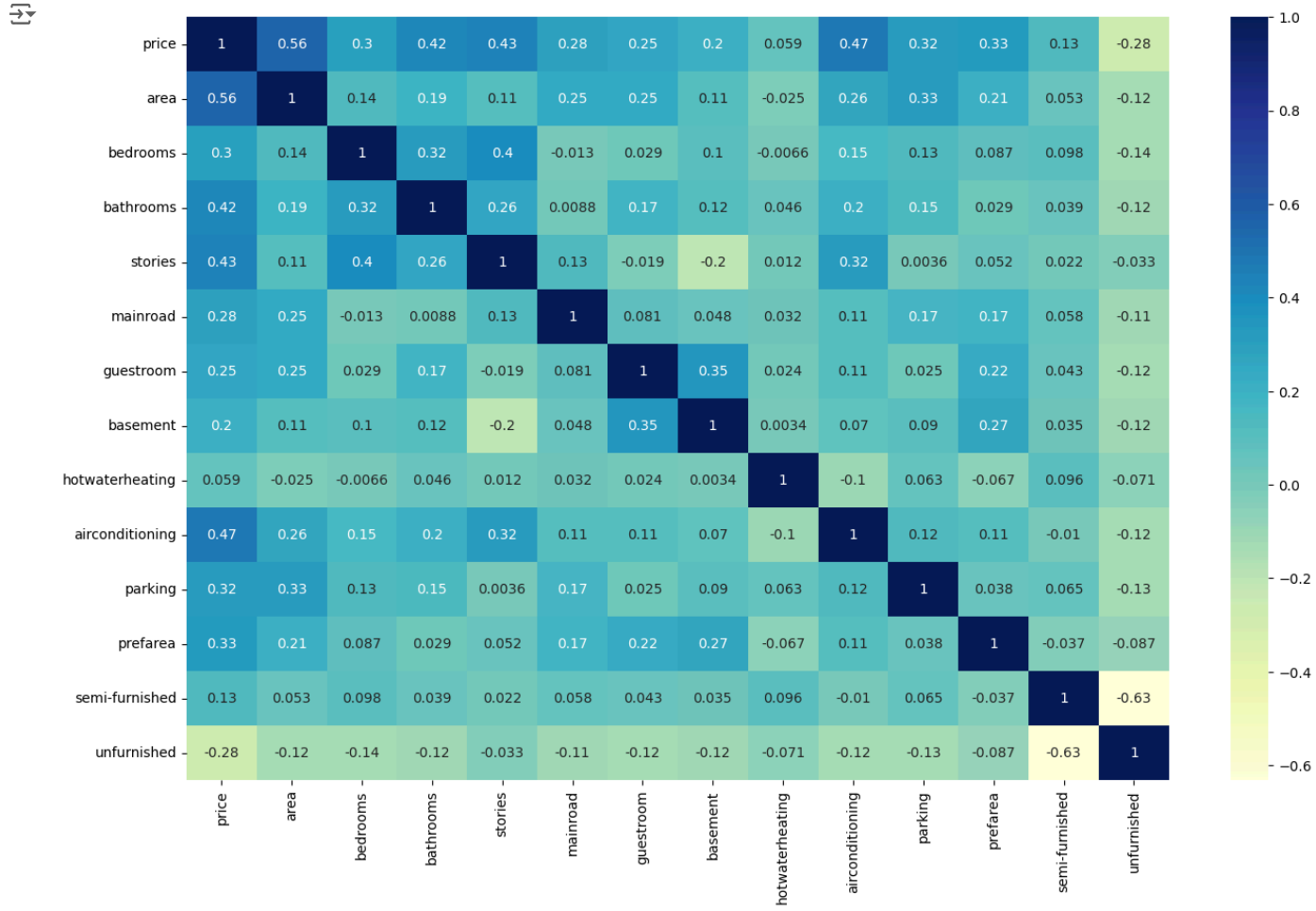
	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	pre
148	0.523810	0.526907	0.4	0.0	0.666667	1	0	0	0	0	0.000000	
236	0.390476	0.114134	0.2	0.0	0.333333	1	1	1	0	0	0.000000	
356	0.275238	0.072738	0.8	0.5	0.000000	0	0	1	0	1	0.333333	
425	0.219048	0.151390	0.2	0.0	0.000000	1	0	1	0	0	0.666667	
516	0.095238	0.157895	0.2	0.0	0.000000	0	1	0	0	0	0.333333	

Next steps: Generate code with df_train View recommended plots

```
df_train.describe()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning
count	361.000000	361.000000	361.000000	361.000000	361.000000	361.000000	361.000000	361.000000	361.000000	361.000000
mean	0.383701	0.350081	0.390582	0.127424	0.268698	0.875346	0.168975	0.349030	0.038781	0.313019
std	0.209712	0.207184	0.149146	0.224465	0.287833	0.330784	0.375250	0.477325	0.193341	0.464366
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.237143	0.189829	0.200000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000
50%	0.338095	0.295092	0.400000	0.000000	0.333333	1.000000	0.000000	0.000000	0.000000	0.000000
75%	0.514286	0.491425	0.400000	0.000000	0.333333	1.000000	0.000000	1.000000	0.000000	1.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

```
# Let's check the correlation coefficients to see which variables are highly correlated

plt.figure(figsize = (16, 10))
sns.heatmap(df_train.corr(), annot = True, cmap="YlGnBu")
plt.show()
```




```
y_train = df_train.pop('price')
X_train = df_train
```

```
# Importing RFE and LinearRegression
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
```

```
# Running RFE with the output number of the variable equal to 10
lm = LinearRegression()
lm.fit(X_train, y_train)
```

```
LinearRegression()
```

```
rfe = RFE(estimator=lm, n_features_to_select=5)
#rfe = RFE(lm, 6)             # running RFE
rfe = rfe.fit(X_train, y_train)
```

```
list(zip(X_train.columns,rfe.support_,rfe.ranking_))
```

```
[('area', True, 1),
 ('bedrooms', False, 8),
 ('bathrooms', True, 1),
 ('stories', True, 1),
 ('mainroad', False, 6),
```

```
        ('guestroom', False, 7),
        ('basement', False, 5),
        ('hotwaterheating', False, 3),
        ('airconditioning', False, 2),
        ('parking', True, 1),
        ('prefarea', True, 1),
        ('semi-furnished', False, 9),
        ('unfurnished', False, 4)]
```

```
col = X_train.columns[rfe.support_]
col
```

```
Index(['area', 'bathrooms', 'stories', 'parking', 'prefarea'], dtype='object')
```

```
# Creating X_test dataframe with RFE selected variables
X_train_rfe = X_train[col]
```

```
# Adding a constant variable
import statsmodels.api as sm
X_train_rfe = sm.add_constant(X_train_rfe)
```

```
lm = sm.OLS(y_train,X_train_rfe).fit()   # Running the linear model
```

```
#Let's see the summary of our linear model
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.573
Model:                            OLS   Adj. R-squared:                  0.567
Method:                 Least Squares   F-statistic:                     95.41
Date:                Wed, 10 Jul 2024   Prob (F-statistic):           1.72e-63
Time:                        09:20:30   Log-Likelihood:                 205.89
No. Observations:                 361   AIC:                            -399.8
Df Residuals:                     355   BIC:                            -376.5
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1092      0.015      7.079      0.000       0.079       0.139
area           0.3911      0.038     10.169      0.000       0.315       0.467
bathrooms      0.2190      0.034      6.409      0.000       0.152       0.286
stories        0.2305      0.026      8.779      0.000       0.179       0.282
parking        0.1081      0.027      4.011      0.000       0.055       0.161
prefarea       0.1161      0.018      6.339      0.000       0.080       0.152
==============================================================================
Omnibus:                       31.940   Durbin-Watson:                   2.137
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               63.240
Skew:                           0.501   Prob(JB):                     1.85e-14
Kurtosis:                       4.789   Cond. No.                         6.58
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
# Calculate the VIFs for the model
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

	Features	VIF
0	const	4.51
1	area	1.20
4	parking	1.13
2	bathrooms	1.11
3	stories	1.08
5	prefarea	1.05

Next steps: Generate code with vif View recommended plots

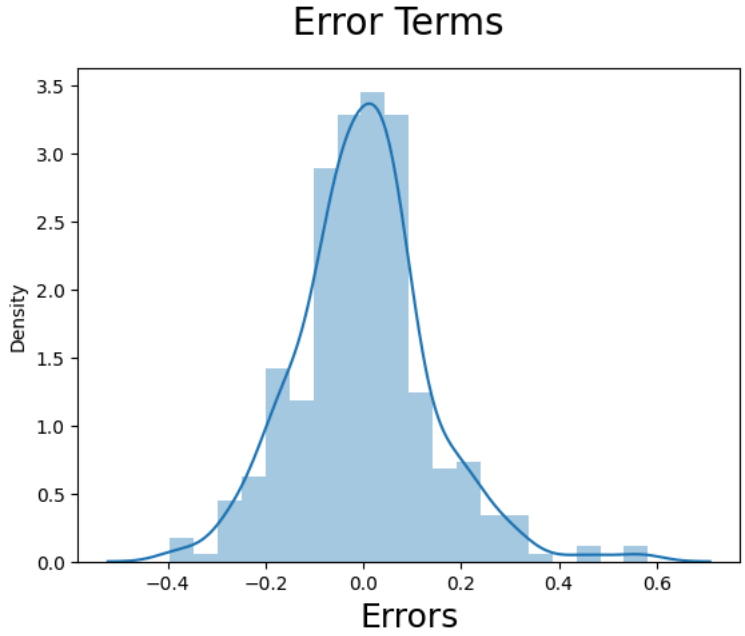
```
y_train_price = lm.predict(X_train_rfe)
```

```
res = (y_train_price - y_train)
```

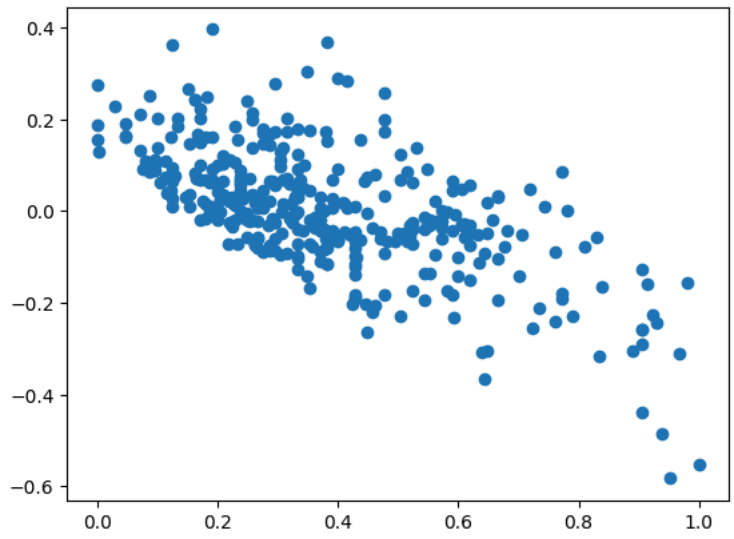
```
# Importing the required libraries for plots.
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
# Plot the histogram of the error terms
fig = plt.figure()
sns.distplot((y_train - y_train_price), bins = 20)
fig.suptitle('Error Terms', fontsize = 20)                  # Plot heading
plt.xlabel('Errors', fontsize = 18)                         # X-label
```

```
Text(0.5, 0, 'Errors')
```




```
plt.scatter(y_train,res)
plt.show()
```




```
num_vars = ['area','stories', 'bathrooms', 'airconditioning', 'prefarea','parking','price']
```

```
df_test[num_vars] = scaler.fit_transform(df_test[num_vars])
```

```
y_test = df_test.pop('price')
X_test = df_test


# Adding constant variable to test dataframe
X_test = sm.add_constant(X_test)


# Creating X_test_new dataframe by dropping variables from X_test
X_test_rfe = X_test[X_train_rfe.columns]


# Making predictions
y_pred = lm.predict(X_test_rfe)


from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

```
0.5182552232826851
```

```
# Plotting y_test and y_pred to understand the spread.
fig = plt.figure()
plt.scatter(y_test,y_pred)
fig.suptitle('y_test vs y_pred', fontsize=20)              # Plot heading
plt.xlabel('y_test', fontsize=18)                          # X-label
plt.ylabel('y_pred', fontsize=16)                          # Y-label
```

```
Text(0, 0.5, 'y_pred')
```