GROUP ASSIGNMENT & INDIVIDUAL ASSIGNMENT PART 2

CT032-3-3-FAI

Further Artificial Intelligence

APU3F2209CSIS & APD3F2209CSIS

HAND IN DATE: **15 DECEMBER 2022**

WEIGHTAGE: **50%**

TP Number	Student Name
TP056066	Yan Mun Kye
TP061524	Hor Shen Hau
TP056267	Tan Sheng Jeh
TP060810	Sia De Long

Table of Contents

1.0 Application Demo

The developers had implemented a house price prediction system that will take in several parameters like area of the house, number of bedrooms, number of bathrooms, whether it is located at the main road, presence of a basement, and so on. As mentioned in the Part 1 of the assignment, the users will be able to choose the model to they prefer to use, either Linear Regression or Polynomial Regression.

Before allowing the users to input the parameters and make a prediction, it is important to build and train the model first. The detailed implementation and choosing of hyperparameters will be explained in the next section. In this section, the model deployed is the final tuned model.

```
preprocessed = poly_preprocessing(data)
high_corr = ['area','bathrooms', 'airconditioning_yes','stories', 'parking']

X = preprocessed[high_corr]
y = preprocessed['price']

models = dict()
models['polynomial'] = make_poly_model(X, y)
```

```
preprocessed_lm = linear_preprocessing(data)
Y = preprocessed_lm.price
# includes the fields other than prices
X = preprocessed_lm.iloc[:,1:]

X.drop(['mainroad','bedrooms'], axis=1, inplace=True)
status = pd.get_dummies(X['furnishingstatus'], drop_first=True)
X = pd.concat([X, status], axis=1)
X.drop(columns='furnishingstatus',inplace=True)

models['linear'] = make_linear_model(X, Y)
```

Preprocessing is done on the training data, and the model is trained with the training data. After training the models, the models are stored in a dictionary called 'models', in which will be accessed by the main application.

```
while True:

    print(f"Select your model choice: ")
    print(f"1. Linear Regression")
    print(f"2. Polynomial Regression")
    choice = input(f"Choice : ")
    model = None
    if (choice == '1'):
        model = models["linear"]
    elif (choice == '2'):
        model = models["polynomial"]
    else:
        model = None
        break

    area = int(input("Area (sq ft): "))
    bedrooms = int(input("Bedrooms: "))
    bathrooms = int(input("Bathrooms: "))
    stories = int(input("Stories: "))
    parking = int(input("Parking:"))
    mainroad = input("Mainroad (yes/no): ")
    guestroom = input("Guestroom (yes/no): ")
    basement = input("Basement (yes/no): ")
    hotwaterheating = input("Hot water heating (yes/no): ")
    airconditioning = input("Air cond (yes/no): ")
    prefarea = input("Prefarea (yes/no): ")
    furninshing = input("Furnishing (unfurnished/semi-furnished/furnished): ")
```

```
    ans = pd.DataFrame(
        {
            'area':[area],
            'bedrooms':[bedrooms],
            'bathrooms':[bathrooms],
            'stories':[stories],
            'mainroad':[mainroad],
            'guestroom':[guestroom],
            'basement':[basement],
            'hotwaterheating':[hotwaterheating],
            'airconditioning':[airconditioning],
            'parking':[parking],
            'prefarea':[prefarea],
            'furnishingstatus':[furninshing]

        }
    )

    if choice == '1' :
        ans = linear_preprocessing(ans)
        ans.drop(['mainroad','bedrooms'], axis=1, inplace=True)
        status = ans.furnishingstatus[0]
        ans.drop(columns='furnishingstatus',inplace=True)
        ans['semi-furnished'] = 1 if (status == 'semi-furnished' and status != 'furnished' and status != 'unfurnished') else
        ans['unfurnished'] = 1 if (status != 'semi-furnished' and status != 'furnished' and status == 'unfurnished') else 0
        print(f"Predicted Price: {model.predict(ans)[0]:.02f}")
    elif choice == '2':
        ans = poly_preprocessing(ans)
        transformer = PolynomialFeatures(degree=2)
        transformedX = transformer.fit_transform(ans)
        print(f"Predicted Price: {model.predict(transformedX)[0]:.02f}")

    print(f"Continue? (y/n)")
    if input() == 'n':
        break
```

In this part of the application, the user is able to input the parameters in which the models will be using to do the predictions. After each prediction, a prompt will ask the user whether they want to continue with the prediction. Below shows a sample of using linear regression and polynomial regression for the prediction.

After choosing the model, the program will prompt the user for the parameters like area of the house, number of bedrooms, number of bathrooms, number of stories, whether it is located at the mainroad, presence of guestroom, presence of basement, presence of hot water heating, presence of air conditioning, number of parkings, whether it is a preferred area and the furnishing status of the house. Then, the values will be stored into a pandas DataFrame. Depending on the model choice of the user, different preprocessing will occur in the answer provided by the user. Finally, the preprocessed and transformed data is fed into the models to produce a prediction.

Using Linear Regression

```
Select your model choice:
1. Linear Regression
2. Polynomial Regression

Choice : 1
```

```
Select your model choice:
1. Linear Regression
2. Polynomial Regression
Choice : 1
Area (sq ft): 2000
Bedrooms: 3
Bathrooms: 2
Stories: 2
Parking:1
Mainroad (yes/no): no
Guestroom (yes/no): no
Basement (yes/no): yes
Hot water heating (yes/no): yes
Air cond (yes/no): yes
Prefarea (yes/no): no
Furnishing (unfurnished/semi-furnished/furnished): semi-furnished
Predicted Price: 6276702.34
Continue? (y/n)
```

Using Polynomial Regression

```
Select your model choice:
1. Linear Regression
2. Polynomial Regression

Choice : 2
```

```
Select your model choice:
1. Linear Regression
2. Polynomial Regression
Choice : 2
Area (sq ft): 2000
Bedrooms: 3
Bathrooms: 2
Stories: 2
Parking:1
Mainroad (yes/no): no
Guestroom (yes/no): no
Basement (yes/no): yes
Hot water heating (yes/no): yes
Air cond (yes/no): yes
Prefarea (yes/no): no
Furnishing (unfurnished/semi-furnished/furnished): semi-furnished
Predicted Price: 5603092.52
Continue? (y/n)
```

As we can see, both linear regression and polynomial regression produces slightly different results even with the same input parameters. This is because there are differences in choosing attributes and preprocessing between the two models.

2.0 Evaluate of Model

2.1 Plotting Experimental Data (Sia De Long TP060810)

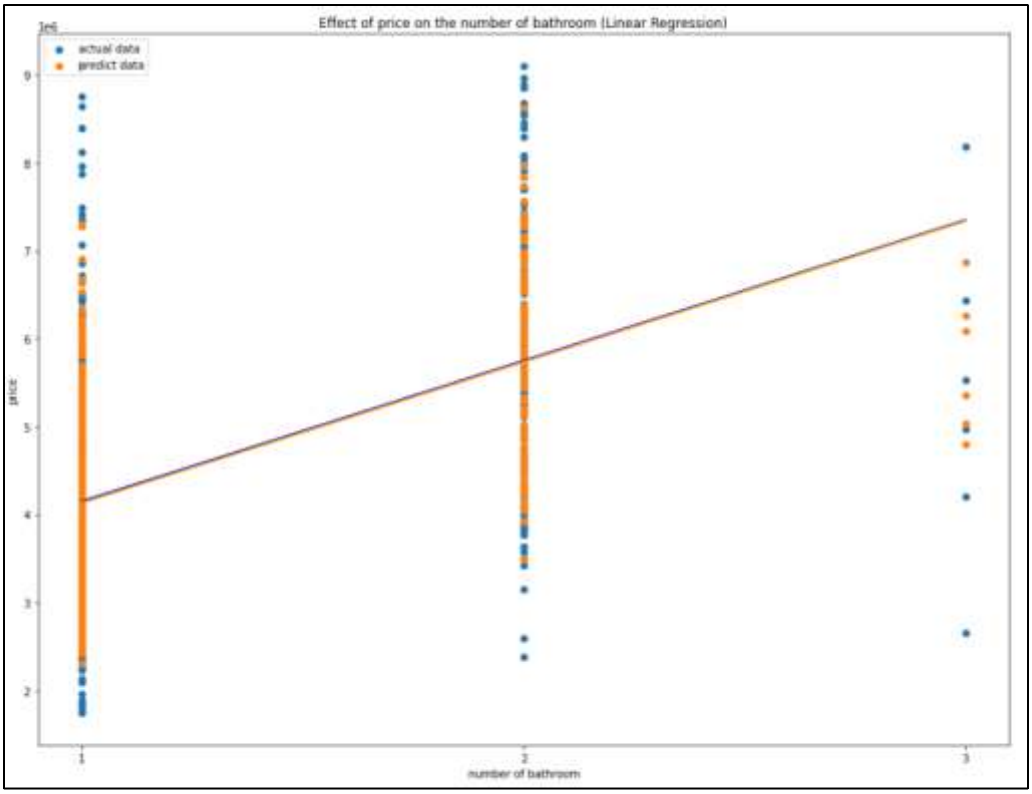


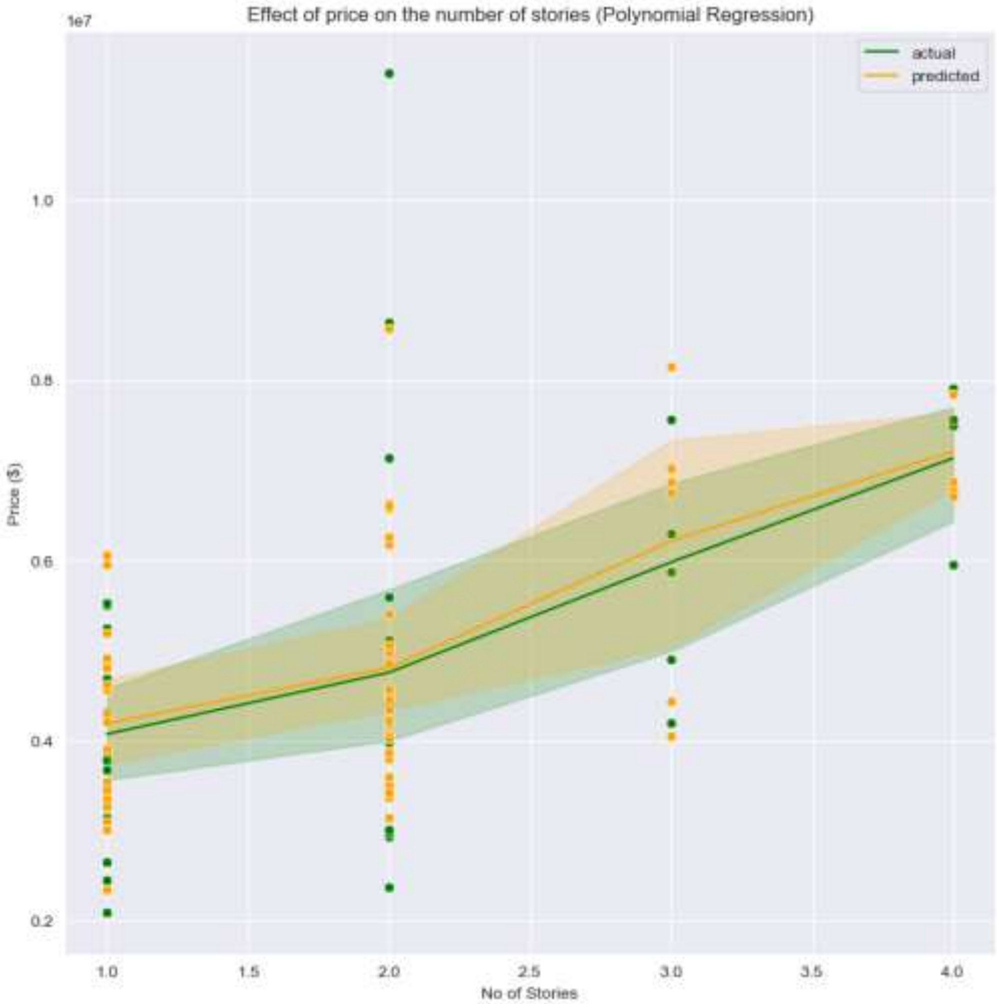
Figure 1: Plotting Experimental Data

Figure above shows the correlation between the number of bathroom and the housing price for both actual data and predicted data. The blue points are the actual price value in the original data while the orange points are the price predicted by the trained linear regression model. There is also a blue regression line represent for the blue points and an orange line represent for the orange points.

From the graph, the distribution of the data points for actual price and predicted price is quite similar. In fact, the predicted data points are wrapped by the actual price which shows that the predicted price will not exceed the actual price and create an outlier. Besides that, from the gradient of the correlation line on the graph, it shows that the relationship between bathroom and price are positive for both actual price and predicted price. Although the correlation line cannot be visible clearly, it is because the actual price and the predicted price calculated by the model do not have a significant difference meaning that the model can have a high accuracy of predicting the housing price on different number of bathrooms.

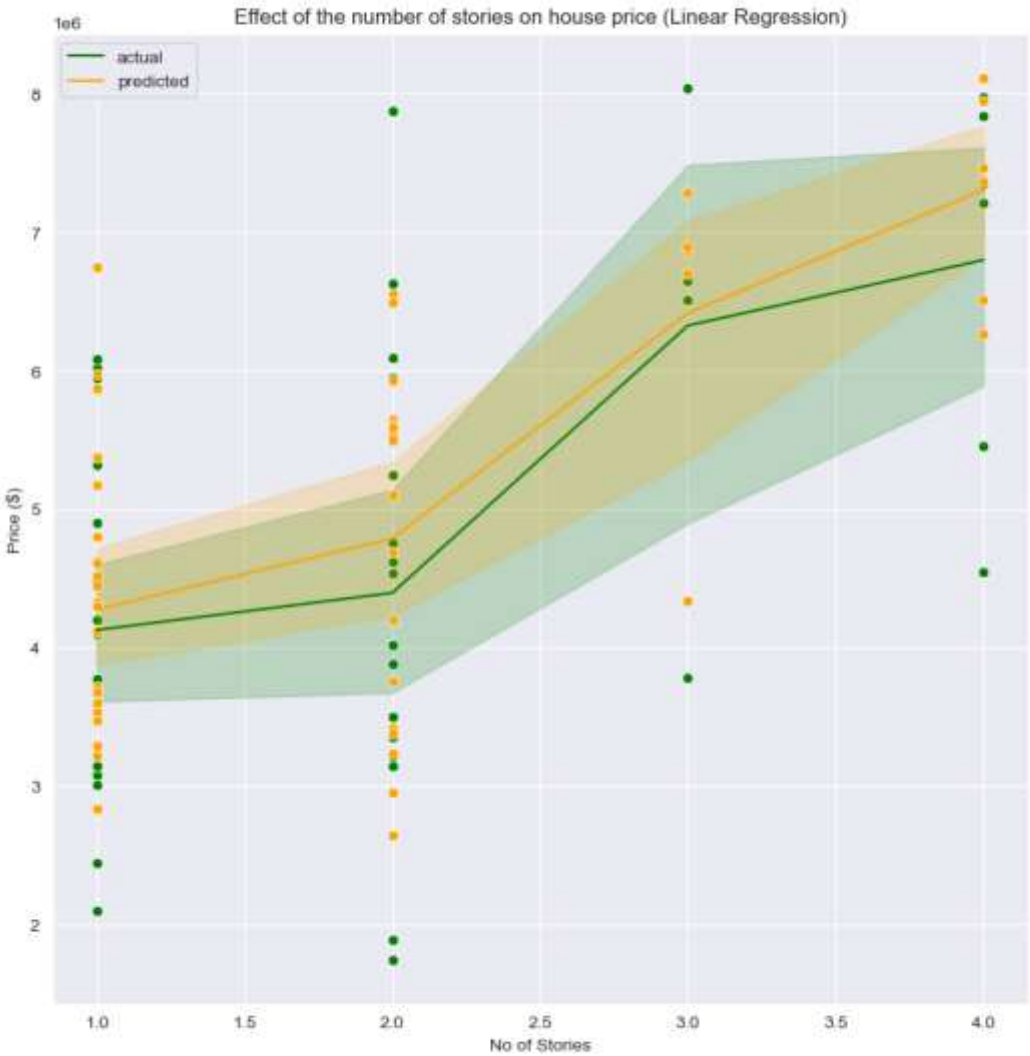
2.2 Plotting Experimental Data (Yan Mun Kye TP056066)

The graph below shows the effect of the price on the number of stories. The graph is plotted using the unseen test data. The green points are the actual price and orange points are the price predicted by the polynomial regression model. The green and orange lines represent the overall trend of the actual and predicted price of the property.



As shown clearly in the graph, the distribution of points are quite similar in both actual price and predicted price. From the points, it may be difficult to see all the points as there may be overlaps of the points. However, it is easy to spot the predicted price are all within the range of the actual price. The line plot shows more clearly that in general, the model is able to predict the price of the trend of the actual price of the house in terms of the number of stories of the house. However, it is noticeable that the predicted trend graph is slightly higher than the actual trend graph. It can be inferred that the number of stories has a higher effect on the price of the house in the polynomial regression model, which will cause the prediction to be slightly higher than the actual amount. However, there is no huge change in the difference in height of the trend, which can be inferred that the model has low error variance across the number of stories of the house.

The graph below shows a similar effect of the number of stories on house price. However, the graph results below are calculated using the linear regression model.



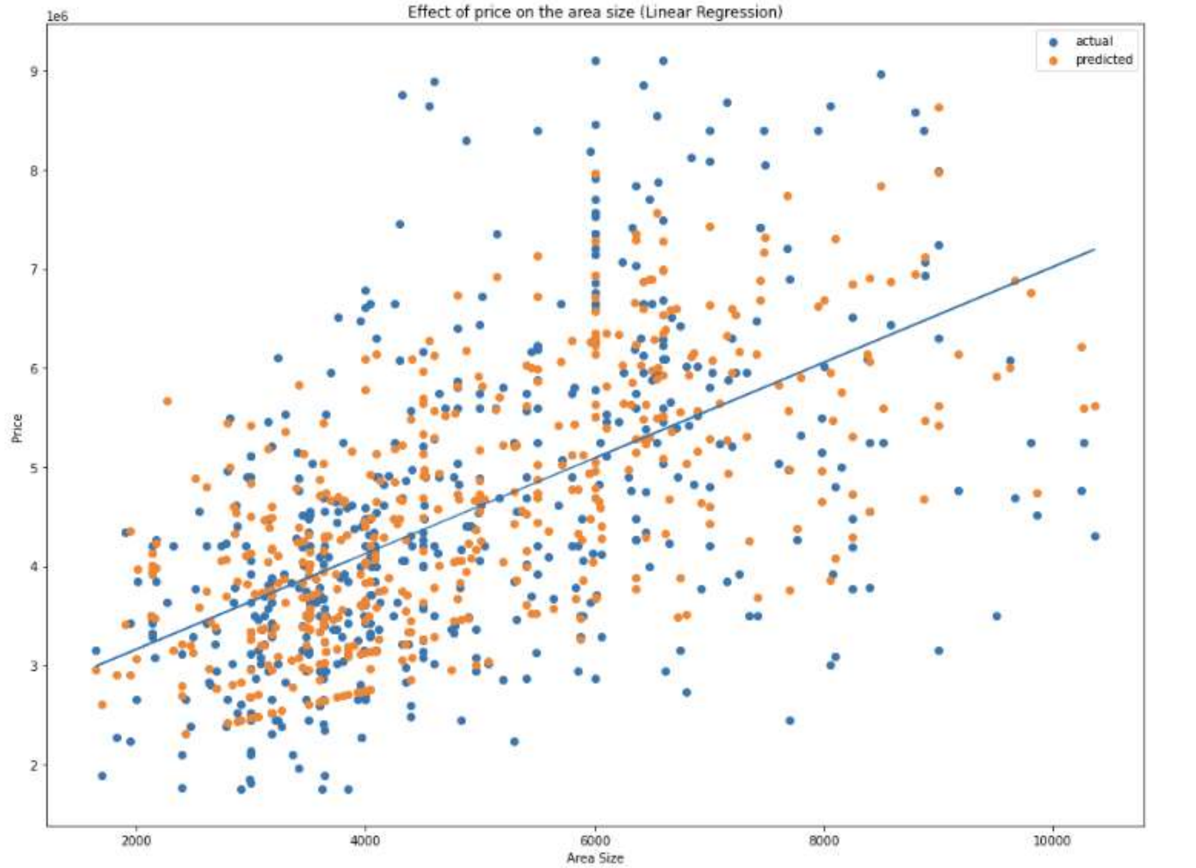
Similar to polynomial regression, the linear regression model is able to capture the trend of the house price against the number of stories. For the most part, the prediction values are all within the actual price range of the houses at each number of stories. Except for when stories equals 1 and stories equals 4. From the graph, we can see that there are some predicted value are higher than the actual value. Similar to the polynomial regression model, the model predicts the price of each number of stories to be slightly higher. However, there are more variance in the error for the linear regression model.

In conclusion, both Polynomial Regression and Linear Regression is able to capture the trend of the price against number of stories. However, a solid conclusion cannot be drawn from this analysis as there are other factors and attributes that affect the performance of the model, and this is only one of the attributes.

2.3 Plotting Experimental Data (Tan Sheng Jeh TP056267)

```
y_predicted = regression.predict(X)

plot.figure(figsize=(16,12))
plot.scatter(X['area'], Y)
plot.scatter(X['area'], y_predicted)
m, b = np.polyfit(X['area'], y_predicted, 1)
plot.plot(X['area'], m*X['area']+b)
plot.xlabel('Area Size')
plot.ylabel('Price')
plot.title("Effect of price on the area size (Linear Regression)")
plot.legend(['actual', 'predicted'], loc='upper right')
```

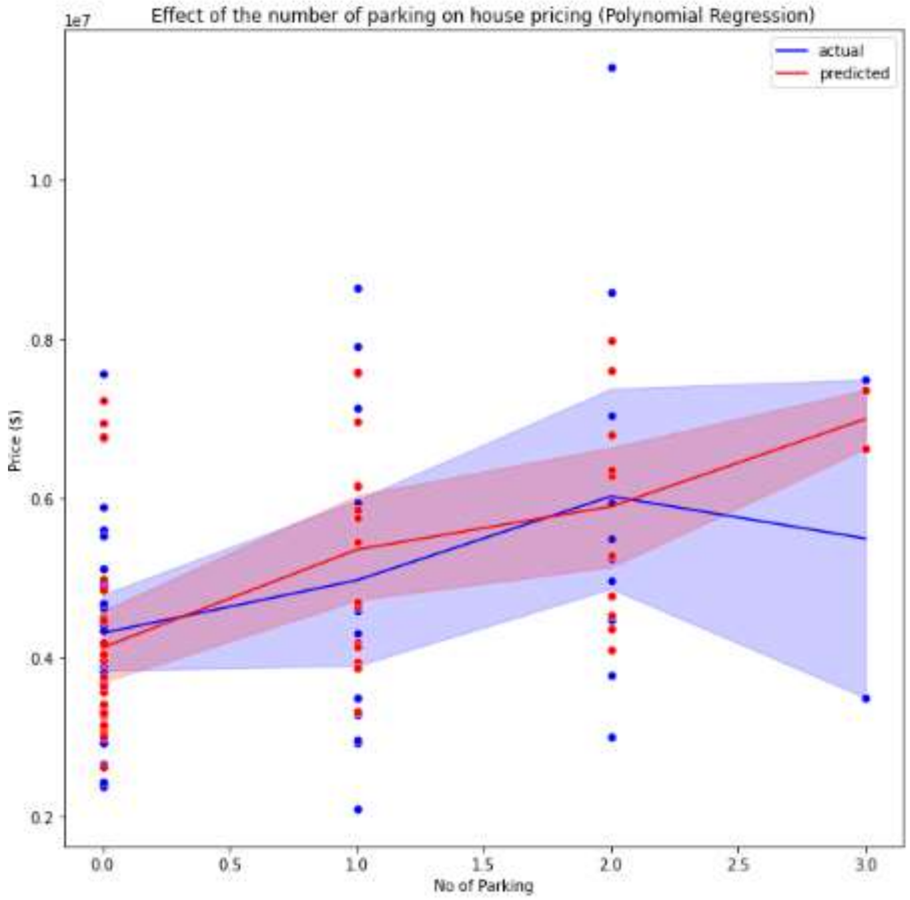
```
<matplotlib.legend.Legend at 0x2367a802670>
```



The graph above shows the effect of price on the area size of housing. The blue point represents the actual price while the orange point represents the predicted price through multiple linear regression models. The blue orange line in the middle is the best fit line for the price. From the graph plotted, it can be seen that the area around 4000-meter square has more dots concentrated which means that the majority of the houses are built with that size. Meanwhile, the best fit line shows that the distance between actual and predicted data is not far away as the line is located at the lower point of the graph. The area with the highest predicted price would be an area with around 9000 square feet. Besides, the number of houses with bigger area size are lesser than the ones with smaller area size. Therefore, the Multiple Linear Regression can be seen as the optimal plot for the effects of price on the area size of housing.

2.4 Plotting Experimental Data (Hor Shen Hau TP061524)

The graph shows how the number of parking affects the price of the housing. The blue points and lines represent the actual price of the housing while the red lines represent the predicted price of the housing by the polynomial regression model based on the number of parking.



It can be observed from the graph that the actual and predicted price points are relatively similar except at the end where the trend line of the predicted price is higher than that of the actual price. The predicted price is all within the ranges of the actual price which shows that the model is able to predict the price of the housing based on the number of parking but it is worth mentioning that there are times the height of the actual price trend lines goes below that of the predicted price trendline and vice versa. This may imply that the model has a high error variance when it comes to the number of parking.

3.0 Implementation

3.1 Multiple Linear Regression

3.1.1 Data Preprocessing

Importing Libraries

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plot
import statsmodels.api as sm
import seaborn as sns
from sklearn import metrics
```

The first step is to import the libraries that will be used for model training. For instance, numpy and pandas packages are imported while matplotlib is imported for data visualization and sklearn is imported for evaluation metrics.

Reading dataset

```
data = pd.read_csv(r'Housing.csv')
data.head(5)
```

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

Then, the housing dataset will be read using the read_csv function to extract the columns and rows from the dataset. After that, the data.head() function will be used to show the first 5 rows of the dataset.

Summary of dataset

```
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
```

The summary of the dataset will be displayed using data.info() function to check whether there is any null attribute column on the dataset or not.

Description of dataset

```
data.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

The data.describe() function is to calculate statistical data of the dataframe like for example the mean, standard deviation and quartile of the data.

Size of dataset

```
data.shape
```

```
(545, 13)
```

The data.shape function is used to summarize the number of rows and columns in the dataset. For instance, the dataset had 545 rows and 13 columns.

Data Cleaning

```
data.isnull().sum()
```

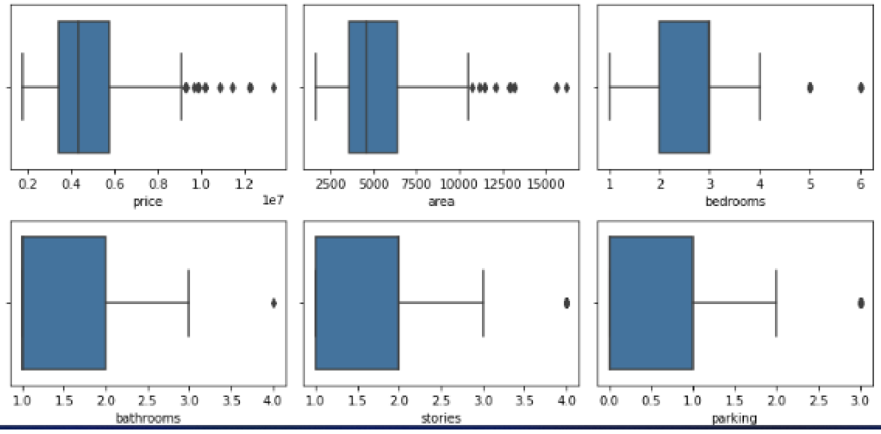
```
price               0
area                0
bedrooms            0
bathrooms           0
stories             0
mainroad            0
guestroom           0
basement            0
hotwaterheating     0
airconditioning     0
parking             0
prefarea            0
semi-furnished      0
unfurnished         0
dtype: int64
```

Data cleaning is being done to make sure there is no null value in the data as if there is null value present, it needs to be replaced so that the prediction result will not be affected.

Detect Outliers

```
def detectOutliers():
    fig, axs = plot.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])
    plot.tight_layout()
detectOutliers()
```

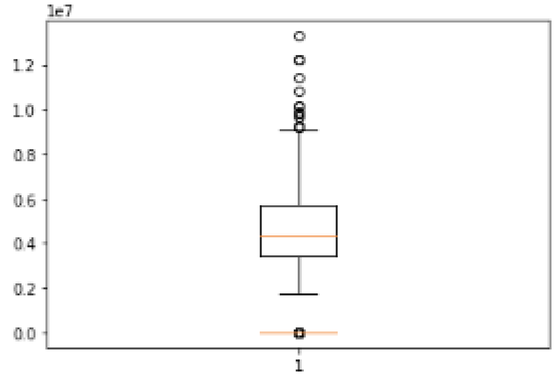
```
C:\Users\Asus\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword ar
g: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit key
word will result in an error or misinterpretation.
  warnings.warn(
C:\Users\Asus\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword ar
g: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit key
word will result in an error or misinterpretation.
  warnings.warn(
C:\Users\Asus\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword ar
g: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit key
word will result in an error or misinterpretation.
  warnings.warn(
C:\Users\Asus\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword ar
g: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit key
word will result in an error or misinterpretation.
  warnings.warn(
C:\Users\Asus\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword ar
g: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit key
word will result in an error or misinterpretation.
  warnings.warn(
C:\Users\Asus\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword ar
g: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit key
word will result in an error or misinterpretation.
  warnings.warn(
```



There is a function named detectOutliers() which is used to detect the outliers present in each of the columns of attributes. Then, boxplot was used to plot out the outliers found in the dataset. For instance, the price and area attributes are found to have a high number of outliers that need to be reduced.

Removing Outliers

```
# Outlier reduction for price
plot.boxplot(data.price)
Q1 = data.price.quantile(0.25)
Q3 = data.price.quantile(0.75)
IQR = Q3 - Q1
data = data[(data.price >= Q1 - 1.5*IQR) & (data.price <= Q3 + 1.5*IQR)]
# Outlier reduction for area
plot.boxplot(data.area)
Q1 = data.area.quantile(0.25)
Q3 = data.area.quantile(0.75)
IQR = Q3 - Q1
data = data[(data.area >= Q1 - 1.5*IQR) & (data.area <= Q3 + 1.5*IQR)]
```

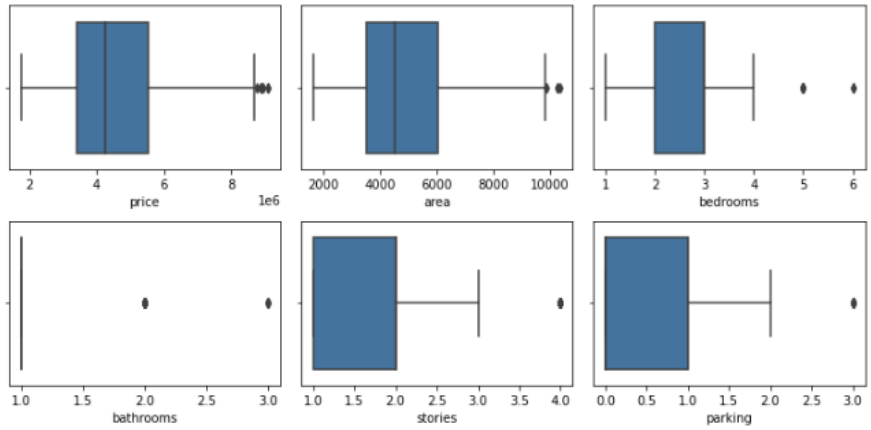


The outliers will be removed through the use of interquartile range to find out which value is outside of Q1 and Q3 range. Then, the outliers will be removed from the dataframe and the outliers will be shown on a boxplot graph.

Checking Outliers After Removed

```
detectOutliers()
```

```
C:\Users\Asus\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword ar
g: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit key
word will result in an error or misinterpretation.
  warnings.warn(
C:\Users\Asus\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword ar
g: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit key
word will result in an error or misinterpretation.
  warnings.warn(
C:\Users\Asus\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword ar
g: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit key
word will result in an error or misinterpretation.
  warnings.warn(
C:\Users\Asus\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword ar
g: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit key
word will result in an error or misinterpretation.
  warnings.warn(
C:\Users\Asus\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword ar
g: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit key
word will result in an error or misinterpretation.
  warnings.warn(
C:\Users\Asus\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword ar
g: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit key
word will result in an error or misinterpretation.
  warnings.warn(
```

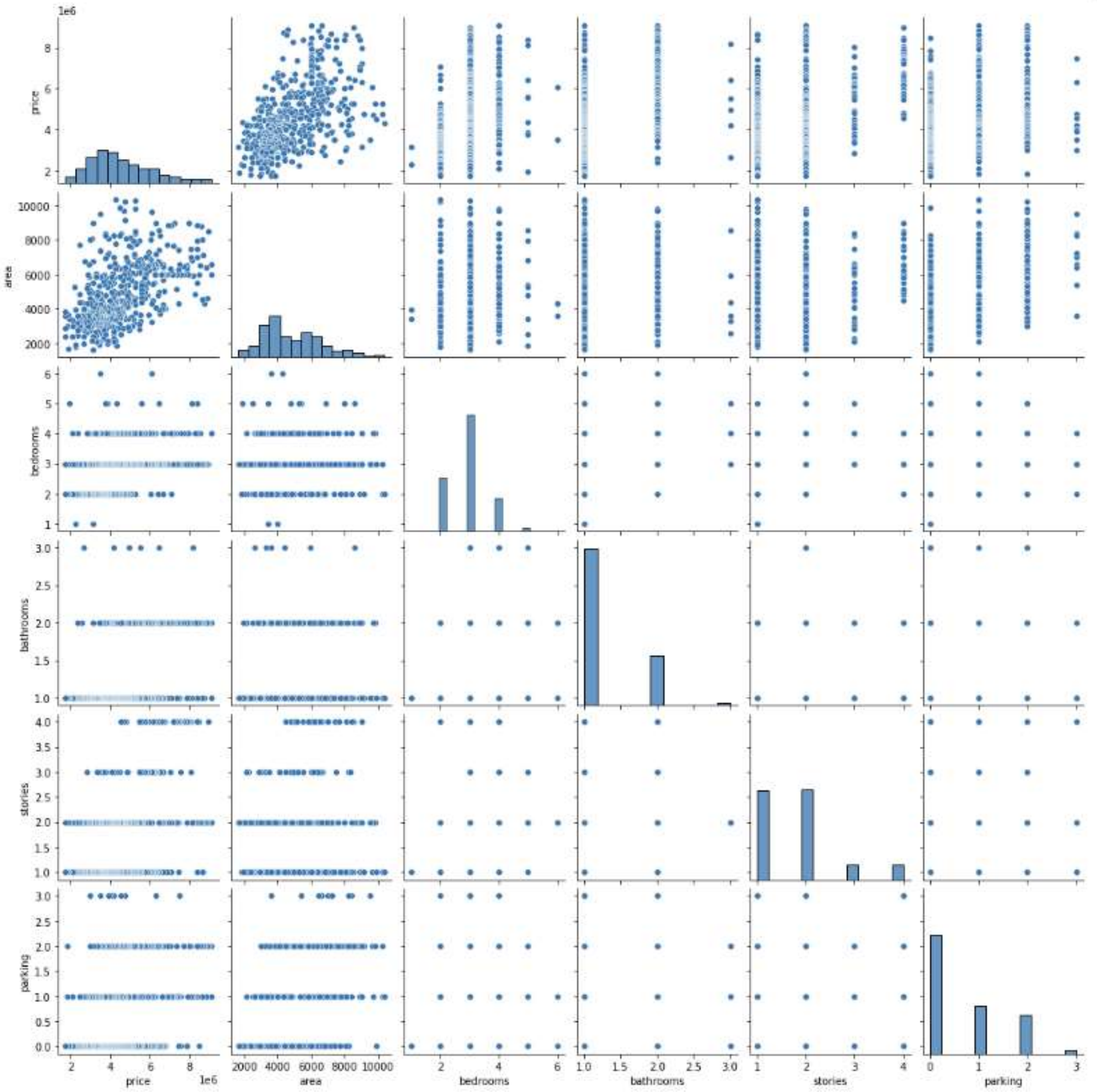


The detectOutliers() function will be used again to determine if the outliers on the data frame have been reduced. As can be seen, the price and area outliers had been reduced as less points can be found on the bottom side of the boxplot.

Data Visualization

Determine Relationship Between Variables

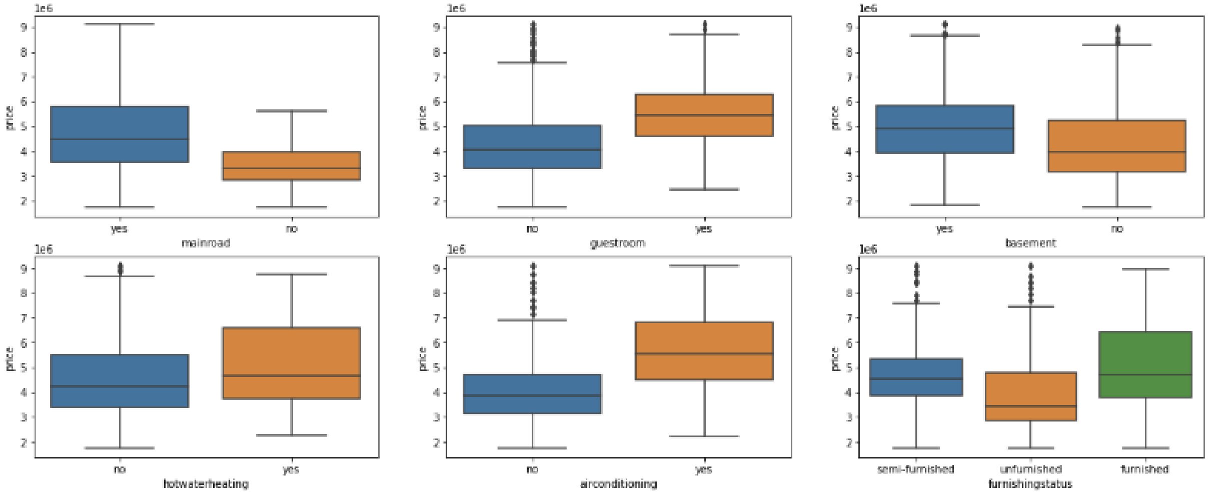
```
sns.pairplot(data)
plot.show()
```



The seaborn.pairplot() function is used to determine the relationship between pairs of variables. For instance, the relationship between price and area.

Visualize variables

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



The dependent variable which is 'price' is being visualized against each category of the independent variables and the result is plotted on a boxplot graph. The visualization is being done to determine the correlation between each independent variable with the dependent variable.

Data Preparation

Convert Data Type

```
def toNumeric(x):
    return x.map({"no":0,"yes":1})
def convert_binary():
    for column in list(data.select_dtypes(['object']).columns):
        if(column != 'furnishingstatus'):
            data[[column]] = data[[column]].apply(toNumeric)
convert_binary()
```

In order to fit data in a regression line, the data need to be numeric. As some of the independent variables consist of string data type, it needs to be converted to numeric type. The convert_binary function is to convert the 'yes' and 'no' data column to numeric form which is '0' and '1'.

Split Column For Variable

```
status = pd.get_dummies(data['furnishingstatus'])
status
```

	furnished	semi-furnished	unfurnished
0	0	1	0
1	0	0	1
2	1	0	0
3	1	0	0
4	0	1	0
...	...	...	...
512	0	0	1
513	0	1	0
514	0	0	1
515	1	0	0
516	0	0	1

517 rows × 3 columns

The dummy variable function is implemented to split the furnishingstatus column to 3 categories namely furnished, semi furnished, and unfurnished.

Dropping "Furnished" Column

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

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

```
data.drop(columns='furnishingstatus',inplace=True)
```

After categorizing the furnishingstatus column into 3 separate columns, the furnished column will be removed because of redundancy as only unfurnished and furnished columns will be concatenated into the dataframe and the furnishingstatus column will be dropped.

Selecting Data For Training

```
Y = data.price
# includes the fields other than prices
X = data.iloc[:,1:]
```

The dependent variable "price" will be selected and stored in Y while other remaining independent variable columns other than "price" will be selected and stored in X.

Determine Multicollinearity

```
from sklearn.preprocessing import MinMaxScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor
def preprocessing(X):
    scaler = MinMaxScaler()
    X_scaled = scaler.fit_transform(X)
    variables = X_scaled
    vif = pd.DataFrame()
    vif["VIF"] = [variance_inflation_factor(variables, i) for i in range(variables.shape[1])]
    vif["Features"] = X.columns
    print(vif)
```

```
preprocessing(X)
```

```
         VIF          Features
0   5.695074              area
1   7.370649          bedrooms
2   1.640001         bathrooms
3   2.702247           stories
4   5.841277          mainroad
5   1.521360         guestroom
6   1.998402          basement
7   1.077140   hotwaterheating
8   1.745831   airconditioning
9   1.912748           parking
10  1.444422          prefarea
11  2.306936     semi-furnished
12  1.941835       unfurnished
```

The above function is made to determine if Multicollinearity existed in the dataset. The multicollinearity columns need to be removed because it will compromise the statistical significance of independent variables. The severity of multicollinearity will be determined through Variance Inflation Factor (VIF). It was found that bedrooms and mainroad have the highest VIF value.

Dropping Multicollinearity Column

```
X.drop(['mainroad','bedrooms'], axis=1, inplace=True)
preprocessing(X)
```

```
         VIF         Features
0   4.272647             area
1   1.572188        bathrooms
2   2.134350          stories
3   1.518522        guestroom
4   1.832215         basement
5   1.074235  hotwaterheating
6   1.745076  airconditioning
7   1.873550          parking
8   1.422639         prefarea
9   1.859642    semi-furnished
10  1.545732      unfurnished
```

The columns with highest VIF value which are mainroad and bedrooms will be dropped from the dataframe. The dataset will be used to proceed to the next step once there is no multicollinearity found.

3.1.2 Data Splitting

Split Data Into Training and Testing Sets

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size = 0.2,random_state=355)
```

X and Y will be applied for training and test dataset with x_train and x_test act as the coordinates. The test size selected is 0.2 which means 20% of sample size of the dataset is selected and the random state is controlling the shuffling process of the dataset each time the data is being trained.

3.1.3 Data Training

Create Linear Regression Model

```
from sklearn.linear_model import LinearRegression
regression = LinearRegression()
regression.fit(x_train,y_train)
```

```
LinearRegression()
```

After the dataset is prepared, the regression model will be fit into the training set through the regression.fit() function.

Making Prediction

```
y_predict = regression.predict(x_test)
y_prediction = y_predict.round(2)
```

The predict function is used to generate predictions from the model after the data training and the value will be rounded off to 2 decimal places due to too many decimal places.

Displaying Results

```
comparison = pd.DataFrame(list(zip(y_train,y_prediction)), columns = ['Actual','Predicted'])
comparison
```

	Actual	Predicted
0	7455000	4096897.83
1	4900000	4701650.27
2	4515000	3276311.92
3	6300000	2968324.16
4	3353000	3026732.11
...	...	...
99	4095000	3779992.70
100	4473000	6439398.66
101	7560000	5304296.70
102	3640000	3882083.98
103	4235000	7109197.74

104 rows × 2 columns

The actual results before training and the prediction results after training will be compared. It was found that the value between both are still not too far off as it shows that the model has quite a good accuracy in predicting the house prices.

Maximum Scores Of Actual and Predicted Value

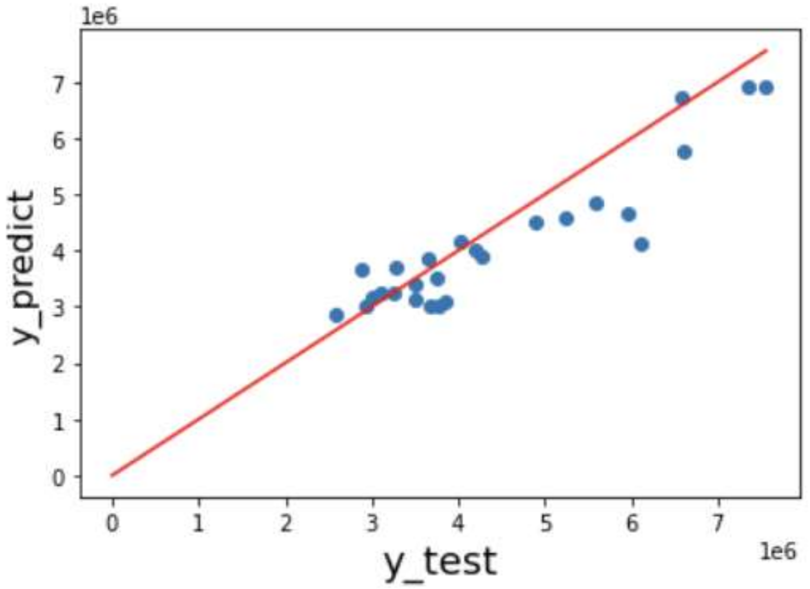
```
#overall maximum score of actual and predicted value
myMax = max(max(y_test), max(y_prediction))
myMax
```

```
9100000
```

The myMax function is used to find the maximum value of actual data and predicted data from the dataset.

Plotting Results in Scatter Plot

```
plot.scatter(y_test,y_prediction)
plot.xlabel('y_test', fontsize=18)
plot.ylabel('y_predict', fontsize=16)
plot.plot([0,myMax],[0,myMax],'r')
plot.show()
```

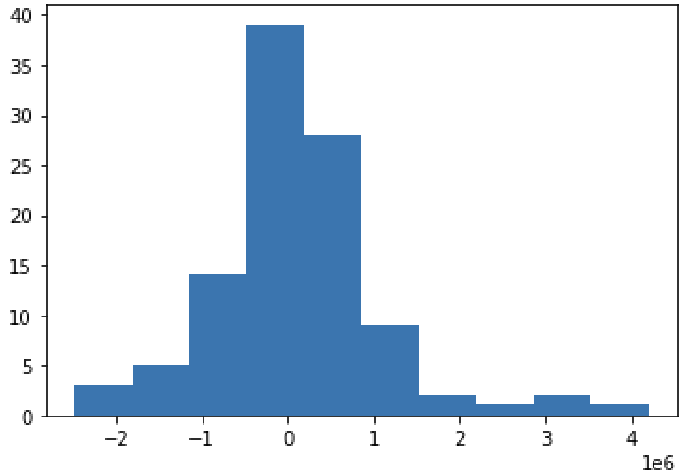


The results are plotted in scatterplot and the identity line is being put to identify whether the price is higher on before or after. If the dots are further away than the line, it means the difference will be higher. Based on the output above, it can be seen that the the numbers of dots on above and below the identity line are equal so it means that the difference between before and after is not big.

Plotting Results in Histogram

```
plot.hist(y_test - y_prediction)
```

```
(array([ 3.,  5., 14., 39., 28.,  9.,  2.,  1.,  2.,  1.]),
 array([-2480069.47 , -1812565.431, -1145061.392,  -477557.353,
          189946.686,   857450.725,  1524954.764,  2192458.803,
         2859962.842,  3527466.881,  4194970.92 ]),
 <BarContainer object of 10 artists>)
```



Based on the histogram above, we can see a normally distributed pattern which indicates that the multiple linear regression model is appropriate to make predictions for the dataset (Mccullum, 2020).

3.1.4 Cross Validation

R-squared (R^2) is a statistical measure use to measure variance of a dependent variable which is explained by one or more independent variables in a regression model. While a correlation will be used to describe the strength of relationship between both independent and independent variable, the variance of one variable's explanation for the variance of the second variable is measured by R-squared (JASON, 2021). Besides that, the cross validation is a statistical technique which will compare and evaluate model by splitting the data into data for training and data for testing with a defined ratio. The most typical way of doing the validation is by using k-fold method where it will separate the data with given k times using the given ratio then a mean of the scoring result will be calculated (Refaeilzadeh, Tang , & Liu , 2009). The calculation of R-squared for X and Y of training data using cross validation method is shown as the figure below.

```
In [35]: from sklearn.model_selection import cross_val_score
         from numpy import mean

         cv = KFold(n_splits=6, random_state=355, shuffle=True)

         cross_val_r2_scores = cross_val_score(regression, x_train, y_train, scoring='r2', cv=cv)
         mean(cross_val_r2_scores)

Out[35]: 0.630490616202016
```

The result of R-squared for training data is 0.63(63%) which is not good enough as the requirement should be at least greater than 70%. Hence the model should be trained with suitable data column selected from the feature selection to improve the result until it greater than 70%

3.1.5 Feature Selection

To improve the performance of the model, the features of the model need to be selected wisely and effective to the regression. Hence the process of feature selection will come in to place, it is a process that play one of the main roles in a model training process where it will be crucial especially when developing predictive model by reducing the number of input variables. The reason of why this process is important including decrease over-fitting the model where it will help the model to reduce the decision made based on noise due to fewer redundant of data, it will also improve the overall accuracy by having fewer misleading data as well as reducing the time model need to be trained since there is less data now (H2O.ai, n.d.).

First and foremost, the optimal number of features is calculated to first define the number of features should be selected to maximise the model performance which is shown as figure below.

```
In [18]: from sklearn.model_selection import KFold
         from sklearn.feature_selection import RFE
         from sklearn.model_selection import GridSearchCV
         from sklearn.linear_model import LinearRegression

         from sklearn.model_selection import train_test_split
         x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size = 0.05,random_state=355)

         folds = KFold(n_splits = 5, shuffle = True, random_state = 100)

         # specify range of hyperparameters to tune
         hyper_params = [{'n_features_to_select': list(range(1, 14))}]

         # specify model
         linear_model = LinearRegression()
         linear_model.fit(x_train, y_train)
         rfe = RFE(linear_model)

         # call GridSearchCV()
         model_cv = GridSearchCV(estimator = rfe,
                                 param_grid = hyper_params,
                                 scoring= 'r2',
                                 cv = folds,
                                 verbose = 1,
                                 return_train_score=True)

         # fit the model
         model_cv.fit(X, Y)

         Fitting 5 folds for each of 13 candidates, totalling 65 fits

Out[18]: GridSearchCV(cv=KFold(n_splits=5, random_state=100, shuffle=True),
                      estimator=RFE(estimator=LinearRegression()),
                      param_grid=[{'n_features_to_select': [1, 2, 3, 4, 5, 6, 7, 8, 9,
                                                            10, 11, 12, 13]}],
                      return_train_score=True, scoring='r2', verbose=1)
```

The original is split to training data and testing data to both rows and columns of the data, then create a linear regression model using the training data to feed it into a cv model to calculate the scoring easily. The result of the scoring is then tabulated and displayed as shown as the figure below.

In [19]:
```
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results
```

Out[19]:

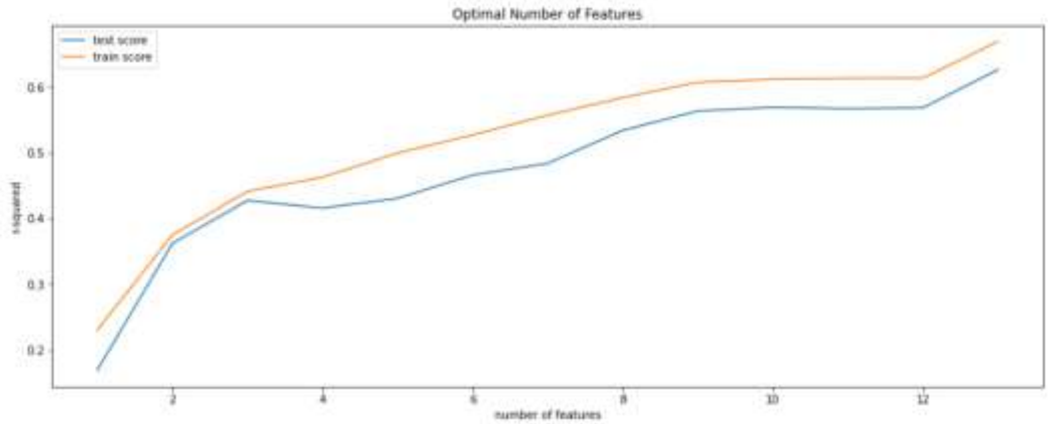
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_features_to_select	params	split0_test_score	split1_test_score	split2_t
0	0.009175	1.933856e-03	0.000991	1.313719e-05	1	{'n_features_to_select': 1}	0.271014	0.055363	
1	0.006981	1.168008e-07	0.006997	2.431402e-07	2	{'n_features_to_select': 2}	0.496025	0.293120	
2	0.006782	1.595426e-03	0.001197	3.989697e-04	3	{'n_features_to_select': 3}	0.577507	0.380712	
3	0.006582	1.196861e-03	0.001396	4.886945e-04	4	{'n_features_to_select': 4}	0.572090	0.374009	
4	0.008389	8.059192e-04	0.001995	6.106495e-07	5	{'n_features_to_select': 5}	0.541882	0.396254	
5	0.006580	1.354179e-03	0.001392	4.915410e-04	6	{'n_features_to_select': 6}	0.565870	0.436488	
6	0.004406	4.739596e-04	0.001002	8.202985e-06	7	{'n_features_to_select': 7}	0.587864	0.461186	
7	0.003989	2.338088e-05	0.001191	4.024829e-04	8	{'n_features_to_select': 8}	0.640155	0.493474	
8	0.004001	1.105757e-03	0.001406	4.893651e-04	9	{'n_features_to_select': 9}	0.665886	0.494916	
9	0.003192	3.983286e-04	0.001198	4.007818e-04	10	{'n_features_to_select': 10}	0.672569	0.494702	
10	0.003798	7.386773e-04	0.001395	4.898085e-04	11	{'n_features_to_select': 11}	0.677692	0.494362	
11	0.002219	3.907199e-04	0.000990	4.339202e-05	12	{'n_features_to_select': 12}	0.677234	0.497477	
12	0.002599	4.839494e-04	0.001790	4.102481e-04	13	{'n_features_to_select': 13}	0.646376	0.597123	

13 rows × 21 columns

To get a better insight from the result, a graph is plot based on the result comparing both test score and train score as shown as the figure below.

In [20]:
```
# plotting cv results
plot.figure(figsize=(16,6))

plot.plot(cv_results["param_n_features_to_select"], cv_results["mean_test_score"])
plot.plot(cv_results["param_n_features_to_select"], cv_results["mean_train_score"])
plot.xlabel('number of features')
plot.ylabel('r-squared')
plot.title("Optimal Number of Features")
plot.legend(['test score', 'train score'], loc='upper left')
```

Out[20]: <matplotlib.legend.Legend at 0x27602bf1760>



From the graph, the conclusion is that the more features is selected from the data, the better the model performance where the model should take all 13 column of the data to have the max score among all.

However, another operation for feature selection is conducted to further filtered some unnecessary features which may affect the model performance. At this point, the indicator named variance inflation factor (VIF) is calculated and used to filter the features. VIF is a measure of the amount of multicollinearity in regression analysis where multicollinearity is refer to correlation happened between multiple independent variables in a multiple regression model. If the VIF of a variable is higher than 5, that means the variable is having multicollinearity and is not suitable to be selected as a feature for the model to be trained because it will affect the model performance (THE INVESTOPEDIA TEAM, 2022). The calculation of VIF for all variables is shown as the figure below.

```
In [21]: from sklearn.preprocessing import MinMaxScaler
         from statsmodels.stats.outliers_influence import variance_inflation_factor
         def preprocessing(X):
             scaler = MinMaxScaler()
             X_scaled = scaler.fit_transform(X)
             variables = X_scaled
             vif = pd.DataFrame()
             vif["VIF"] = [variance_inflation_factor(variables, i) for i in range(variables.shape[1])]
             vif["Features"] = X.columns
             print(vif)

In [22]: preprocessing(X)

         VIF          Features
0   5.695074              area
1   7.370649          bedrooms
2   1.640001         bathrooms
3   2.702247           stories
4   5.841277          mainroad
5   1.521360         guestroom
6   1.998402          basement
7   1.077140   hotwaterheating
8   1.745831   airconditioning
9   1.912748           parking
10  1.444422          prefarea
11  2.306936    semi-furnished
12  1.941835       unfurnished
```

From the calculation result of VIF, there are two variables with VIF greater than 5 found which should be removed from the data for better model performance. The action is performed as shown as the figure below.

```
In [23]: X.drop(['mainroad','bedrooms'], axis=1, inplace=True)
         preprocessing(X)

         VIF          Features
0   4.272647              area
1   1.572188         bathrooms
2   2.134350           stories
3   1.518522         guestroom
4   1.832215          basement
5   1.074235   hotwaterheating
6   1.745076   airconditioning
7   1.873550           parking
8   1.422639          prefarea
9   1.859642    semi-furnished
10  1.545732       unfurnished
```

3.1.6 Overall Accuracy

With the completion of model training, the overall accuracy of the model will be measured from multiple indicators to prove it reach the requirement and is a reliable prediction model.

```
In [32]: MAE = metrics.mean_absolute_error(y_test,y_prediction)
         MAE
Out[32]: 490892.63961538457
```

Absolute error is the size of the discrepancy between the forecast of an observation and its actual value in machine learning. The size of errors for the entire group is determined by Mean Absolute Error (MAE) by averaging the absolute errors for a set of forecasts and observations (c3.ai, n.d.). From the calculation result, it showed that every prediction from the model have the MAE of 490892.64 with the actual observed data.

```
In [33]: from sklearn.metrics import mean_squared_error
         mse = mean_squared_error(y_test, y_prediction)
         print(mse)

         424521475148.489
```

Mean Squared Error (MSE) is an indicator that determine how close a regression line resembles a set of data points. It is a risk function that corresponds to the squared error loss's expected value. The average, more particularly the mean, of errors squared from data related to a function is used to determine mean square error (Gupta, 2022). From the calculation result, it showed that the mean for every data point disperse from the predicted regression is 424521475148.489.

```
In [34]: np.sqrt(metrics.mean_squared_error(y_test, y_prediction))
Out[34]: 651553.1253462675
```

One of the methods most frequently used to assess the accuracy of forecasts is root mean square error (RMSE), also known as root mean square deviation. It illustrates the Euclidean distance between measured true values and forecasts (c3.ai, n.d.). From the calculation result, it showed that the mean for data points disperse from the predicted regression is 424521475148.489.

```
In [36]: from sklearn import metrics
         from sklearn.metrics import r2_score
         r2_score(y_test,y_predict)

Out[36]: 0.7954299575715643
```

Since the model is improved overall, the R-squared of it is tested again to check if it improved in performance and meet the requirement of 70%. The R-squared is calculate using the predicted value and the testing value which get the result of 0.80 (80%) which is proven improved and meeting the requirement.

3.2 Polynomial Regression

Hor Shen Hau (TP061524)

As discussed in the Part 1 of this assignment, Polynomial Regression is chosen to apply the Housing dataset as it may produce better results if the relationship between the attribute and target variable is non-linear.

To build the polynomial regression model, the dataset has to first be cleaned and transformed into a desired shape and format. First and foremost, the developers had removed outliers which had been identified in the previous section.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
# reading the dataset
df = pd.read_csv(r'D:\University Course Materials APU\Year 3 Semester 1\FAI\Group Assignment\housing\Housing.csv');
df.head(10)
```

Out[2]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished
5	10850000	7500	3	3	1	yes	no	yes	no	yes	2	yes	semi-furnished
6	10150000	8580	4	3	4	yes	no	no	no	yes	2	yes	semi-furnished
7	10150000	16200	5	3	2	yes	no	no	no	no	0	no	unfurnished
8	9870000	8100	4	1	2	yes	yes	yes	no	yes	2	yes	furnished
9	9800000	5750	3	2	4	yes	yes	no	no	yes	1	yes	unfurnished

The housing dataset is first read using read_csv function and then the first 10 rows previewed using .head(10) to ensure that the dataset has been read correctly. As there are several outliers that have been pre identified in the previous section, the outliers have to be removed prior to performing any operations in regards to the model. This is because outliers can negatively affect the model's performance, and this is especially the case as the dataset that has been used is very small.

In [3]:
```python
# get outliers

area_IQR = df.area.quantile(.75) - df.area.quantile(.25)
area_outlier = (df.area < (df.area.quantile(.25) - 1.5*area_IQR)) | (df.area > (df.area.quantile(.75) + 1.5*area_IQR))

# select only the ones that are NOT the outliers
df_removed_outlier = df[(~area_outlier)]
df_removed_outlier
```

Out[3]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished
...	...	...	...	...	...	...	...	...	...	...	...	...	...
540	1820000	3000	2	1	1	yes	no	yes	no	no	2	no	unfurnished
541	1767150	2400	3	1	1	no	no	no	no	no	0	no	semi-furnished
542	1750000	3620	2	1	1	yes	no	no	no	no	0	no	unfurnished
543	1750000	2910	3	1	1	no	no	no	no	no	0	no	furnished
544	1750000	3850	3	1	2	yes	no	no	no	no	0	no	unfurnished

533 rows × 13 columns

In [4]:
```python
print(f'Size of original dataset : {df.shape[0]}')
print(f'Size of cleaned dataset : {df_removed_outlier.shape[0]}')

print(f"Percentage removed : {(df.shape[0] - df_removed_outlier.shape[0]) * 100 / df_removed_outlier.shape[0]:.2f}%")
```

```
Size of original dataset : 545
Size of cleaned dataset : 533
Percentage removed : 2.25%
```

2.25% of the dataset has been removed as they are outliers leaving 533 rows of the original 545 in the origin housing dataset. The new dataset is now saved under the variable df_removed_outlier. As the price column is the target variable, there will be no normalization operations performed on it. Instead normalization will be done on the area column as its value ranges are far different from the other numerical data in the dataset.

In [5]:
```python
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

df_removed_outlier['area'] = scaler.fit_transform(df_removed_outlier[['area']])
df_removed_outlier
```

```
C:\Users\horsh\AppData\Local\Temp\ipykernel_72368\1068512545.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  df_removed_outlier['area'] = scaler.fit_transform(df_removed_outlier[['area']])
```

Out[5]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	0.651977	4	2	3	yes	no	no	no	yes	2	yes	furnished
1	12250000	0.825989	4	4	4	yes	no	no	no	yes	3	no	furnished
2	12250000	0.938983	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
3	12215000	0.661017	4	2	2	yes	no	yes	no	yes	3	yes	furnished
4	11410000	0.651977	4	1	2	yes	yes	yes	no	yes	2	no	furnished
...	...	...	...	...	...	...	...	...	...	...	...	...	...
540	1820000	0.152542	2	1	1	yes	no	yes	no	no	2	no	unfurnished
541	1767150	0.084746	3	1	1	no	no	no	no	no	0	no	semi-furnished
542	1750000	0.222599	2	1	1	yes	no	no	no	no	0	no	unfurnished
543	1750000	0.142373	3	1	1	no	no	no	no	no	0	no	furnished
544	1750000	0.248588	3	1	2	yes	no	no	no	no	0	no	unfurnished

533 rows × 13 columns

Once the area column has been normalized, the developer will check for any missing values or negative values present in the dataset as the dataset has been reconstructed with the new normalized area values. MinMaxScaler from sklearn had been used to scale the value of area from 0 to 1.

In [6]:
```
# find any missing value or negative values
df_removed_outlier.describe()
```

Out[6]:

	price	area	bedrooms	bathrooms	stories	parking
count	5.330000e+02	533.000000	533.000000	533.000000	533.000000	533.000000
mean	4.726995e+06	0.376353	2.960600	1.287054	1.808630	0.684803
std	1.851251e+06	0.209634	0.735988	0.500152	0.871953	0.859541
min	1.750000e+06	0.000000	1.000000	1.000000	1.000000	0.000000
25%	3.430000e+06	0.213559	2.000000	1.000000	1.000000	0.000000
50%	4.305000e+06	0.322034	3.000000	1.000000	2.000000	0.000000
75%	5.652500e+06	0.525424	3.000000	2.000000	2.000000	1.000000
max	1.330000e+07	1.000000	6.000000	4.000000	4.000000	3.000000

In [7]:
```
df_removed_outlier.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 533 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   price             533 non-null    int64  
 1   area              533 non-null    float64
 2   bedrooms          533 non-null    int64  
 3   bathrooms         533 non-null    int64  
 4   stories           533 non-null    int64  
 5   mainroad          533 non-null    object 
 6   guestroom         533 non-null    object 
 7   basement          533 non-null    object 
 8   hotwaterheating   533 non-null    object 
 9   airconditioning   533 non-null    object 
 10  parking           533 non-null    int64  
 11  prefarea          533 non-null    object 
 12  furnishingstatus  533 non-null    object 
dtypes: float64(1), int64(5), object(7)
memory usage: 58.3+ KB
```

Next, the developers have chosen dummy encoding to be performed on the categorical data in the dataset namely the ‘furnishingstatus’, ‘prefarea’,’airconditioning’,’basement’,’mainroad’, ‘guestroom’ and ‘hotwaterheating’ columns. This step is necessary as the machine learning algorithm model that the developer has chosen do not support string values as input variables therefore it is necessary to replace these string values with numbers that represent their values accordingly. Dummy encoding method had been used as it converts different string values into separate columns containing either 0 or 1.

In [8]:
```python
cols_to_encode = ["furnishingstatus", "prefarea", "airconditioning", "basement","mainroad","guestroom","hotwaterheating"]

df_encoded = pd.get_dummies(df_removed_outlier, columns=cols_to_encode, drop_first=True)
df_encoded
```

Out[8]:

	price	area	bedrooms	bathrooms	stories	parking	furnishingstatus_semi-furnished	furnishingstatus_unfurnished	prefarea_yes	airconditioning_yes	basem
0	13300000	0.651977	4	2	3	2	0	0	1	1	
1	12250000	0.825989	4	4	4	3	0	0	0	1	
2	12250000	0.938983	3	2	2	2	1	0	1	0	
3	12215000	0.661017	4	2	2	3	0	0	1	1	
4	11410000	0.651977	4	1	2	2	0	0	0	1	
...	...	...	...	...	...	...	...	...	...	...	
540	1820000	0.152542	2	1	1	2	0	1	0	0	
541	1767150	0.084746	3	1	1	0	1	0	0	0	
542	1750000	0.222599	2	1	1	0	0	1	0	0	
543	1750000	0.142373	3	1	1	0	0	0	0	0	
544	1750000	0.248588	3	1	2	0	0	1	0	0	

533 rows × 14 columns

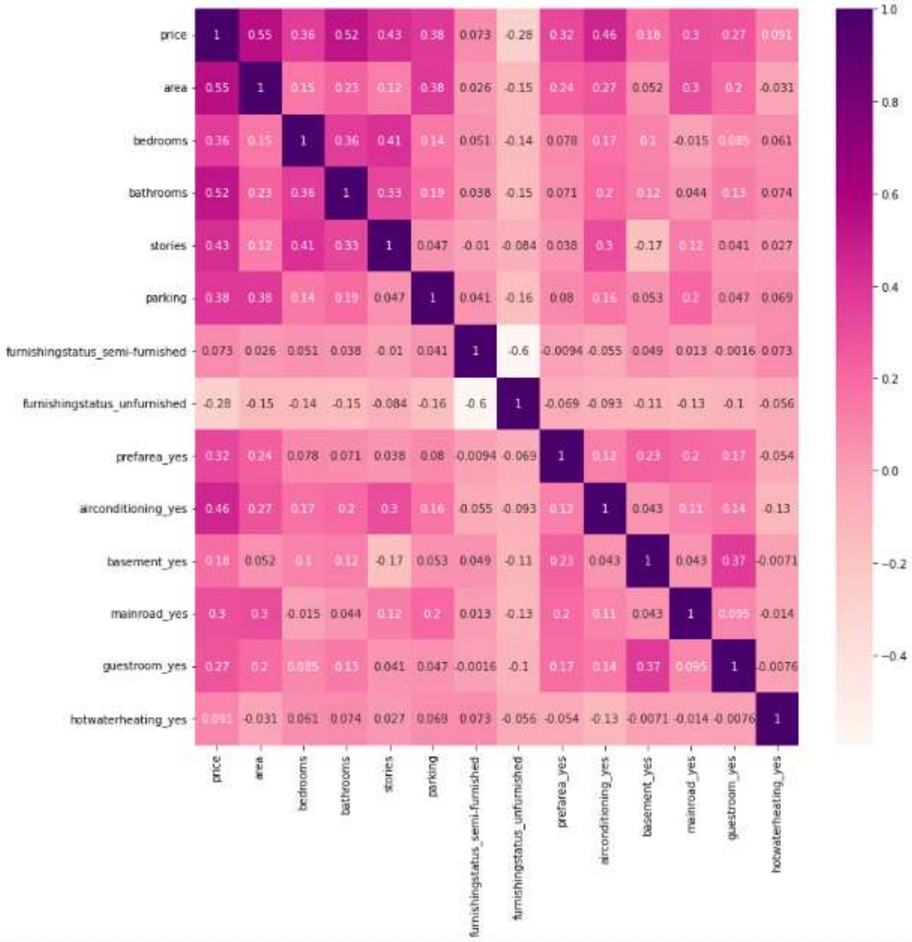
In [9]:
```python
df_encoded.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 533 entries, 0 to 544
Data columns (total 14 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   price                            533 non-null    int64  
 1   area                             533 non-null    float64
 2   bedrooms                         533 non-null    int64  
 3   bathrooms                        533 non-null    int64  
 4   stories                          533 non-null    int64  
 5   parking                          533 non-null    int64  
 6   furnishingstatus_semi-furnished  533 non-null    uint8  
 7   furnishingstatus_unfurnished     533 non-null    uint8  
 8   prefarea_yes                     533 non-null    uint8  
 9   airconditioning_yes              533 non-null    uint8  
 10  basement_yes                     533 non-null    uint8  
 11  mainroad_yes                     533 non-null    uint8  
 12  guestroom_yes                    533 non-null    uint8  
 13  hotwaterheating_yes              533 non-null    uint8  
dtypes: float64(1), int64(5), uint8(8)
memory usage: 33.3 KB
```

As can be seen in the screencapture above, the column's data which were previously string values are now populated with numerical values representative of their original string values in the respective columns.

Yan Mun Kye (TP056066)

```
In [12]: import seaborn as sns
         plt.figure(figsize=[12,12])
         sns.heatmap(df_encoded.corr(), annot=True,cmap=plt.cm.RdPu)

Out[12]: <AxesSubplot:>
```



A correlation heatmap is then used to visualize the correlation between the different variables of the dataset. With the heatmap, it can be seen that there are some variables with high correlation with price which are ‘area’, ‘bathrooms’,’airconditioning_yes’,’stories’ and ‘parking’. These variables will then be used in the training and testing of the model.

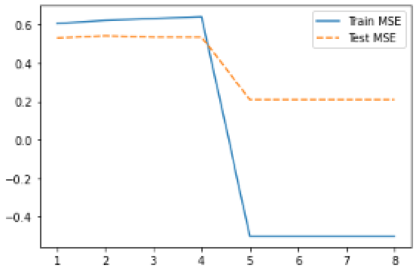
```
In [13]: from sklearn.model_selection import train_test_split

         # from the heatmap above, we can see there are some terms with high correlation with price. We will use those terms

         high_corr = ['area','bathrooms', 'airconditioning_yes','stories', 'parking']

         X = df_encoded[high_corr]
         y = df_encoded['price']

         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.1, random_state=1000)

         print(X_train.shape)
         print(y_train.shape)
         print(X_test.shape)
         print(y_test.shape)

         (479, 5)
         (479,)
         (54, 5)
         (54,)
```

The train and test dataset is split using the train_test_split library from sklearn. The train test split ratio is 90% training and 10% testing as the dataset itself is relatively small. This will allow more data to be used for training which enables the model to learn the patterns from more data points.

In [14]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
# we will do polynomial regression. First, we must find the degree which has the best result
max_degree = 9
xaxis = []
prediction_score=[]
train_score = []
for i in range(1,max_degree):
    p = PolynomialFeatures(degree=i, interaction_only=True)
    lm = LinearRegression()
    X_poly = p.fit_transform(X_train)
    X_poly_test = p.fit_transform(X_test)
    lm.fit(X_poly, y_train)
    y_pred = lm.predict(X_poly_test)
    prediction_score.append(r2_score(y_test, y_pred))
    train_score.append(r2_score(y_train,lm.predict(X_poly)))
    xaxis.append(i)

best_index =prediction_score.index(max(prediction_score))
print("Best degree is", xaxis[best_index])

f,ax = plt.subplots(1,1)
sns.lineplot(data=pd.DataFrame({'Train MSE':train_score, 'Test MSE':prediction_score}, index=xaxis), ax=ax, color='blue')
ax.legend()

# sns.lineplot(x=xaxis, y=train_score)
```

```
Best degree is 2
```

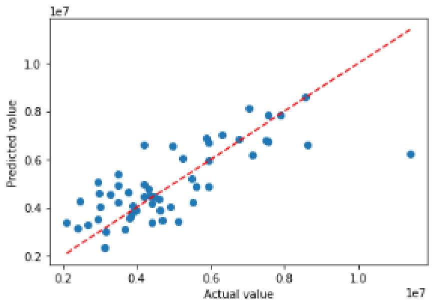
Out[14]: <matplotlib.legend.Legend at 0x2a11f6a6b50>



From the plot above,we can see that the test MSE for degree of 2 is lowest. Therefore we will use degree=2.

After splitting the dataset into x_train, x_test, y_train, y_test, the PolynomialFeatures and LinearRegression model is imported from sklearn. Due to the presence of a hyperparameter degree in polynomial regression, it is necessary for the developer to fine tune the hyperparameter and determine the best degree. Here the best prediction score is obtained and the degree that provides that score is determined as the best degree. In this case the reported optimal degree to be used is 2 as shown in the code output. The MSE plot of degree 2 for both train and test is plotted and displayed.

```
In [15]: from sklearn.model_selection import cross_val_score


         poly_transformer = PolynomialFeatures(degree=2)
         X_train_poly = poly_transformer.fit_transform(X_train)
         X_test_poly = poly_transformer.fit_transform(X_test)

         model = LinearRegression()
         cv = cross_val_score(model, X_train_poly, y_train, scoring='r2', cv=10)
         print(f"Cross validated train R2 : ",np.mean(cv))

         model.fit(X_train_poly, y_train)
         y_pred = model.predict(X_test_poly)
         print(f'Test MSE : {mean_squared_error(y_test, y_pred)}')
         print(f'Test R2 : {r2_score(y_test, y_pred)}')

         Cross validated train R2 :  0.576900634580267
         Test MSE : 1490002688421.702
         Test R2 : 0.5681617494574471

In [16]: plt.scatter(y_test, y_pred)
         plt.plot([y_test.min(),y_test.max()],[y_test.min(),y_test.max()], 'r--')
         plt.xlabel("Actual value")
         plt.ylabel("Predicted value")

Out[16]: Text(0, 0.5, 'Predicted value')
```



The Polynomial Regression model is the trained with X_train and y_train by first transforming X_train into polynomial features of degree 2. The hyperparameter degree of 2 had been determined from the step above to be the best performing. The model had been cross validated using the training data using 10-fold cross validation to ensure the results obtained was not by the specific random distribution of the train-test-split. R^2 had been used as a performance metric as R^2 describes how much the model is able to explain the variance in the target variable by using the individual variables.

Finally, the model is validated using the test dataset, in which the data had not been seen by the model during the training. The final R^2 score of the polynomial regression model is 56.8%, while the final MSE is 1,490,002,688,421.702. However, MSE cannot be used as an general indicator of model performance as different problems will have different range of MSE. The R^2 is actually too low for any real world application as only a little more than half of the variance in the prediction is explained by the model. This might have happened due to several reasons. First reason is the dataset is relatively small, which does not allow the model to learn the pattern of the data properly. Secondly, the pattern of the data is better described with higher order curves or linear lines, as polynomial regression assumes a shape on the data. In this case, the shape of the data is assumed to be of degree 2, which is quadratic. If the shape of the data does not match the assumed shape of the polynomial regression, the model will not be able to predict values accurately.

References

JASON, F. (12nd September, 2021). *Investopedia.* Retrieved from R-Squared Formula, Regression, and Interpretations: https://www.investopedia.com/terms/r/r-squared.asp

c3.ai. (n.d.). *c3.ai*. Retrieved from What is Mean Absolute Error (MAE)?: https://c3.ai/glossary/data-science/mean-absolute-error/#:~:text=What%20is%20Mean%20Absolute%20Error,true%20value%20of%20that%20observation.

c3.ai. (n.d.). *c3.ai*. Retrieved from Root Mean Square Error (RMSE): https://c3.ai/glossary/data-science/root-mean-square-error-rmse/#:~:text=What%20is%20Root%20Mean%20Square,true%20values%20using%20Euclidean%20distance.

Gupta, A. (28th September, 2022). *SimpliLearn*. Retrieved from Mean Squared Error : Overview, Examples, Concepts and More: https://www.simplilearn.com/tutorials/statistics-tutorial/mean-squared-error#:~:text=The%20Mean%20Squared%20Error%20measures,it%20relates%20to%20a%20function.

H2O.ai. (n.d.). *H2O.ai*. Retrieved from Feature Selection: https://h2o.ai/wiki/feature-selection/#:~:text=Why%20is%20Feature%20Selection%20important,the%20redundant%20and%20irrelevant%20ones.

Mccullum, N. (2020). How to Build and Train Linear and Logistic Regression ML Models in Python. https://www.freecodecamp.org/news/how-to-build-and-train-linear-and-logistic-regression-ml-models-in-python/.

Refaeilzadeh, P., Tang , L., & Liu , H. (2009). *Cross-Validation.* Boston: Springer, Boston, MA.

THE INVESTOPEDIA TEAM. (26th July, 2022). *Investopedia*. Retrieved from Variance Inflation Factor (VIF): https://www.investopedia.com/terms/v/variance-inflation-factor.asp#:~:text=A%20variance%20inflation%20factor%20(VIF)%20is%20a%20measure%20of%20the,adversely%20affect%20the%20regression%20results.

Workload Matrix

Workload	Responsible Students
Application Demo	Yan Mun Kye
Evaluation of Model	Yan Mun Kye & Hor Shen Hau & Tan Sheng Jeh & Sia De Long
Multiple Linear Regression • Data Preprocessing • Data Splitting • Data Trainning	Tan Sheng Jeh
Multiple Linear Regression • Cross Validation • Feature Selection • Overall Accuracy	Sia De Long
Polynomial Regression	Yan Mun Kye & Hor Shen Hau