Topic: Real Estate Price Prediction

Brief Background

Real estate is not in range recently, and they are driven by unscientific and illogical terms. People with more money always consider some factors while considering real estate, these factors are house age, distance to the nearest stations and number of stores around. This dataset gives us a chance to investigate the data on what really influence the value of a house considering the listed factors.

Outline of the Focus of the Analysis

Data Source

In [ ]:
```
import pandas as pd
import numpy as np
```

In [ ]:
```
#Step 1: Import Data from Source Link
#Step 1: Import Data from Source Link
path= 'Real estate.csv'
df = pd.read_csv(path)
df.head()
```

Out[ ]:

	No	X1 transaction date	X2 house age	X3 distance to the nearest MRT station	X4 number of convenience stores	X5 latitude	X6 longitude	Y house price of unit area
0	1	2012.917	32.0	84.87882	10	24.98298	121.54024	37.9
1	2	2012.917	19.5	306.59470	9	24.98034	121.53951	42.2
2	3	2013.583	13.3	561.98450	5	24.98746	121.54391	47.3
3	4	2013.500	13.3	561.98450	5	24.98746	121.54391	54.8
4	5	2012.833	5.0	390.56840	5	24.97937	121.54245	43.1

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

Out[ ]:

	No	X1 transaction date	X2 house age	X3 distance to the nearest MRT station	X4 number of convenience stores	X5 latitude	X6 longitude	Y house price of unit area
410	411	2012.667	5.6	90.45606	9	24.97433	121.54310	50.0
411	412	2013.250	18.8	390.96960	7	24.97923	121.53986	40.6
412	413	2013.000	8.1	104.81010	5	24.96674	121.54067	52.5
413	414	2013.500	6.5	90.45606	9	24.97433	121.54310	63.9

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 414 entries, 0 to 413
Data columns (total 8 columns):
 #   Column                                  Non-Null Count  Dtype  
---  ------                                  --------------  -----  
 0   No                                      414 non-null    int64  
 1   X1 transaction date                     414 non-null    float64
 2   X2 house age                            414 non-null    float64
 3   X3 distance to the nearest MRT station  414 non-null    float64
 4   X4 number of convenience stores         414 non-null    int64  
 5   X5 latitude                             414 non-null    float64
 6   X6 longitude                            414 non-null    float64
 7   Y house price of unit area              414 non-null    float64
dtypes: float64(6), int64(2)
memory usage: 26.0 KB
```

In [ ]:
```
#Renanme each column to at least have an uinderscore (_)
col_names=df.columns
col_names=col_names.str.replace(" ","_")
df.columns=col_names
```

In [ ]:
```
##Checking for missing varaibles
df.isnull().sum()
```

Out[ ]:
```
No                                      0
X1_transaction_date                     0
X2_house_age                            0
X3_distance_to_the_nearest_MRT_station  0
X4_number_of_convenience_stores         0
X5_latitude                             0
X6_longitude                            0
Y_house_price_of_unit_area              0
dtype: int64
```

Key Descriptive Statistics with interpretation

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

Out[ ]:

	No	X1_transaction_date	X2_house_age	X3_distance_to_the_nearest_MRT_station	X4_number_of_convenience_stores	X5_latitude
count	414.000000	414.000000	414.000000	414.000000	414.000000	414.000000
mean	207.500000	2013.148971	17.712560	1083.885689	4.094203	24.969030
std	119.655756	0.281967	11.392485	1262.109595	2.945562	0.012410
min	1.000000	2012.667000	0.000000	23.382840	0.000000	24.932070
25%	104.250000	2012.917000	9.025000	289.324800	1.000000	24.963000
50%	207.500000	2013.167000	16.100000	492.231300	4.000000	24.971100
75%	310.750000	2013.417000	28.150000	1454.279000	6.000000	24.977455
max	414.000000	2013.583000	43.800000	6488.021000	10.000000	25.014590

In [ ]:
```
#Drop insignificant variable
df.drop(['No'], axis = 1, inplace=True)
df.head(2)
```

Out[ ]:

	X1_transaction_date	X2_house_age	X3_distance_to_the_nearest_MRT_station	X4_number_of_convenience_stores	X5_latitude	X6_longitude	Y
0	2012.917	32.0	84.87882	10	24.98298	121.54024	
1	2012.917	19.5	306.59470	9	24.98034	121.53951	

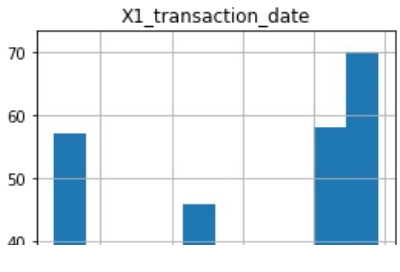
Graphical analysis with interpretation

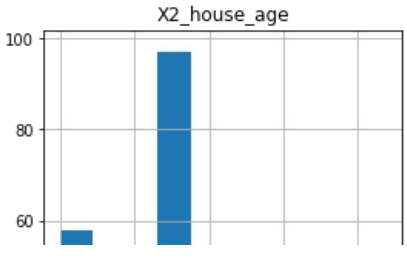
In [ ]:
```
%%capture
# ! pip install seaborn
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

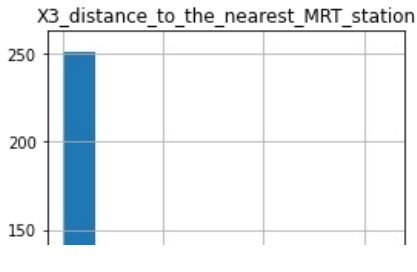
Data Visualization of each Independent Variables and Dependent Variable

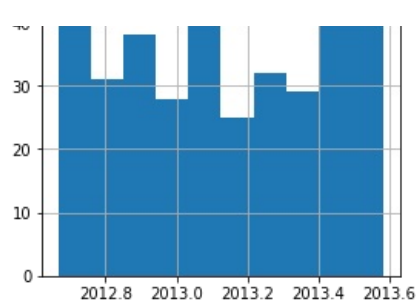
In [ ]:
```
df.hist(figsize = (15,20))
```

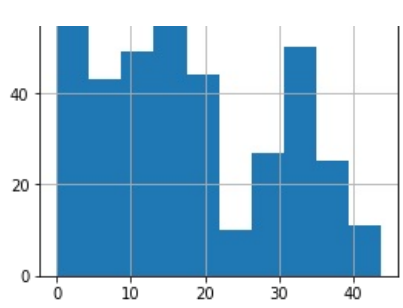
Out[ ]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f8acbe54750>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8acbcb02d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8acbce53d0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f8acbc9a890>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8acbc50e10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8acbc123d0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f8acbbc59d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8acbb7ce90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8acbb7ced0>]],
      dtype=object)
```

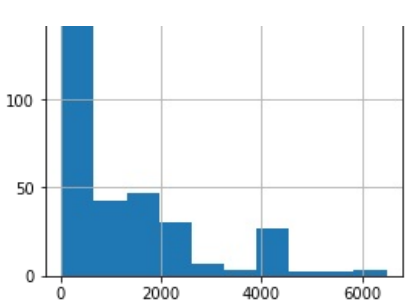


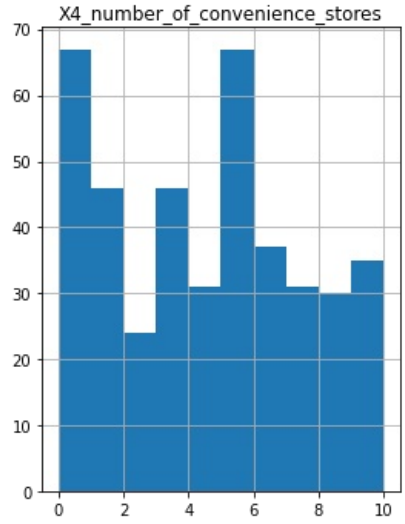


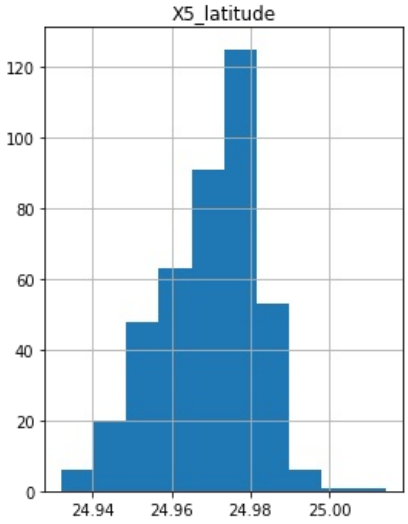


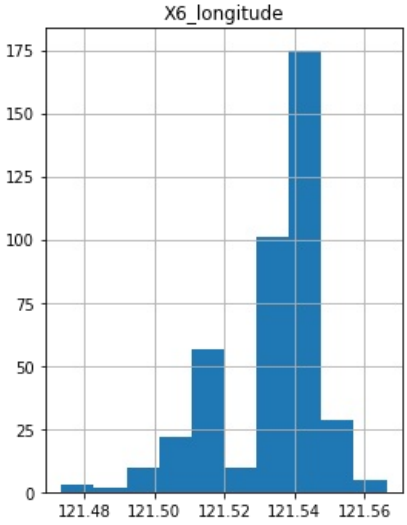


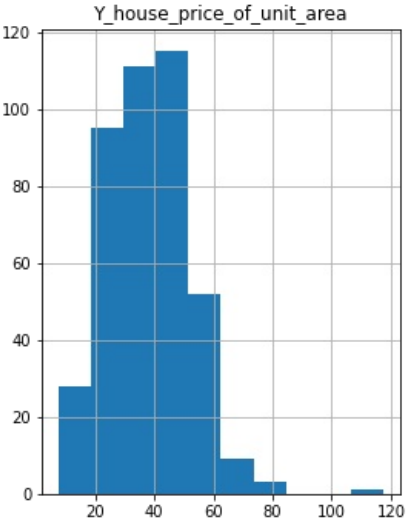












Correlation between Independent Variables and Dependent Variable

In [ ]:
```
df_corr =df.corr()
#df_corr on a Table
df_corr
```

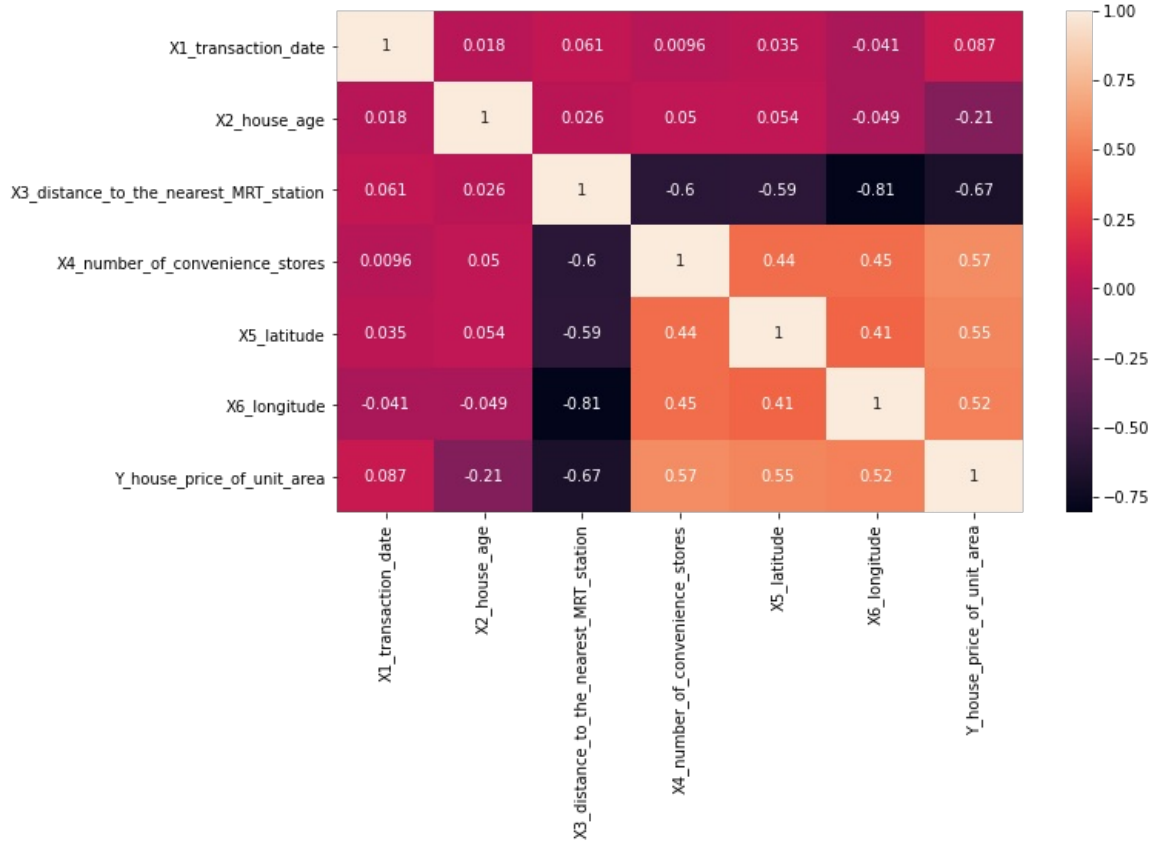
Out[ ]:

	X1_transaction_date	X2_house_age	X3_distance_to_the_nearest_MRT_station	X4_number_of_convenienc
X1_transaction_date	1.000000	0.017549	0.060880	
X2_house_age	0.017549	1.000000	0.025622	
X3_distance_to_the_nearest_MRT_station	0.060880	0.025622	1.000000	-
X4_number_of_convenience_stores	0.009635	0.049593	-0.602519	
X5_latitude	0.035058	0.054420	-0.591067	
X6_longitude	-0.041082	-0.048520	-0.806317	
Y_house_price_of_unit_area	0.087491	-0.210567	-0.673613	

In [ ]:
```
#df_corr with heatmap
```

```
plt.figure(figsize=(10,6))
sns.heatmap(df_corr,annot=True)
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8acbaaf850>



In [ ]:
```
#Looks like 'X1 transaction date' is not correlated to the price, so, let drop it.
df.drop(['X1_transaction_date'], axis = 1, inplace=True)
df.head(2)
```

Out[ ]:

	X2_house_age	X3_distance_to_the_nearest_MRT_station	X4_number_of_convenience_stores	X5_latitude	X6_longitude	Y_house_price_of_unit_
0	32.0	84.87882	10	24.98298	121.54024	
1	19.5	306.59470	9	24.98034	121.53951	

Relationship between Independent Variables and Dependent Variable and Checking for Outliers

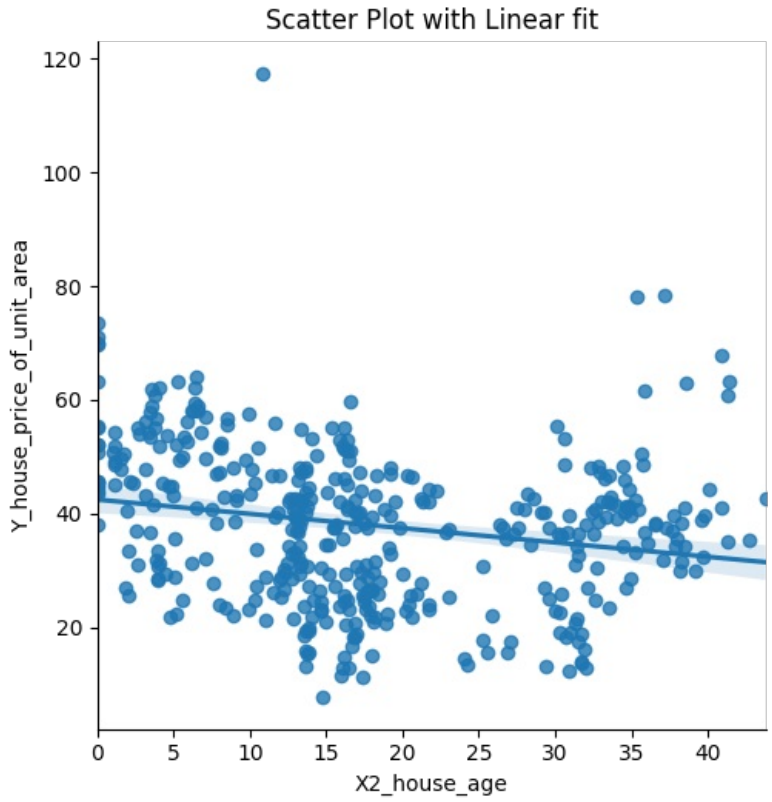
In [ ]:
```
#Function to display a scatter plot
def show_graph (X, Y, data):
  plt.rcParams.update({'figure.figsize':(10,8), 'figure.dpi':100})
  sns.lmplot(x=X, y=Y, data=data)
  ade = plt.title("Scatter Plot with Linear fit")
  return ade
```

In [ ]:
```
#Function to detect and remove outliers
def outlier_detector (variable):
  # IQR
  Q1 = np.percentile(variable, 25,
                     interpolation = 'midpoint')

  Q3 = np.percentile(variable, 75,
                     interpolation = 'midpoint')
  IQR = Q3 - Q1

  # Upper bound
  upper = np.where(variable >= (Q3+1.5*IQR))
  # Lower bound
  lower = np.where(variable <= (Q1-1.5*IQR))

  #Removing the Outliers
  df.drop(upper[0], inplace = True)
  df.drop(lower[0], inplace = True)
  return
```

In [ ]:
```
# For X2 house age
show_graph (X= "X2_house_age", Y="Y_house_price_of_unit_area", data= df)
```

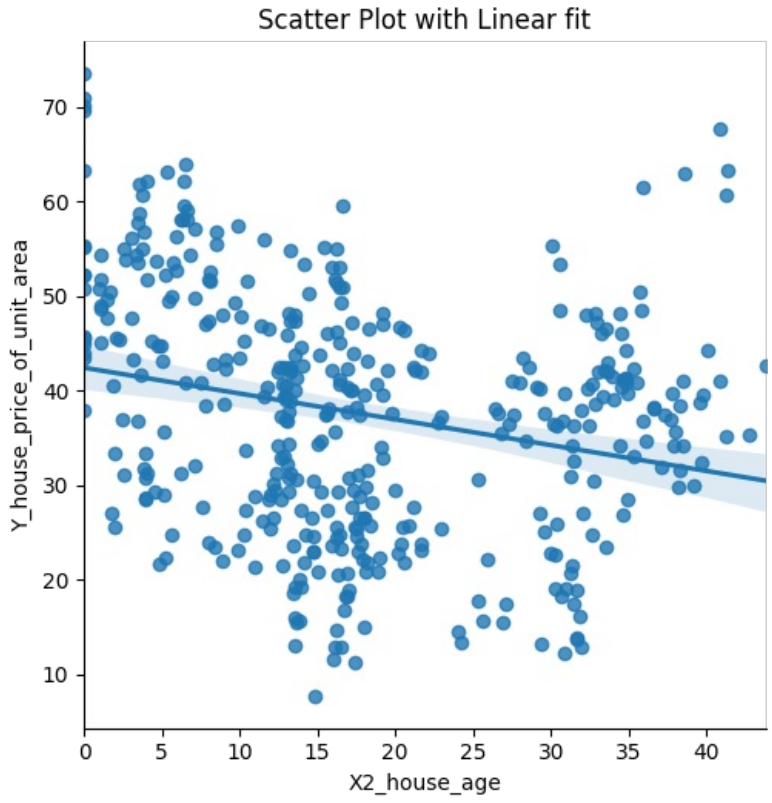
```
#Result below shows good relationship but one outlier on dependent variable.
```

Out[ ]: Text(0.5, 1.0, 'Scatter Plot with Linear fit')



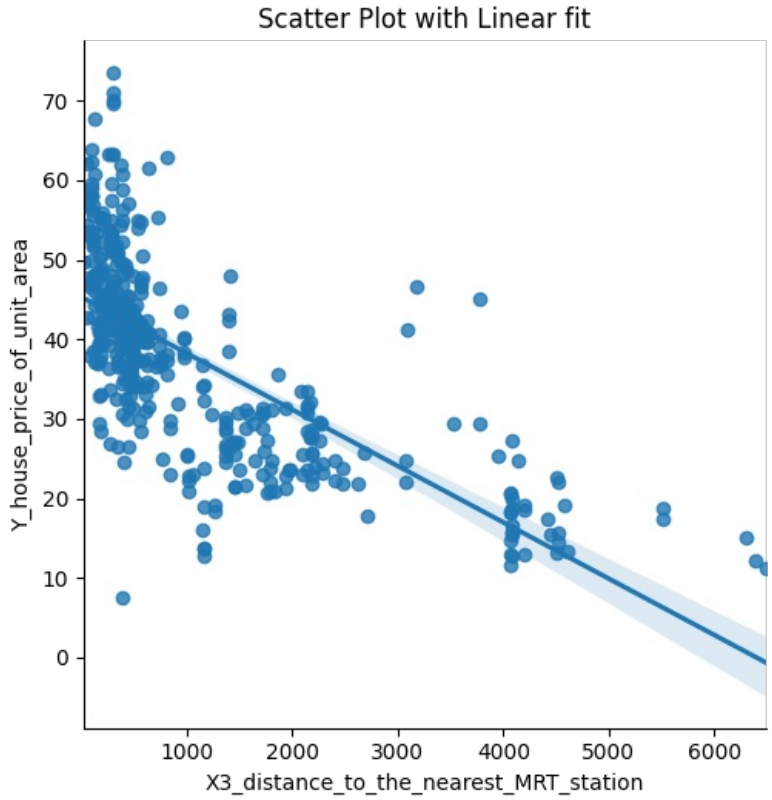
In [ ]:
```
#Remove outlier on dependent variable
outlier_detector (df.Y_house_price_of_unit_area)
show_graph (X= "X2_house_age", Y="Y_house_price_of_unit_area", data= df)
# Result below shows good relationship with no more outliers
```

Out[ ]: Text(0.5, 1.0, 'Scatter Plot with Linear fit')



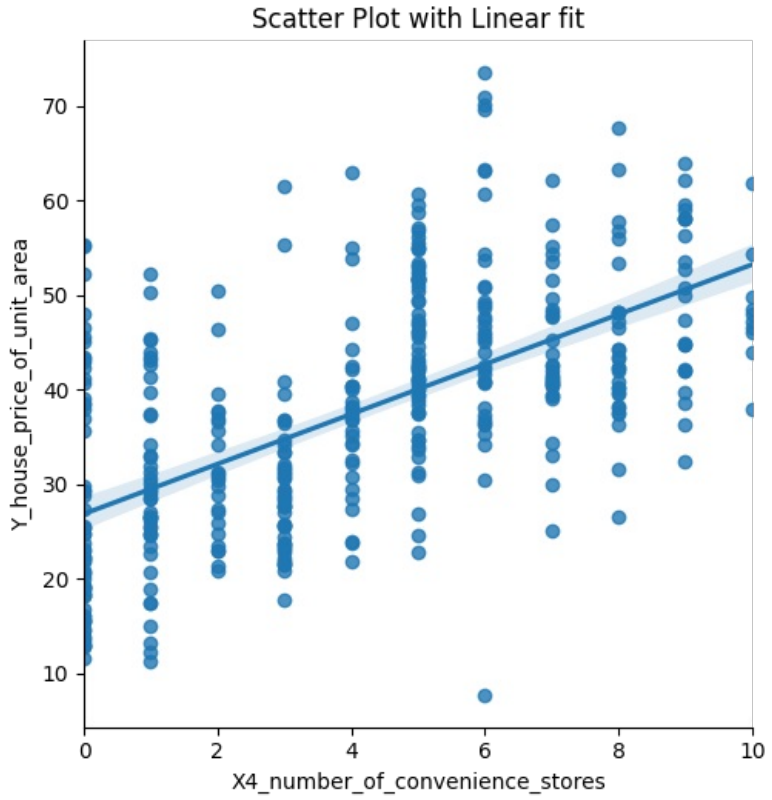
In [ ]:
```
# For X3 distance to the nearest MRT station
show_graph (X= "X3_distance_to_the_nearest_MRT_station", Y="Y_house_price_of_unit_area", data= df)
# Result below shows good relationship with no outliers
```

Out[ ]: Text(0.5, 1.0, 'Scatter Plot with Linear fit')



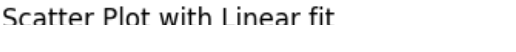
In [ ]:
```
# For X4 number of convenience stores
show_graph (X= "X4_number_of_convenience_stores", Y="Y_house_price_of_unit_area", data= df)
# Result below shows good relationship with no outliers
```

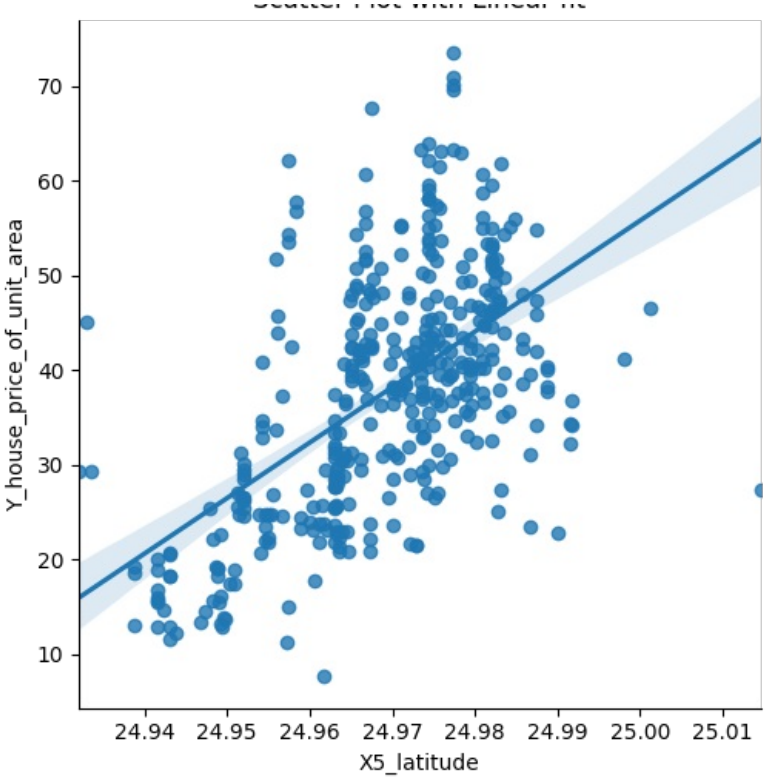
Out[ ]: Text(0.5, 1.0, 'Scatter Plot with Linear fit')



In [ ]:
```
# For X5 latitude
show_graph (X= "X5_latitude", Y="Y_house_price_of_unit_area", data= df)
# Result below shows good relationship with one outlier on the independent varaible
```

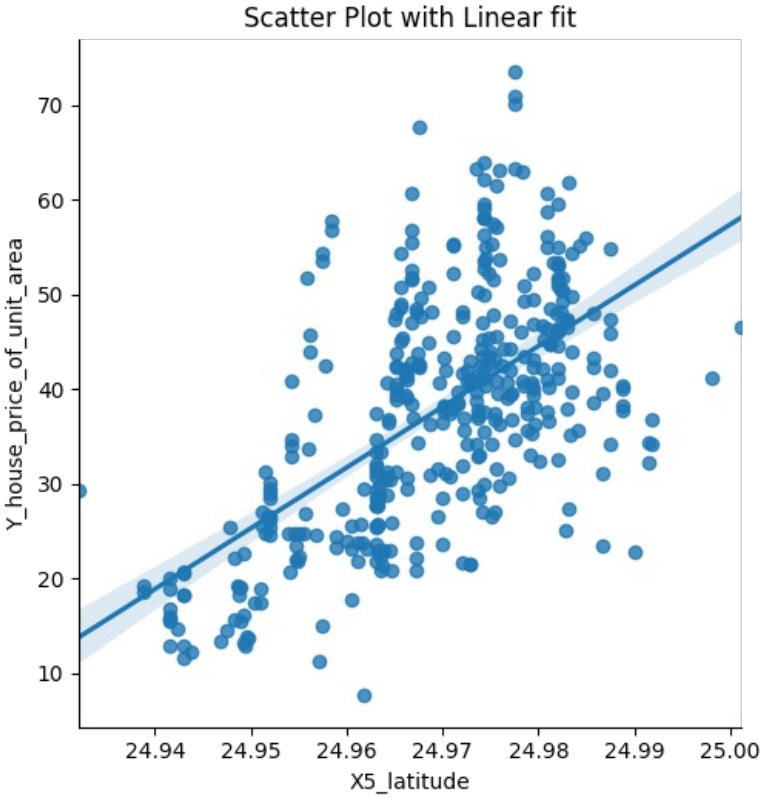
Out[ ]: Text(0.5, 1.0, 'Scatter Plot with Linear fit')





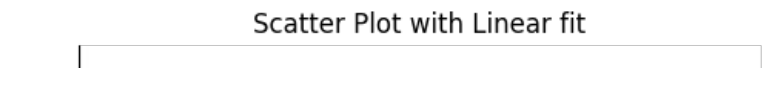
In [ ]:
```
#Remove outlier on this independent variable
outlier_detector (df.X5_latitude)
show_graph (X= "X5_latitude", Y="Y_house_price_of_unit_area", data= df)
# Result below shows good relationship with no more outliers
```

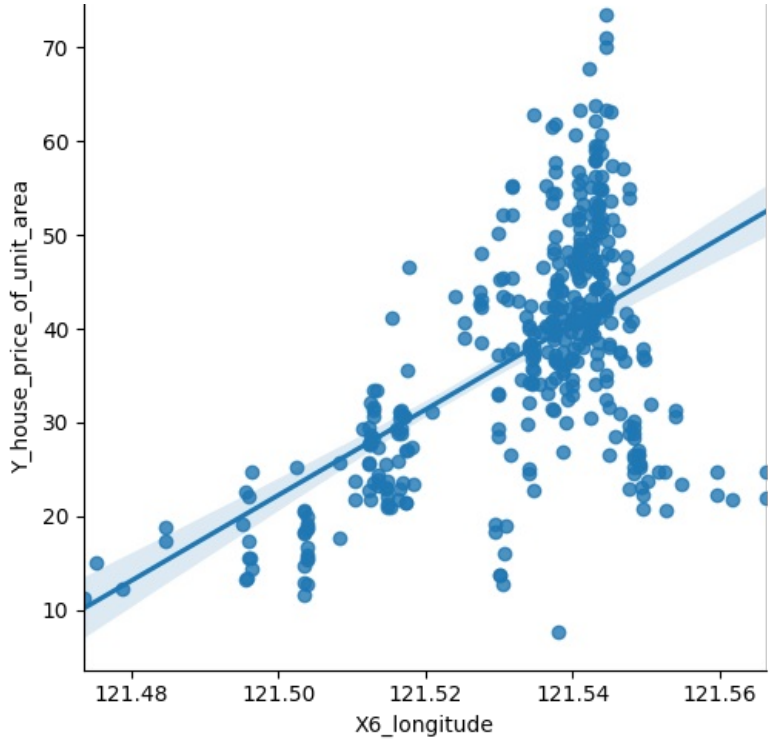
Out[ ]: Text(0.5, 1.0, 'Scatter Plot with Linear fit')



In [ ]:
```
# For X6 longitude
show_graph (X= "X6_longitude", Y="Y_house_price_of_unit_area", data= df)
# Result below shows good relationship with no outliers
```

Out[ ]: Text(0.5, 1.0, 'Scatter Plot with Linear fit')





Statistical Analysis

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

Out[ ]:

	X2_house_age	X3_distance_to_the_nearest_MRT_station	X4_number_of_convenience_stores	X5_latitude	X6_longitude	Y_house_price_of_unit_
0	32.0	84.87882	10	24.98298	121.54024	
1	19.5	306.59470	9	24.98034	121.53951	
2	13.3	561.98450	5	24.98746	121.54391	
3	13.3	561.98450	5	24.98746	121.54391	
4	5.0	390.56840	5	24.97937	121.54245	

In [ ]:
```
#Independent variables
x=model_data.drop('Y_house_price_of_unit_area',axis=1)
x.head()
```

Out[ ]:

	X2_house_age	X3_distance_to_the_nearest_MRT_station	X4_number_of_convenience_stores	X5_latitude	X6_longitude
0	32.0	84.87882	10	24.98298	121.54024
1	19.5	306.59470	9	24.98034	121.53951
2	13.3	561.98450	5	24.98746	121.54391
3	13.3	561.98450	5	24.98746	121.54391
4	5.0	390.56840	5	24.97937	121.54245

In [ ]:
```
#Dependent Variable
y = model_data['Y_house_price_of_unit_area']
y.head()
```

Out[ ]:
```
0    37.9
1    42.2
2    47.3
3    54.8
4    43.1
Name: Y_house_price_of_unit_area, dtype: float64
```

Train Data and get best Model

In [ ]:
```
import statsmodels.api as sm
```

In [ ]:
```
import statsmodels.api as sm
x=sm.add_constant(x)
model=sm.OLS(y,x).fit()
model.summary()
```

Out[ ]:

OLS Regression Results

Dep. Variable:	Y_house_price_of_unit_area	**R-squared:**	0.651
Model:	OLS	**Adj. R-squared:**	0.647
Method:	Least Squares	**F-statistic:**	148.4
Date:	Mon, 24 May 2021	**Prob (F-statistic):**	1.59e-88
Time:	18:36:39	**Log-Likelihood:**	-1381.8
No. Observations:	403	**AIC:**	2776.
Df Residuals:	397	**BIC:**	2800.
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-9509.3405	5319.637	-1.788	0.075	-2e+04	948.839
X2_house_age	-0.2733	0.033	-8.242	0.000	-0.338	-0.208
X3_distance_to_the_nearest_MRT_station	-0.0036	0.001	-5.767	0.000	-0.005	-0.002
X4_number_of_convenience_stores	1.1674	0.162	7.209	0.000	0.849	1.486
X5_latitude	282.5482	40.677	6.946	0.000	202.578	362.518
X6_longitude	20.5363	41.698	0.493	0.623	-61.440	102.513

Omnibus:	37.807	**Durbin-Watson:**	1.980
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	86.278
Skew:	0.491	**Prob(JB):**	1.84e-19
Kurtosis:	5.043	**Cond. No.**	2.33e+07

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.33e+07. This might indicate that there are
strong multicollinearity or other numerical problems.

In [ ]:
```
#Remove variable with bad pvalue
x=x.drop(['X6_longitude'],axis=1)
model=sm.OLS(y,x).fit()
model.summary()
```

Out[ ]:

OLS Regression Results

Dep. Variable:	Y_house_price_of_unit_area	**R-squared:**	0.651
Model:	OLS	**Adj. R-squared:**	0.648
Method:	Least Squares	**F-statistic:**	185.8
Date:	Mon, 24 May 2021	**Prob (F-statistic):**	1.23e-89
Time:	18:36:39	**Log-Likelihood:**	-1381.9
No. Observations:	403	**AIC:**	2774.
Df Residuals:	398	**BIC:**	2794.
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-6936.4765	1002.774	-6.917	0.000	-8907.872	-4965.081
X2_house_age	-0.2738	0.033	-8.269	0.000	-0.339	-0.209
X3_distance_to_the_nearest_MRT_station	-0.0039	0.000	-9.050	0.000	-0.005	-0.003
X4_number_of_convenience_stores	1.1627	0.162	7.199	0.000	0.845	1.480
X5_latitude	279.4747	40.158	6.959	0.000	200.527	358.422

Omnibus:	37.506	**Durbin-Watson:**	1.981
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	86.865
Skew:	0.482	**Prob(JB):**	1.37e-19
Kurtosis:	5.060	**Cond. No.**	4.39e+06

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.39e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

Divide your dataset into Train (80 Percent) and Test (20 percent)

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size=0.20, random_state=1)


print("number of test samples on X:", X_test.shape[0])
print("number of training samples on X:",X_train.shape[0])

print("number of test samples on Y:", Y_test.shape[0])
print("number of training samples on Y:",Y_train.shape[0])
```

```
number of test samples on X: 81
number of training samples on X: 322
number of test samples on Y: 81
number of training samples on Y: 322
```

Multiple Linear Regression

In [ ]:
```
# %%capture
# ! pip install ipywidgets
from ipywidgets import interact, interactive, fixed, interact_manual
from sklearn.linear_model import LinearRegression
```

In [ ]:
```
lr=LinearRegression()
lr.fit(X_train, Y_train)
```

Out[ ]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [ ]:
```
#Prediction using test data:
yhat_test = lr.predict(X_test)
yhat_test[0:5]
```

Out[ ]:
```
array([45.66166314, 44.72593071, 52.15165935, 22.00465565, 41.96849825])
```

In [ ]:
```
#Intercept and independent variable coefficients
intercept =lr.intercept_
coefficient =lr.coef_
print('Intercept:{0:0.3f}'.format(intercept))
print('Coefficient(s): ', (*['{0:0.4}'.format(i) for i in coefficient]),sep='\n' )
```

```
Intercept:-7893.161
Coefficient(s):
0.0
-0.2643
-0.003351
1.225
317.7
```

In [ ]:
```
import sklearn.metrics as sma
from sklearn.model_selection import cross_val_score
print("Mean absolute error =", round(sma.mean_absolute_error(Y_test, yhat_test), 2))
print("Mean squared error =", round(sma.mean_squared_error(Y_test, yhat_test), 2))
print("Median absolute error =", round(sma.median_absolute_error(Y_test, yhat_test), 2))
print("R-square  =", round(sma.r2_score(Y_test,yhat_test,), 2))
```

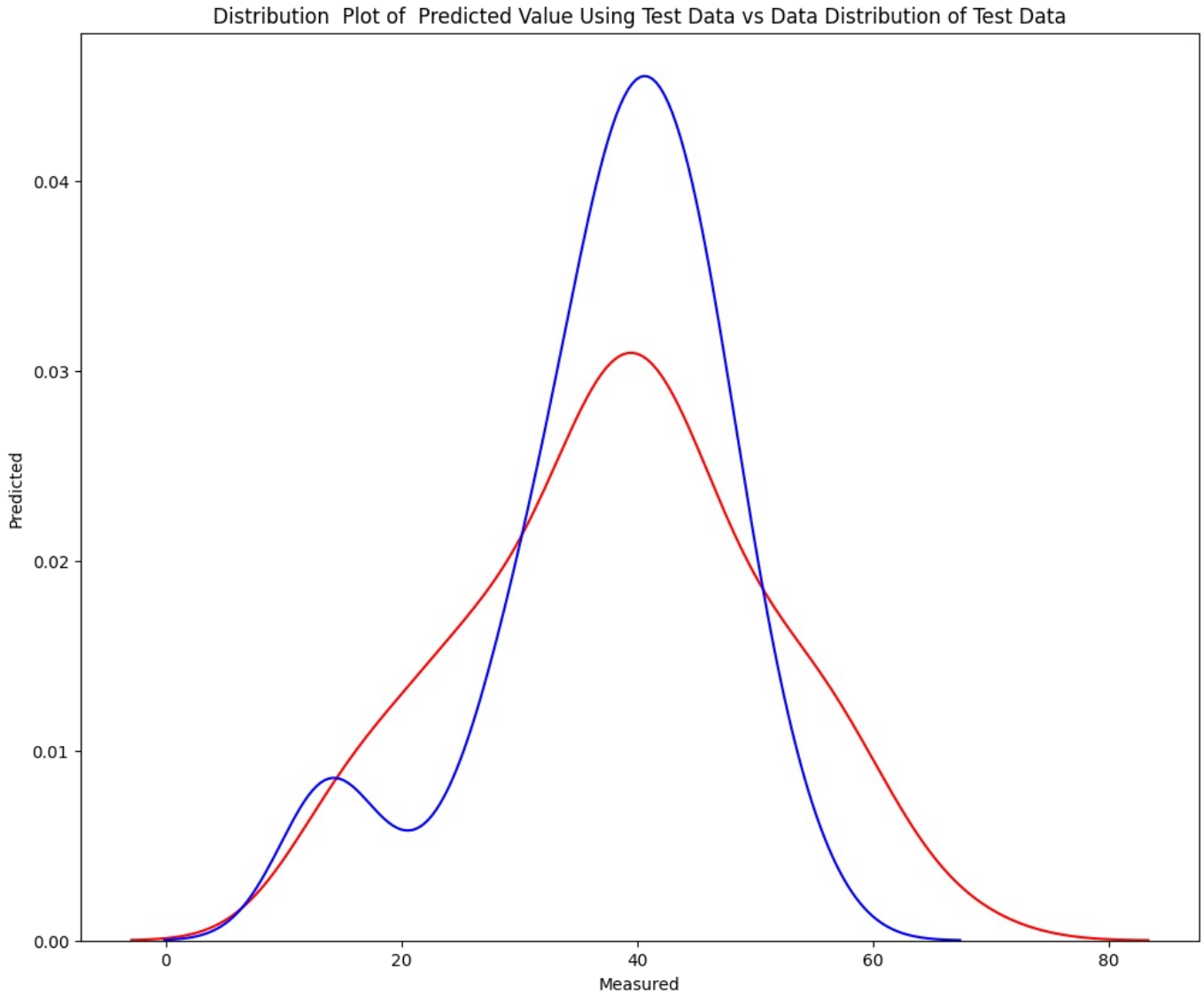
```
Mean absolute error = 5.2
Mean squared error = 51.77
Median absolute error = 3.78
R-square  = 0.67
```

Result Visualization

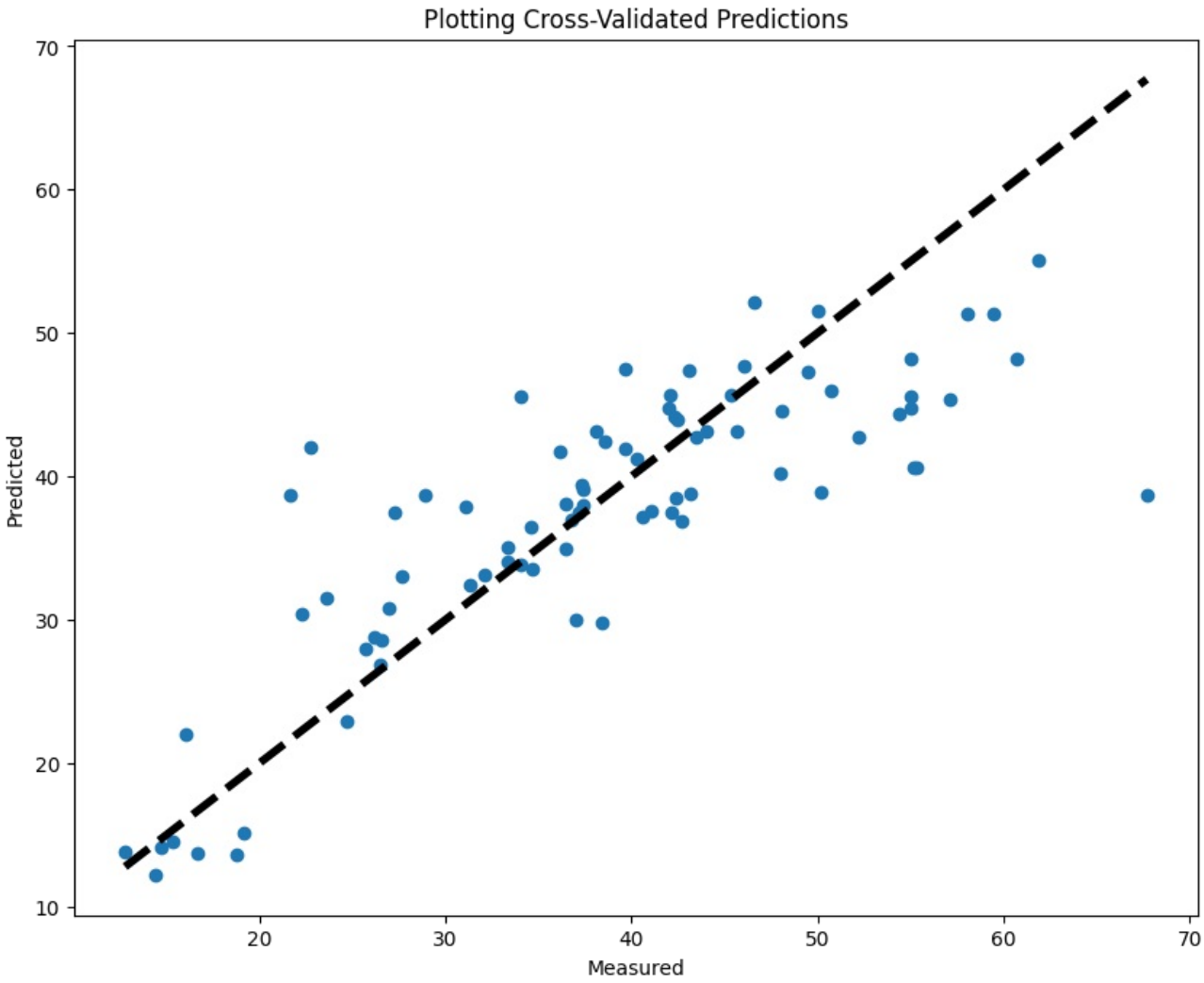
In [ ]:
```
#Functions for plotting
def DistributionPlot(RedFunction, BlueFunction, RedName, BlueName, Title):
    width = 12
    height = 10
    plt.figure(figsize=(width, height))

    ax1 = sns.distplot(RedFunction, hist=False, color="r", label=RedName)
    ax2 = sns.distplot(BlueFunction, hist=False, color="b", label=BlueName, ax=ax1)

    plt.title(Title)
    plt.xlabel('Measured')
    plt.ylabel('Predicted')

    plt.show()
    plt.close()
```

In [ ]:
```
#Plot of predicted value using the test data compared to the test data.
Title='Distribution  Plot of  Predicted Value Using Test Data vs Data Distribution of Test Data'
DistributionPlot(Y_test,yhat_test,"Actual Values (Test)","Predicted Values (Test)",Title)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a deprecated f
unction and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level f
unction with similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a deprecated f
unction and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level f
unction with similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
```



In [ ]:
```
fig, ax = plt.subplots()
ax.scatter(Y_test,yhat_test)
ax.plot([Y_test.min(), Y_test.max()], [Y_test.min(), Y_test.max()], 'k--', lw=4)
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
plt.title('Plotting Cross-Validated Predictions')
plt.show()
```



Summary and Conclusions

Looking at the R-squared value we see that it is 0.67 which means that 67% of the time the prediction of house price is correct. The p value is also very low indicating that the fit of the model is statistically significant, and that the prediction is true. The multi linear regression model developed is capable of predicting the test data with effective result

In [ ]: