Multiple Linear Regression

Housing Case Study

Problem Statement:

Consider a real estate company that has a dataset containing the prices of properties in the Delhi region. It wishes to use the data to optimise the sale prices of the properties based on important factors such as area, bedrooms, parking, etc.

Essentially, the company wants —

- To identify the variables affecting house prices, e.g. area, number of rooms, bathrooms, etc.
- To create a linear model that quantitatively relates house prices with variables such as number of rooms, area, number of bathrooms, etc.
- To know the accuracy of the model, i.e. how well these variables can predict house prices.

So interpretation is important!

Step 1: Reading and Understanding the Data

Let us first import NumPy and Pandas and read the housing dataset

In [4]:

```
# Supress Warnings

import warnings
warnings.filterwarnings('ignore')
```

In [5]:

```
import numpy as np
import pandas as pd
```

In [6]:

```
housing = pd.read_csv("C:/Users/91920/Downloads/Housing.csv")
```

In [7]:

```
# Check the head of the dataset
housing.head()
```

Out[7]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterh
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	

Inspect the various aspects of the housing dataframe

In [8]:

```
housing.shape
```

Out[8]:

```
(545, 13)
```

In [9]:

```
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
```

In [10]:

```
housing.describe()
```

Out[10]:

	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

Step 2: Visualising the Data

Let's now spend some time doing what is arguably the most important step - **understanding the data**.

- If there is some obvious multicollinearity going on, this is the first place to catch it
- Here's where you'll also identify if some predictors directly have a strong association with the outcome variable

We'll visualise our data using `matplotlib` and `seaborn` .

In [11]:

```
import matplotlib.pyplot as plt
import seaborn as sns
```

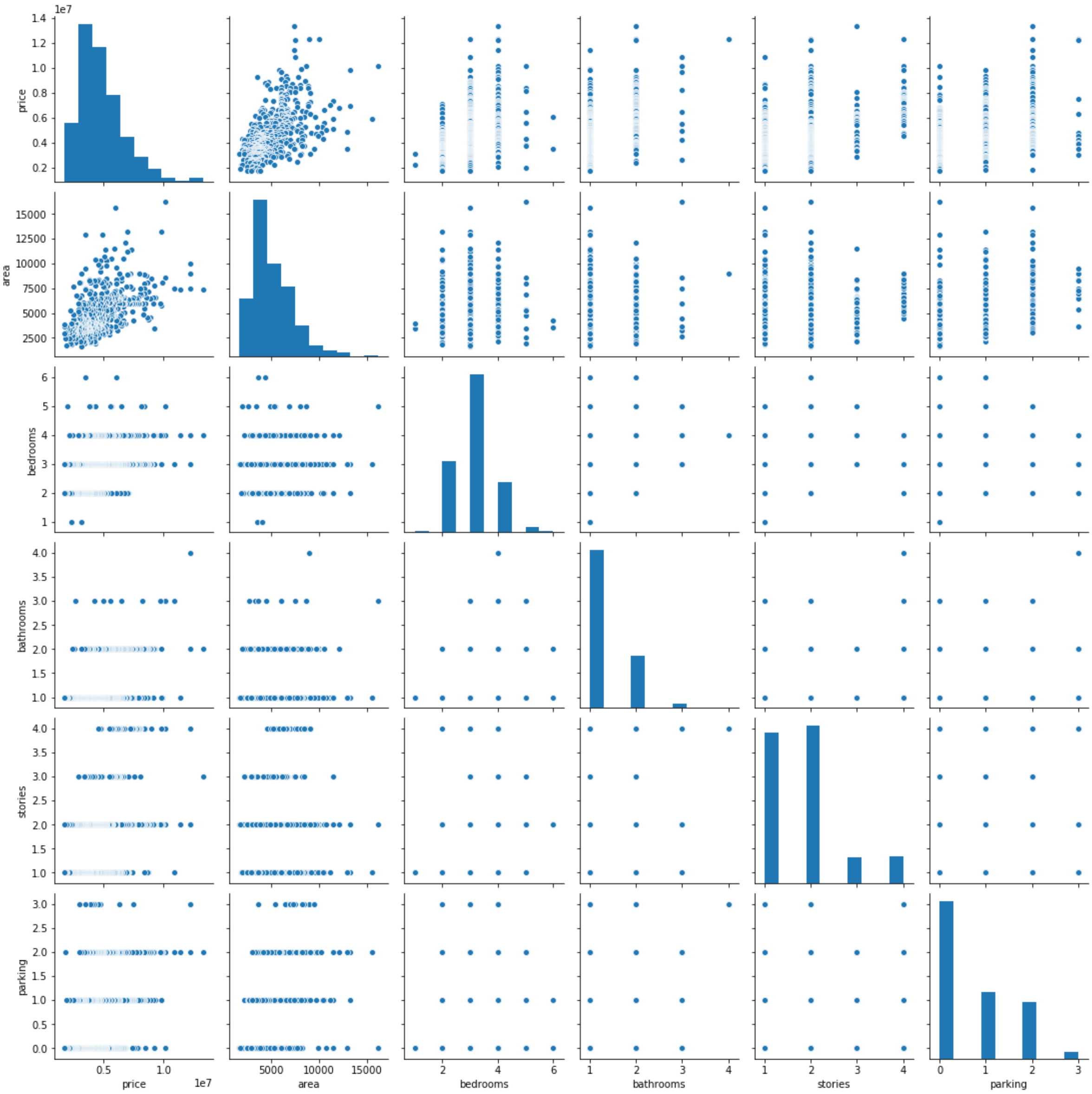
```
Bad key "text.kerning_factor" on line 4 in
C:\Users\91920\anaconda3\lib\site-packages\matplotlib\mpl-data\stylelib\_cla
ssic_test_patch.mplstyle.
You probably need to get an updated matplotlibrc file from
https://github.com/matplotlib/matplotlib/blob/v3.1.3/matplotlibrc.template
 (https://github.com/matplotlib/matplotlib/blob/v3.1.3/matplotlibrc.templat
e)
or from the matplotlib source distribution
```

Visualising Numeric Variables

Let's make a pairplot of all the numeric variables

In [12]:

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

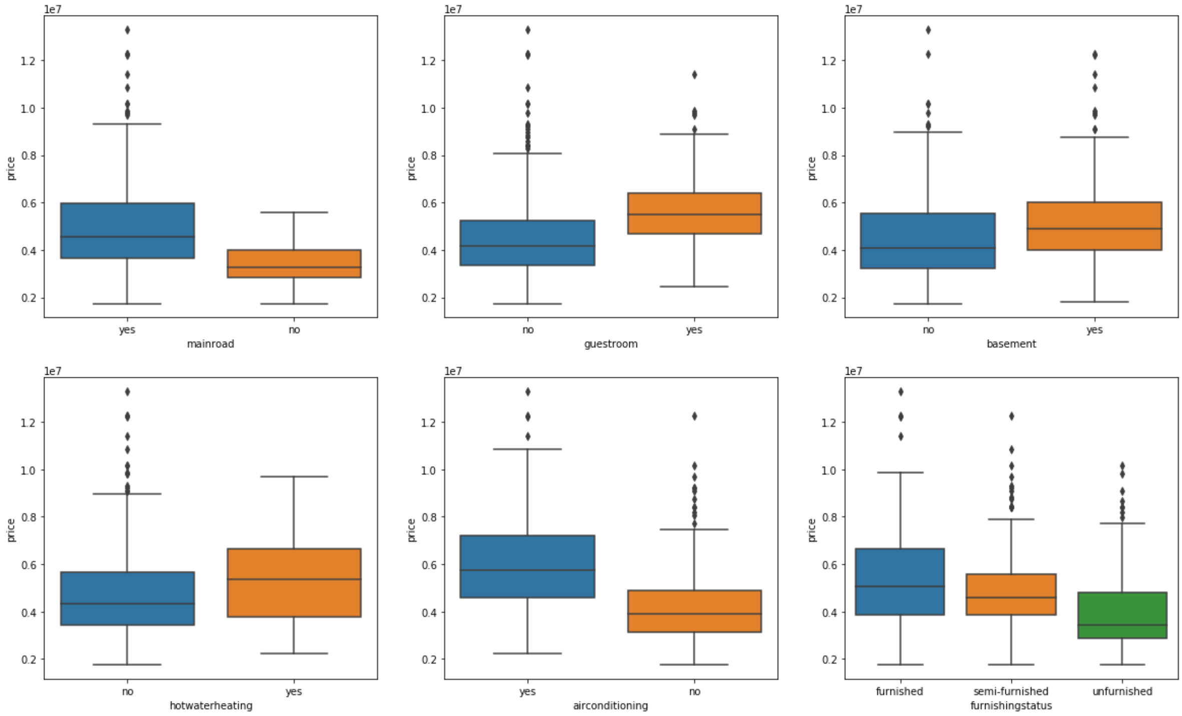


Visualising Categorical Variables

As you might have noticed, there are a few categorical variables as well. Let's make a boxplot for some of these variables.

In [11]:

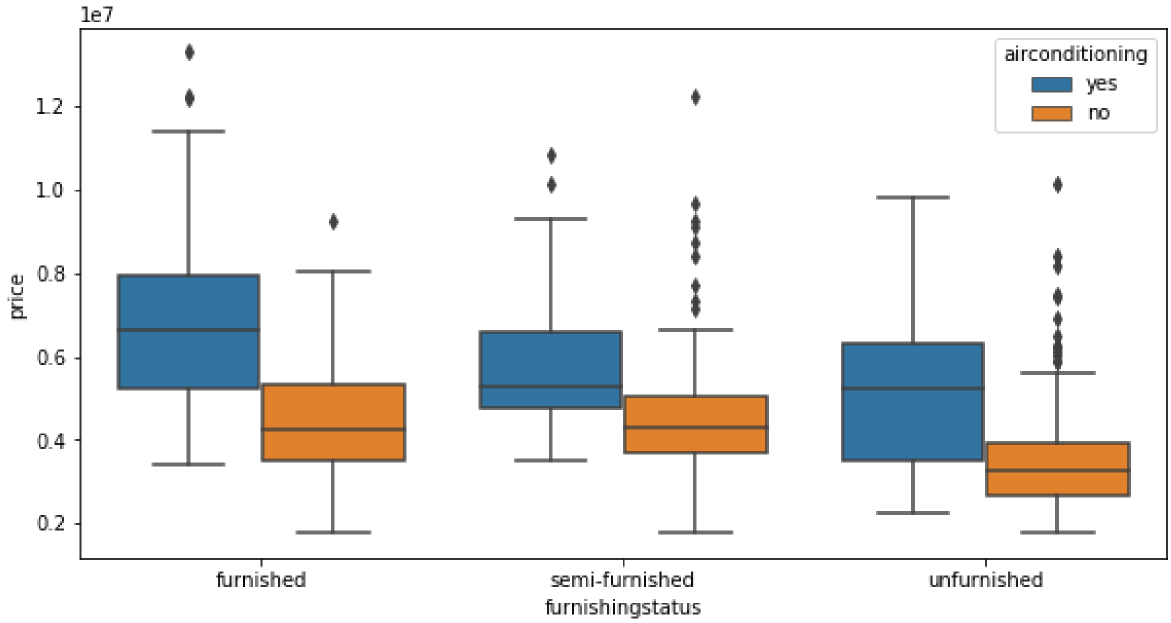
```
plt.figure(figsize=(20, 12))
plt.subplot(2,3,1)
sns.boxplot(x = 'mainroad', y = 'price', data = housing)
plt.subplot(2,3,2)
sns.boxplot(x = 'guestroom', y = 'price', data = housing)
plt.subplot(2,3,3)
sns.boxplot(x = 'basement', y = 'price', data = housing)
plt.subplot(2,3,4)
sns.boxplot(x = 'hotwaterheating',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



We can also visualise some of these categorical features parallely by using the `hue` argument. Below is the plot for `furnishingstatus` with `airconditioning` as the hue.

In [12]:

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



Step 3: Data Preparation

- You can see that your dataset has many columns with values as 'Yes' or 'No'.
- But in order to fit a regression line, we would need numerical values and not string. Hence, we need to convert them to 1s and 0s, where 1 is a 'Yes' and 0 is a 'No'.

In [13]:

```
# List of variables to map

varlist =  ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'pre

# 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)
```

In [14]:

```
# Check the housing dataframe now

housing.head()
```

Out[14]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterh
0	13300000	7420	4	2	3	1	0	0	
1	12250000	8960	4	4	4	1	0	0	
2	12250000	9960	3	2	2	1	0	1	
3	12215000	7500	4	2	2	1	0	1	
4	11410000	7420	4	1	2	1	1	1	

Dummy Variables

The variable `furnishingstatus` has three levels. We need to convert these levels into integer as well.

For this, we will use something called `dummy variables`.

In [15]:

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

In [16]:

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

Out[16]:

	furnished	semi-furnished	unfurnished
0	1	0	0
1	1	0	0
2	0	1	0
3	1	0	0
4	1	0	0

Now, you don't need three columns. You can drop the `furnished` column, as the type of furnishing can be identified with just the last two columns where —

- `00` will correspond to `furnished`
- `01` will correspond to `unfurnished`
- `10` will correspond to `semi-furnished`

In [15]:

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

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

In [16]:

```
# Add the results to the original housing dataframe

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

In [17]:

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

housing.head()
```

Out[17]:

bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning
4	2	3	1	0	0	0	1
4	4	4	1	0	0	0	1
3	2	2	1	0	1	0	0
4	2	2	1	0	1	0	1
4	1	2	1	1	1	0	1

In [18]:

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

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

In [19]:

```
housing.head()
```

Out[19]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterhe
0	13300000	7420	4	2	3	1	0	0	
1	12250000	8960	4	4	4	1	0	0	
2	12250000	9960	3	2	2	1	0	1	
3	12215000	7500	4	2	2	1	0	1	
4	11410000	7420	4	1	2	1	1	1	

Step 4: Splitting the Data into Training and Testing Sets

As you know, the first basic step for regression is performing a train-test split.

In [20]:

```
from sklearn.model_selection import train_test_split

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

Rescaling the Features

As you saw in the demonstration for Simple Linear Regression, scaling doesn't impact your model. Here we can see that except for `area`, all the columns have small integer values. So it is extremely important to rescale the variables so that they have a comparable scale. If we don't have comparable scales, then some of the coefficients as obtained by fitting the regression model might be very large or very small as compared to the other coefficients. This might become very annoying at the time of model evaluation. So it is advised to use standardization or normalization so that the units of the coefficients obtained are all on the same scale. As you know, there are two common ways of rescaling:

1. Min-Max scaling
2. Standardisation (mean-0, sigma-1)

This time, we will use MinMax scaling.

In [21]:

```
from sklearn.preprocessing import MinMaxScaler
```

In [22]:

```
scaler = MinMaxScaler()
```

In [23]:

```
# 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])
```

In [24]:

```
df_train.head()
```

Out[24]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hc
359	0.169697	0.155227	0.4	0.0	0.000000	1	0	0	
19	0.615152	0.403379	0.4	0.5	0.333333	1	0	0	
159	0.321212	0.115628	0.4	0.5	0.000000	1	1	1	
35	0.548133	0.454417	0.4	0.5	1.000000	1	0	0	
28	0.575758	0.538015	0.8	0.5	0.333333	1	0	1	

In [25]:

```
df_train.describe()
```

Out[25]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	b
count	381.000000	381.000000	381.000000	381.000000	381.000000	381.000000	381.000000	38
mean	0.260333	0.288710	0.386352	0.136483	0.268591	0.855643	0.170604	
std	0.157607	0.181420	0.147336	0.237325	0.295001	0.351913	0.376657	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.151515	0.155227	0.200000	0.000000	0.000000	1.000000	0.000000	
50%	0.221212	0.234424	0.400000	0.000000	0.333333	1.000000	0.000000	
75%	0.345455	0.398099	0.400000	0.500000	0.333333	1.000000	0.000000	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

In [27]:

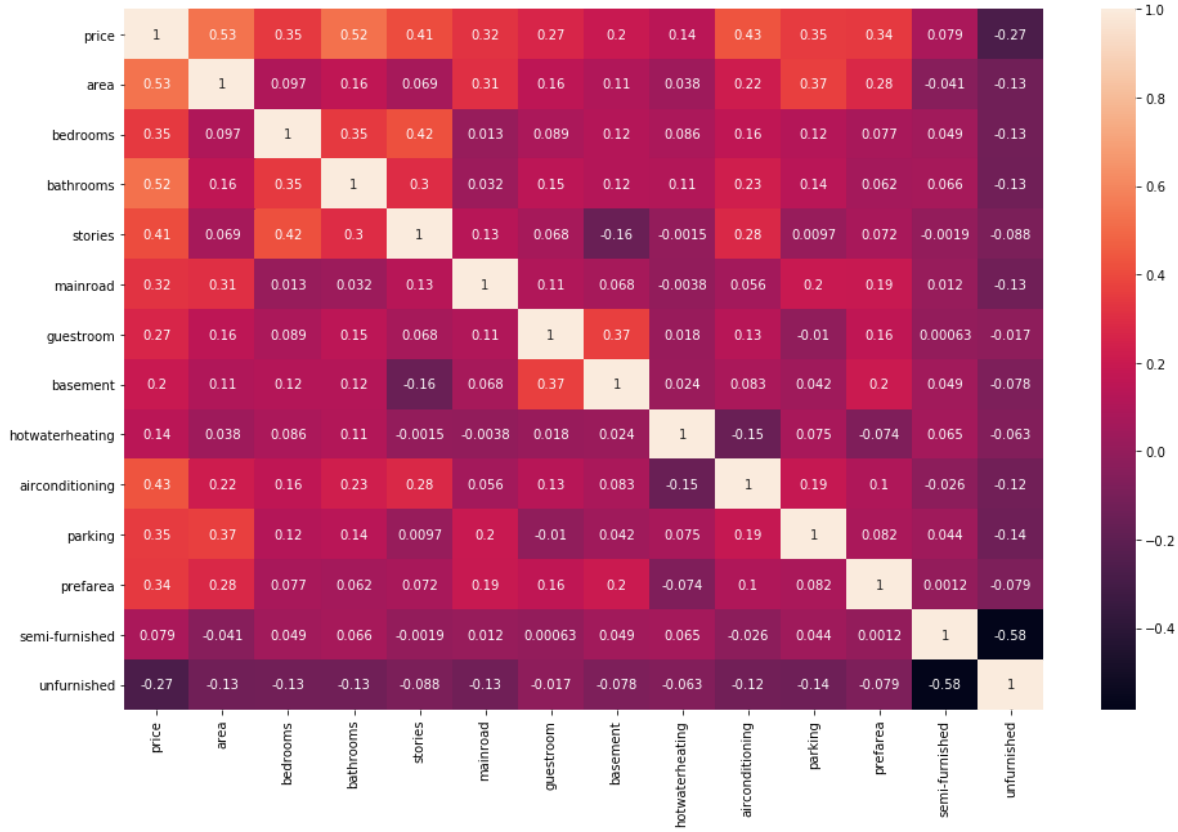
```
df_train.corr()
```

Out[27]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	ba
price	1.000000	0.532025	0.349825	0.524246	0.409464	0.319208	0.265877	0
area	0.532025	1.000000	0.097462	0.163446	0.069274	0.308272	0.155665	0
bedrooms	0.349825	0.097462	1.000000	0.346925	0.419582	0.012655	0.089488	0
bathrooms	0.524246	0.163446	0.346925	1.000000	0.295689	0.031716	0.150982	0
stories	0.409464	0.069274	0.419582	0.295689	1.000000	0.129427	0.068088	-0
mainroad	0.319208	0.308272	0.012655	0.031716	0.129427	1.000000	0.106875	0
guestroom	0.265877	0.155665	0.089488	0.150982	0.068088	0.106875	1.000000	0
basement	0.200743	0.106971	0.120618	0.120857	-0.155313	0.067937	0.367345	1
ıotwaterheating	0.141202	0.037719	0.085818	0.112732	-0.001538	-0.003779	0.018394	0
airconditioning	0.433162	0.215888	0.161890	0.228315	0.279779	0.056375	0.130300	0
parking	0.352081	0.365658	0.120363	0.138241	0.009708	0.200389	-0.010240	0
prefarea	0.344543	0.279878	0.076503	0.061584	0.072192	0.191465	0.161877	0
semi-furnished	0.078917	-0.040753	0.049111	0.065987	-0.001883	0.012254	0.000632	0
unfurnished	-0.270114	-0.131524	-0.125921	-0.128192	-0.088199	-0.129101	-0.017198	-0

In [26]:

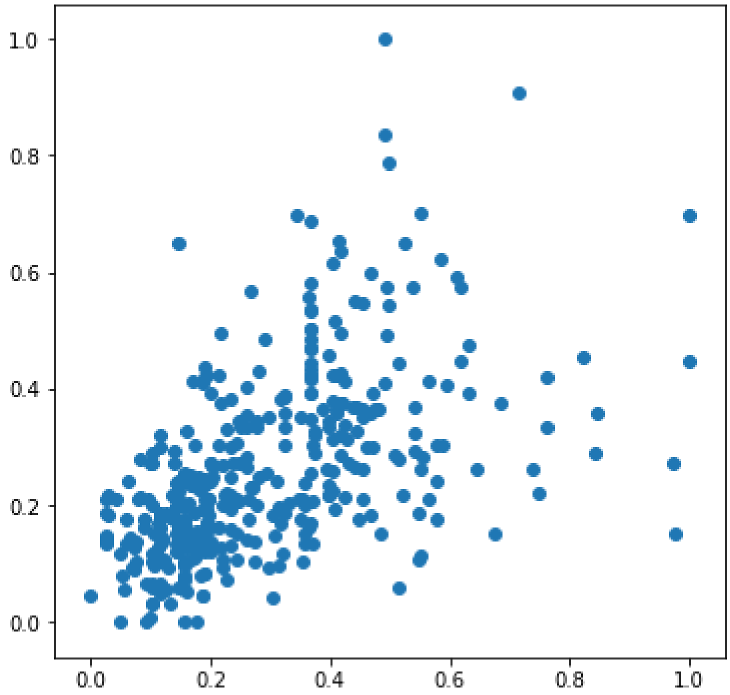
```
# Let's check the correlation coefficients to see which variables are highly correlated

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



As you might have noticed, `area` seems to the correlated to `price` the most. Let's see a pairplot for `area` vs `price`.

In [30]:

```
plt.figure(figsize=[6,6])
plt.scatter(df_train.area, df_train.price)
plt.show()
```



So, we pick `area` as the first variable and we'll try to fit a regression line to that.

Dividing into X and Y sets for the model building

In [28]:

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

In [30]:

```
print(y_train)
print(X_train)
```

```
359    0.169697
19     0.615152
159    0.321212
35     0.548133
28     0.575758
         ...
526    0.048485
53     0.484848
350    0.175758
79     0.424242
520    0.060606
Name: price, Length: 381, dtype: float64
         area  bedrooms  bathrooms   stories  mainroad  guestroom  basement  \
359  0.155227       0.4        0.0  0.000000         1          0         0
19   0.403379       0.4        0.5  0.333333         1          0         0
159  0.115628       0.4        0.5  0.000000         1          1         1
35   0.454417       0.4        0.5  1.000000         1          0         0
28   0.538015       0.8        0.5  0.333333         1          0         1
..        ...       ...        ...       ...       ...        ...       ...
526  0.118268       0.2        0.0  0.000000         1          0         0
53   0.291623       0.4        0.5  1.000000         1          0         0
350  0.139388       0.2        0.0  0.333333         1          0         0
79   0.366420       0.4        0.5  0.666667         1          1         0
520  0.516015       0.2        0.0  0.000000         1          0         0

     hotwaterheating  airconditioning   parking  prefarea  semi-furnished  \
359                0                0  0.333333         0               0
19                 0                1  0.333333         1               1
159                0                1  0.000000         0               0
35                 0                1  0.666667         0               0
28                 1                0  0.666667         0               0
..               ...              ...       ...       ...             ...
526                0                0  0.000000         0               0
53                 0                1  0.666667         0               1
350                1                0  0.333333         0               1
79                 0                1  0.000000         0               0
520                0                0  0.000000         0               0

     unfurnished
359            1
19             0
159            0
35             0
28             1
..           ...
526            1
53             0
350            0
79             0
520            1

[381 rows x 13 columns]
```

Step 5: Building a linear model

Fit a regression line through the training data using `statsmodels`. Remember that in `statsmodels`, you need to explicitly fit a constant using `sm.add_constant(X)` because if we don't perform this step, `statsmodels` fits a regression line passing through the origin, by default.

In [32]:

```
import statsmodels.api as sm

# Add a constant
X_train_lm = sm.add_constant(X_train[['area']])

# Create a first fitted model
lr = sm.OLS(y_train, X_train_lm).fit()
```

In [33]:

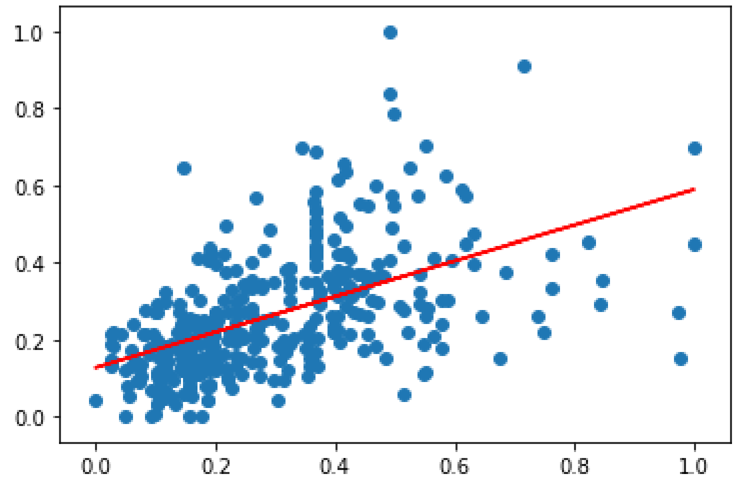
```
# Check the parameters obtained

lr.params
```

Out[33]:

```
const    0.126894
area     0.462192
dtype: float64
```

In [34]:

```
# Let's visualise the data with a scatter plot and the fitted regression line
plt.scatter(X_train_lm.iloc[:, 1], y_train)
plt.plot(X_train_lm.iloc[:, 1], 0.127 + 0.462*X_train_lm.iloc[:, 1], 'r')
plt.show()
```



In [36]:

```
X_train_lm
```

Out[36]:

	const	area
359	1.0	0.155227
19	1.0	0.403379
159	1.0	0.115628
35	1.0	0.454417
28	1.0	0.538015
...	...	...
526	1.0	0.118268
53	1.0	0.291623
350	1.0	0.139388
79	1.0	0.366420
520	1.0	0.516015

381 rows × 2 columns

In [35]:

```
X_train_lm.iloc[:, 1]
```

Out[35]:

```
359    0.155227
19     0.403379
159    0.115628
35     0.454417
28     0.538015
         ...   
526    0.118268
53     0.291623
350    0.139388
79     0.366420
520    0.516015
Name: area, Length: 381, dtype: float64
```

In [33]:

```
# Print a summary of the linear regression model obtained
print(lr.summary())
```

```
                            OLS Regression Results
============================================================================
==
Dep. Variable:                  price   R-squared:                       0.2
83
Model:                            OLS   Adj. R-squared:                  0.2
81
Method:                 Least Squares   F-statistic:                      14
9.6
Date:                Tue, 09 Oct 2018   Prob (F-statistic):           3.15e-
29
Time:                        13:02:46   Log-Likelihood:                 227.
23
No. Observations:                 381   AIC:                            -45
0.5
Df Residuals:                     379   BIC:                            -44
2.6
Df Model:                           1
Covariance Type:            nonrobust
============================================================================
==
                 coef    std err          t      P>|t|      [0.025      0.97
5]
----------------------------------------------------------------------------
--
const          0.1269      0.013      9.853      0.000       0.102       0.1
52
area           0.4622      0.038     12.232      0.000       0.388       0.5
36
============================================================================
==
Omnibus:                       67.313   Durbin-Watson:                   2.0
18
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              143.0
63
Skew:                           0.925   Prob(JB):                     8.59e-
32
Kurtosis:                       5.365   Cond. No.                         5.
99
============================================================================
==

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is corre
ctly specified.
```

Adding another variable

The R-squared value obtained is `0.283` . Since we have so many variables, we can clearly do better than this. So let's go ahead and add the second most highly correlated variable, i.e. `bathrooms` .

In [34]:

```
# Assign all the feature variables to X
X_train_lm = X_train[['area', 'bathrooms']]
```

In [35]:

```
# Build a linear model

import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train_lm)

lr = sm.OLS(y_train, X_train_lm).fit()

lr.params
```

Out[35]:

```
const        0.104589
area         0.398396
bathrooms    0.298374
dtype: float64
```

In [36]:

```
# Check the summary
print(lr.summary())
```

```
                            OLS Regression Results
==============================================================================
==
Dep. Variable:                  price   R-squared:                       0.4
80
Model:                            OLS   Adj. R-squared:                  0.4
77
Method:                 Least Squares   F-statistic:                     17
4.1
Date:                Tue, 09 Oct 2018   Prob (F-statistic):           2.51e-
54
Time:                        13:02:47   Log-Likelihood:                 288.
24
No. Observations:                 381   AIC:                            -57
0.5
Df Residuals:                     378   BIC:                            -55
8.6
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
==
                 coef    std err          t      P>|t|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
const          0.1046      0.011      9.384      0.000       0.083       0.1
27
area           0.3984      0.033     12.192      0.000       0.334       0.4
63
bathrooms      0.2984      0.025     11.945      0.000       0.249       0.3
47
==============================================================================
==
Omnibus:                       62.839   Durbin-Watson:                   2.1
57
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              168.7
90
Skew:                           0.784   Prob(JB):                     2.23e-
37
Kurtosis:                       5.859   Cond. No.                         6.
17
==============================================================================
==

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is corre
ctly specified.
```

We have clearly improved the model as the value of adjusted R-squared as its value has gone up to `0.477` from `0.281` . Let's go ahead and add another variable, `bedrooms` .

In [37]:

```
# Assign all the feature variables to X
X_train_lm = X_train[['area', 'bathrooms','bedrooms']]
```

In [38]:

```
# Build a linear model

import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train_lm)

lr = sm.OLS(y_train, X_train_lm).fit()

lr.params
```

Out[38]:

```
const        0.041352
area         0.392211
bathrooms    0.259978
bedrooms     0.181863
dtype: float64
```

In [39]:

```
# Print the summary of the model

print(lr.summary())
```

```
                            OLS Regression Results
==============================================================================
==
Dep. Variable:                  price   R-squared:                       0.5
05
Model:                            OLS   Adj. R-squared:                  0.5
01
Method:                 Least Squares   F-statistic:                      12
8.2
Date:                Tue, 09 Oct 2018   Prob (F-statistic):           3.12e-
57
Time:                        13:02:47   Log-Likelihood:                 297.
76
No. Observations:                 381   AIC:                            -58
7.5
Df Residuals:                     377   BIC:                            -57
1.7
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
==
                 coef    std err          t      P>|t|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
const          0.0414      0.018      2.292      0.022       0.006       0.0
77
area           0.3922      0.032     12.279      0.000       0.329       0.4
55
bathrooms      0.2600      0.026     10.033      0.000       0.209       0.3
11
bedrooms       0.1819      0.041      4.396      0.000       0.101       0.2
63
==============================================================================
==
Omnibus:                       50.037   Durbin-Watson:                   2.1
36
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              124.8
06
Skew:                           0.648   Prob(JB):                     7.92e-
28
Kurtosis:                       5.487   Cond. No.                         8.
87
==============================================================================
==

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is corre
ctly specified.
```

We have improved the adjusted R-squared again. Now let's go ahead and add all the feature variables.

Adding all the variables to the model

In [37]:

```
# Check all the columns of the dataframe

housing.columns
```

Out[37]:

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

In [38]:

```
#Build a linear model

import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train)

lr_1 = sm.OLS(y_train, X_train_lm).fit()

lr_1.params
```

Out[38]:

```
const              0.020033
area               0.234664
bedrooms           0.046735
bathrooms          0.190823
stories            0.108516
mainroad           0.050441
guestroom          0.030428
basement           0.021595
hotwaterheating    0.084863
airconditioning    0.066881
parking            0.060735
prefarea           0.059428
semi-furnished     0.000921
unfurnished       -0.031006
dtype: float64
```

In [39]:

```
print(lr_1.summary())
```

```
                            OLS Regression Results
==============================================================================
==
Dep. Variable:                  price   R-squared:                       0.6
81
Model:                            OLS   Adj. R-squared:                  0.6
70
Method:                 Least Squares   F-statistic:                     60.
40
Date:                Tue, 14 Dec 2021   Prob (F-statistic):           8.83e-
83
Time:                        01:12:18   Log-Likelihood:                 381.
79
No. Observations:                 381   AIC:                            -73
5.6
Df Residuals:                     367   BIC:                            -68
0.4
Df Model:                          13
Covariance Type:            nonrobust
==============================================================================
=======
                      coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-------
const               0.0200      0.021      0.955      0.340      -0.021
0.061
area                0.2347      0.030      7.795      0.000       0.175
0.294
bedrooms            0.0467      0.037      1.267      0.206      -0.026
0.119
bathrooms           0.1908      0.022      8.679      0.000       0.148
0.234
stories             0.1085      0.019      5.661      0.000       0.071
0.146
mainroad            0.0504      0.014      3.520      0.000       0.022
0.079
guestroom           0.0304      0.014      2.233      0.026       0.004
0.057
basement            0.0216      0.011      1.943      0.053      -0.000
0.043
hotwaterheating     0.0849      0.022      3.934      0.000       0.042
0.127
airconditioning     0.0669      0.011      5.899      0.000       0.045
0.089
parking             0.0607      0.018      3.365      0.001       0.025
0.096
prefarea            0.0594      0.012      5.040      0.000       0.036
0.083
semi-furnished      0.0009      0.012      0.078      0.938      -0.022
0.024
unfurnished        -0.0310      0.013     -2.440      0.015      -0.056
-0.006
==============================================================================
==
Omnibus:                       93.687   Durbin-Watson:                   2.0
93
```

```
Prob(Omnibus):                0.000   Jarque-Bera (JB):              304.9
17
Skew:                         1.091   Prob(JB):                     6.14e-
67
Kurtosis:                     6.801   Cond. No.                         1
4.6
============================================================================
==

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is corre
ctly specified.
```

Looking at the p-values, it looks like some of the variables aren't really significant (in the presence of other variables).

Maybe we could drop some?

We could simply drop the variable with the highest, non-significant p value. A better way would be to supplement this with the VIF information.

Checking VIF

Variance Inflation Factor or VIF, gives a basic quantitative idea about how much the feature variables are correlated with each other. It is an extremely important parameter to test our linear model. The formula for calculating `VIF` is:

$$VIF_i = \frac{1}{1 - {R_i}^2}$$

In [40]:

```
# Check for the VIF values of the feature variables.
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [41]:

```
# Create a dataframe that will contain the names of all the feature variables and their res
vif = pd.DataFrame()
vif['Features'] = X_train.columns
vif['VIF'] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.shape[1])
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[41]:

	Features	VIF
1	bedrooms	7.33
4	mainroad	6.02
0	area	4.67
3	stories	2.70
11	semi-furnished	2.19
9	parking	2.12
6	basement	2.02
12	unfurnished	1.82
8	airconditioning	1.77
2	bathrooms	1.67
10	prefarea	1.51
5	guestroom	1.47
7	hotwaterheating	1.14

In [43]:

```
X_train.shape[1]
```

Out[43]:

```
13
```

In [42]:

```
[ i for i in range(X_train.shape[1])]
```

Out[42]:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

In [44]:

```
[variance_inflation_factor(X_train.values, i) for i in range(X_train.shape[1])] # coloum va
```

Out[44]:

```
[4.672431227793009,
 7.334389784233085,
 1.6665120091225607,
 2.701091923665566,
 6.021773605401158,
 1.4691060005063157,
 2.015021565600942,
 1.1358604562890016,
 1.7708466990969685,
 2.122672173883017,
 1.5071065114121442,
 2.1881722464928153,
 1.8227179542187855]
```

In [ ]:

```

```

We generally want a VIF that is less than 5. So there are clearly some variables we need to drop.

Dropping the variable and updating the model

As you can see from the summary and the VIF dataframe, some variables are still insignificant. One of these variables is, `semi-furnished` as it has a very high p-value of `0.938` . Let's go ahead and drop this variables

In [45]:

```
# Dropping highly correlated variables and insignificant variables

X = X_train.drop('semi-furnished', 1,)
```

In [49]:

```
X
```

Out[49]:

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheat
359	0.155227	0.4	0.0	0.000000	1	0	0	
19	0.403379	0.4	0.5	0.333333	1	0	0	
159	0.115628	0.4	0.5	0.000000	1	1	1	
35	0.454417	0.4	0.5	1.000000	1	0	0	
28	0.538015	0.8	0.5	0.333333	1	0	1	
...	...	...	...	...	...	...	...	
526	0.118268	0.2	0.0	0.000000	1	0	0	
53	0.291623	0.4	0.5	1.000000	1	0	0	
350	0.139388	0.2	0.0	0.333333	1	0	0	
79	0.366420	0.4	0.5	0.666667	1	1	0	
520	0.516015	0.2	0.0	0.000000	1	0	0	

381 rows × 12 columns

In [46]:

```
# Build a third fitted model
X_train_lm = sm.add_constant(X)

lr_2 = sm.OLS(y_train, X_train_lm).fit()
```

In [47]:

```
# Print the summary of the model
print(lr_2.summary())
```

```
                            OLS Regression Results
============================================================================
==
Dep. Variable:                  price   R-squared:                       0.6
81
Model:                            OLS   Adj. R-squared:                  0.6
71
Method:                 Least Squares   F-statistic:                     65.
61
Date:                Tue, 09 Oct 2018   Prob (F-statistic):           1.07e-
83
Time:                        13:02:48   Log-Likelihood:                 381.
79
No. Observations:                 381   AIC:                            -73
7.6
Df Residuals:                     368   BIC:                            -68
6.3
Df Model:                          12
Covariance Type:            nonrobust
============================================================================
=======
                      coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------------
-------
const               0.0207      0.019      1.098      0.273      -0.016
0.058
area                0.2344      0.030      7.845      0.000       0.176
0.293
bedrooms            0.0467      0.037      1.268      0.206      -0.026
0.119
bathrooms           0.1909      0.022      8.697      0.000       0.148
0.234
stories             0.1085      0.019      5.669      0.000       0.071
0.146
mainroad            0.0504      0.014      3.524      0.000       0.022
0.079
guestroom           0.0304      0.014      2.238      0.026       0.004
0.057
basement            0.0216      0.011      1.946      0.052      -0.000
0.043
hotwaterheating     0.0849      0.022      3.941      0.000       0.043
0.127
airconditioning     0.0668      0.011      5.923      0.000       0.045
0.089
parking             0.0608      0.018      3.372      0.001       0.025
0.096
prefarea            0.0594      0.012      5.046      0.000       0.036
0.083
unfurnished        -0.0316      0.010     -3.096      0.002      -0.052
-0.012
============================================================================
==
Omnibus:                       93.538   Durbin-Watson:                   2.0
92
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              303.8
```

```
44
Skew:                           1.090   Prob(JB):                    1.05e-
66
Kurtosis:                       6.794   Cond. No.                        1
4.1
============================================================================
==

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is corre
ctly specified.
```

In [48]:

```
# Calculate the VIFs again for the new model

vif = pd.DataFrame()
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
```

Out[48]:

	Features	VIF
1	bedrooms	6.59
4	mainroad	5.68
0	area	4.67
3	stories	2.69
9	parking	2.12
6	basement	2.01
8	airconditioning	1.77
2	bathrooms	1.67
10	prefarea	1.51
5	guestroom	1.47
11	unfurnished	1.40
7	hotwaterheating	1.14

Dropping the Variable and Updating the Model

As you can notice some of the variable have high VIF values as well as high p-values. Such variables are insignificant and should be dropped.

As you might have noticed, the variable `bedroom` has a significantly high VIF (`6.6`) and a high p-value (`0.206`) as well. Hence, this variable isn't of much use and should be dropped.

In [49]:

```
# Dropping highly correlated variables and insignificant variables
X = X.drop('bedrooms', 1)
```

In [50]:

```
# Build a second fitted model
X_train_lm = sm.add_constant(X)

lr_3 = sm.OLS(y_train, X_train_lm).fit()
```

In [51]:

```
# Print the summary of the model

print(lr_3.summary())
```

```
                            OLS Regression Results
============================================================================
==
Dep. Variable:                  price   R-squared:                       0.6
80
Model:                            OLS   Adj. R-squared:                  0.6
71
Method:                 Least Squares   F-statistic:                     71.
31
Date:                Tue, 09 Oct 2018   Prob (F-statistic):           2.73e-
84
Time:                        13:02:48   Log-Likelihood:                 380.
96
No. Observations:                 381   AIC:                            -73
7.9
Df Residuals:                     369   BIC:                            -69
0.6
Df Model:                          11
Covariance Type:            nonrobust
============================================================================
=======
                      coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------------
-------
const               0.0357      0.015      2.421      0.016       0.007
0.065
area                0.2347      0.030      7.851      0.000       0.176
0.294
bathrooms           0.1965      0.022      9.132      0.000       0.154
0.239
stories             0.1178      0.018      6.654      0.000       0.083
0.153
mainroad            0.0488      0.014      3.423      0.001       0.021
0.077
guestroom           0.0301      0.014      2.211      0.028       0.003
0.057
basement            0.0239      0.011      2.183      0.030       0.002
0.045
hotwaterheating     0.0864      0.022      4.014      0.000       0.044
0.129
airconditioning     0.0665      0.011      5.895      0.000       0.044
0.089
parking             0.0629      0.018      3.501      0.001       0.028
0.098
prefarea            0.0596      0.012      5.061      0.000       0.036
0.083
unfurnished        -0.0323      0.010     -3.169      0.002      -0.052
-0.012
============================================================================
==
Omnibus:                       97.661   Durbin-Watson:                   2.0
97
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              325.3
88
```

```
Skew:                           1.130   Prob(JB):                    2.20e-
71
Kurtosis:                       6.923   Cond. No.                        1
0.6
============================================================================
==

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is corre
ctly specified.
```

In [52]:

```python
# Calculate the VIFs again for the new model
vif = pd.DataFrame()
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
```

Out[52]:

	Features	VIF
3	mainroad	4.79
0	area	4.55
2	stories	2.23
8	parking	2.10
5	basement	1.87
7	airconditioning	1.76
1	bathrooms	1.61
9	prefarea	1.50
4	guestroom	1.46
10	unfurnished	1.33
6	hotwaterheating	1.12

Dropping the variable and updating the model

As you might have noticed, dropping `semi-furnised` decreased the VIF of `mainroad` as well such that it is now under 5. But from the summary, we can still see some of them have a high p-value. `basement` for instance, has a p-value of 0.03. We should drop this variable as well.

In [53]:

```python
X = X.drop('basement', 1)
```

In [54]:

```
# Build a fourth fitted model
X_train_lm = sm.add_constant(X)

lr_4 = sm.OLS(y_train, X_train_lm).fit()
```

In [55]:

```
print(lr_4.summary())
```

```
                            OLS Regression Results                            
==============================================================================
==
Dep. Variable:                  price   R-squared:                       0.6
76
Model:                            OLS   Adj. R-squared:                  0.6
67
Method:                 Least Squares   F-statistic:                     77.
18
Date:                Tue, 09 Oct 2018   Prob (F-statistic):           3.13e-
84
Time:                        13:02:49   Log-Likelihood:                 378.
51
No. Observations:                 381   AIC:                            -73
5.0
Df Residuals:                     370   BIC:                            -69
1.7
Df Model:                          10                                         
Covariance Type:            nonrobust                                         
==============================================================================
=======
                      coef    std err          t      P>|t|      [0.025      
0.975]
------------------------------------------------------------------------------
-------
const               0.0428      0.014      2.958      0.003       0.014      
0.071
area                0.2335      0.030      7.772      0.000       0.174      
0.293
bathrooms           0.2019      0.021      9.397      0.000       0.160      
0.244
stories             0.1081      0.017      6.277      0.000       0.074      
0.142
mainroad            0.0497      0.014      3.468      0.001       0.022      
0.078
guestroom           0.0402      0.013      3.124      0.002       0.015      
0.065
hotwaterheating     0.0876      0.022      4.051      0.000       0.045      
0.130
airconditioning     0.0682      0.011      6.028      0.000       0.046      
0.090
parking             0.0629      0.018      3.482      0.001       0.027      
0.098
prefarea            0.0637      0.012      5.452      0.000       0.041      
0.087
unfurnished        -0.0337      0.010     -3.295      0.001      -0.054      
-0.014
==============================================================================
==
Omnibus:                       97.054   Durbin-Watson:                   2.0
99
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              322.0
34
Skew:                           1.124   Prob(JB):                     1.18e-
70
Kurtosis:                       6.902   Cond. No.                         1
0.3
```

```
============================================================================
==

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is corre
ctly specified.
```

In [56]:

```
# Calculate the VIFs again for the new model
vif = pd.DataFrame()
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
```

Out[56]:

	Features	VIF
3	mainroad	4.55
0	area	4.54
2	stories	2.12
7	parking	2.10
6	airconditioning	1.75
1	bathrooms	1.58
8	prefarea	1.47
9	unfurnished	1.33
4	guestroom	1.30
5	hotwaterheating	1.12

Now as you can see, the VIFs and p-values both are within an acceptable range. So we go ahead and make our predictions using this model only.

Step 7: Residual Analysis of the train data

So, now to check if the error terms are also normally distributed (which is infact, one of the major assumptions of linear regression), let us plot the histogram of the error terms and see what it looks like.

In [57]:

```
y_train_price = lr_4.predict(X_train_lm)
```

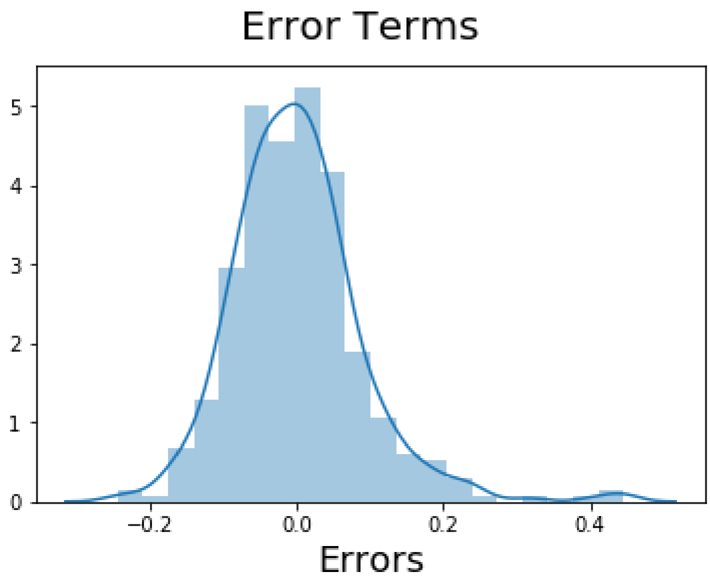
In [58]:

```
# 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
```

```
C:\Users\admin\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: Us
erWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'd
ensity' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

Out[58]:

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



Step 8: Making Predictions Using the Final Model

Now that we have fitted the model and checked the normality of error terms, it's time to go ahead and make predictions using the final, i.e. fourth model.

Applying the scaling on the test sets

In [59]:

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

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

In [60]:

```
df_test.describe()
```

Out[60]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	b
count	164.000000	164.000000	164.000000	164.000000	164.000000	164.000000	164.000000	16
mean	0.263176	0.298548	0.408537	0.158537	0.268293	0.865854	0.195122	
std	0.172077	0.211922	0.147537	0.281081	0.276007	0.341853	0.397508	
min	0.006061	-0.016367	0.200000	0.000000	0.000000	0.000000	0.000000	
25%	0.142424	0.148011	0.400000	0.000000	0.000000	1.000000	0.000000	
50%	0.226061	0.259724	0.400000	0.000000	0.333333	1.000000	0.000000	
75%	0.346970	0.397439	0.400000	0.500000	0.333333	1.000000	0.000000	
max	0.909091	1.263992	0.800000	1.500000	1.000000	1.000000	1.000000	

Dividing into X_test and y_test

In [61]:

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

In [62]:

```
# Adding constant variable to test dataframe
X_test_m4 = sm.add_constant(X_test)
```

In [63]:

```
# Creating X_test_m4 dataframe by dropping variables from X_test_m4

X_test_m4 = X_test_m4.drop(["bedrooms", "semi-furnished", "basement"], axis = 1)
```

In [64]:

```
# Making predictions using the fourth model

y_pred_m4 = lr_4.predict(X_test_m4)
```

Step 9: Model Evaluation

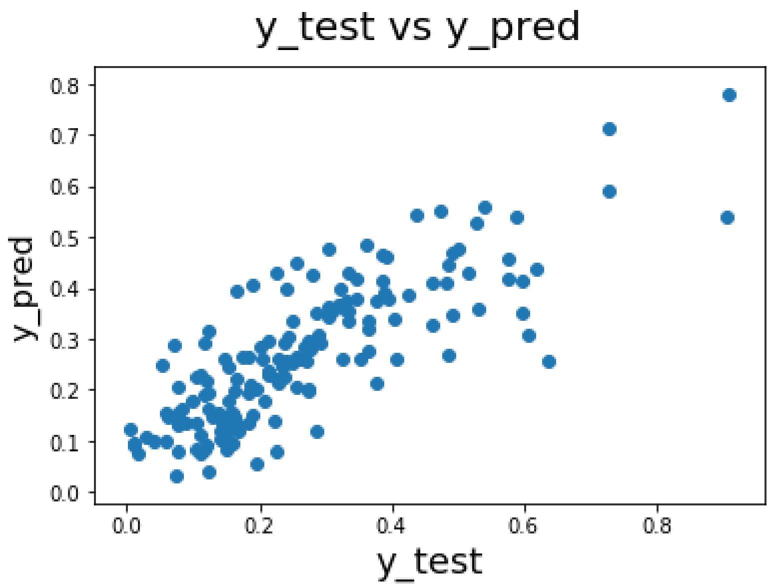
Let's now plot the graph for actual versus predicted values.

In [65]:

```
# Plotting y_test and y_pred to understand the spread

fig = plt.figure()
plt.scatter(y_test, y_pred_m4)
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)
```

Out[65]:

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



We can see that the equation of our best fitted line is:

$$price = 0.236 \times area + 0.202 \times bathrooms + 0.11 \times stories + 0.05 \times mainroad + 0.04 \times guestroom +$$
$$\times airconditioning + 0.0629 \times parking + 0.0637 \times prefarea - 0.0337 \times unfurnished$$

Overall we have a decent model, but we also acknowledge that we could do better.

We have a couple of options:

1. Add new features (bathrooms/bedrooms, area/stories, etc.)
2. Build a non-linear model