Forecasting the Return on Investment (ROI) for a New House in Melbourne: A Five-year Sales Regression Analysis

BDA & DaViz Group Member:

2540135473 – Dora Kalifa Dharmawan – 30%

2501983105 – Pristian Budi Dharmawan – 40%

2540131310 – Pirelli Rahelya Piri – 30%

I. Business Understanding

A real estate company just got an investment of $100 million. This investment aims to construct several new home clusters with a target Return on Investment (ROI) of around $200 thousand for each unit sold. As we know, a property's cost is increasing yearly by around 7.9% per year in Melbourne (propertyupdate.com).

This company has difficulties developing home clusters for certain areas in Melbourne. Their target market is the new family or a family with a medium- to high-class monetary level. They wanted to open these home clusters with a low population density, a good environment, and the nearest to the centre of the city. They took a dataset from 2017 to 2018 to see which region has the highest ROI in Melbourne. However, there are several limitations to building a home cluster, such as the law, construction costs, land availability, etc.

With those criteria, the company assumes that they will be able to overcome their limitations in this project. To increase their ROI, they could build certain facilities in their area, bundle packages, sell furnished or unfurnished houses, etc. Other than that, to make their forecasting of ROI more valid, they wanted to use the dataset that had already been received.

However, they are unclear about the dataset they received. Thus, this company hired a data scientist to process the data and give them the best advice on which region they should construct to have the highest ROI within five years of analysis. The data scientist suggested that to build a forecasting model, he wanted to conduct regression analysis by leveraging several regression models, such as Multiple Linear Regression (MLR), Lasso Regression (LR), and Random Forest Regression (RFR).

II. Data Understanding

The dataset of this project can be accessed through the link below:

Melbourne Housing Market or Click Here

This dataset contains the following attributes:

• Suburb • Address • Rooms • Price • Method • Type • SellerG • Date • Distance • Postcode • Bedroom2	• Bathroom • Car • Landsize • BuildingArea • YearBuild • CouncilArea • Lattitude • Longitude • Regionname • Propertycount

The summary of the dataset using Python .describe and .info is shown below:

```
df.describe(include='all').transpose()
```

[3] ✓ 0.1s

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
Suburb	34857	351	Reservoir	844	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Address	34857	34009	5 Charles St	6	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Rooms	34857.0	NaN	NaN	NaN	3.031012	0.969933	1.0	2.0	3.0	4.0	16.0
Type	34857	3	h	23980	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Price	27247.0	NaN	NaN	NaN	1050173.344955	641467.130105	85000.0	635000.0	870000.0	1295000.0	11200000.0
Method	34857	9	S	19744	NaN	NaN	NaN	NaN	NaN	NaN	NaN
SellerG	34857	388	Jellis	3359	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Date	34857	78	28/10/2017	1119	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Distance	34856.0	NaN	NaN	NaN	11.184929	6.788892	0.0	6.4	10.3	14.0	48.1
Postcode	34856.0	NaN	NaN	NaN	3116.062859	109.023903	3000.0	3051.0	3103.0	3156.0	3978.0
Bedroom2	26640.0	NaN	NaN	NaN	3.084647	0.98069	0.0	2.0	3.0	4.0	30.0
Bathroom	26631.0	NaN	NaN	NaN	1.624798	0.724212	0.0	1.0	2.0	2.0	12.0
Car	26129.0	NaN	NaN	NaN	1.728845	1.010771	0.0	1.0	2.0	2.0	26.0
Landsize	23047.0	NaN	NaN	NaN	593.598993	3398.841946	0.0	224.0	521.0	670.0	433014.0
BuildingArea	13742.0	NaN	NaN	NaN	160.2564	401.26706	0.0	102.0	136.0	188.0	44515.0
YearBuilt	15551.0	NaN	NaN	NaN	1965.289885	37.328178	1196.0	1940.0	1970.0	2000.0	2106.0
CouncilArea	34854	33	Boroondara City Council	3675	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Lattitude	26881.0	NaN	NaN	NaN	-37.810634	0.090279	-38.19043	-37.86295	-37.8076	-37.7541	-37.3902
Longtitude	26881.0	NaN	NaN	NaN	145.001851	0.120169	144.42379	144.9335	145.0078	145.0719	145.52635
Regionname	34854	8	Southern Metropolitan	11836	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Propertycount	34854.0	NaN	NaN	NaN	7572.888306	4428.090313	83.0	4385.0	6763.0	10412.0	21650.0

```
df.info()
```

[4] ✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34857 entries, 0 to 34856
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Suburb         34857 non-null  object
 1   Address        34857 non-null  object
 2   Rooms          34857 non-null  int64
 3   Type           34857 non-null  object
 4   Price          27247 non-null  float64
 5   Method         34857 non-null  object
 6   SellerG        34857 non-null  object
 7   Date           34857 non-null  object
 8   Distance       34856 non-null  float64
 9   Postcode       34856 non-null  float64
 10  Bedroom2       26640 non-null  float64
 11  Bathroom       26631 non-null  float64
 12  Car            26129 non-null  float64
 13  Landsize       23047 non-null  float64
 14  BuildingArea   13742 non-null  float64
 15  YearBuilt      15551 non-null  float64
 16  CouncilArea    34854 non-null  object
 17  Lattitude      26881 non-null  float64
 18  Longtitude     26881 non-null  float64
 19  Regionname     34854 non-null  object
 20  Propertycount  34854 non-null  float64
dtypes: float64(12), int64(1), object(8)
memory usage: 5.6+ MB
```

III. DATA PREPROCESSING

We decided to choose the "Melbourne_housing_FULL.csv" dataset to be analyzed using Multiple Linear Regression (MLR). In this section, we will preprocess the dataset for MLR using the Ordinary Least Square Method. The difference between the uncleaned dataset and the cleaned dataset can be seen below:

Dirty Dataset

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34857 entries, 0 to 34856
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Suburb         34857 non-null  object
 1   Address        34857 non-null  object
 2   Rooms          34857 non-null  int64
 3   Type           34857 non-null  object
 4   Price          27247 non-null  float64
 5   Method         34857 non-null  object
 6   SellerG        34857 non-null  object
 7   Date           34857 non-null  object
 8   Distance       34856 non-null  float64
 9   Postcode       34856 non-null  float64
 10  Bedroom2       26640 non-null  float64
 11  Bathroom       26631 non-null  float64
 12  Car            26129 non-null  float64
 13  Landsize       23047 non-null  float64
 14  BuildingArea   13742 non-null  float64
 15  YearBuilt      15551 non-null  float64
 16  CouncilArea    34854 non-null  object
 17  Lattitude      26881 non-null  float64
 18  Longtitude     26881 non-null  float64
 19  Regionname     34854 non-null  object
 20  Propertycount  34854 non-null  float64
dtypes: float64(12), int64(1), object(8)
memory usage: 5.6+ MB
```

Cleaned Dataset

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29238 entries, 0 to 29237
Data columns (total 19 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Suburb         29238 non-null  category
 1   Rooms          29238 non-null  int64
 2   Type           29238 non-null  category
 3   Method         29238 non-null  category
 4   SellerG        29238 non-null  category
 5   Date           29238 non-null  category
 6   Distance       29238 non-null  float64
 7   Postcode       29238 non-null  category
 8   Bathroom       29238 non-null  float64
 9   Car            29238 non-null  float64
 10  Landsize       29238 non-null  float64
 11  BuildingArea   29238 non-null  float64
 12  YearBuilt      29238 non-null  float64
 13  CouncilArea    29238 non-null  category
 14  Lattitude      29238 non-null  float64
 15  Longtitude     29238 non-null  float64
 16  Regionname     29238 non-null  category
 17  Propertycount  29238 non-null  float64
 18  log_price      29238 non-null  float64
dtypes: category(8), float64(10), int64(1)
memory usage: 2.8 MB
```

It might be hard to see the differences, thus we made the full documentation of the data preprocessing in our GitHub below:

Full Documentation of the Melbourne Housing Market Regression Analysis

OLS Method

OLS Method is a statistical method for estimating new data or parameters in a linear regression model by minimizing the Sum of Squared Error (SSE). We could explore the determinants of a good regression using the ANOVA framework, which are:

1. Sum of Squares Total (SST)
   It's the sum of the squared differences between the observed dependent variable and its mean. The formula to calculate SST is as follows:
   $$SST = \sum_{i=1}^{n}(y_i - \bar{y})^2$$

2. Sum of Squares Regression (SSR)
   The sum of the squared differences between each predicted value and its mean. The formula of SSR is:
   $$SSR = \sum_{i=1}^{n}(\hat{y} - \bar{y})^2$$

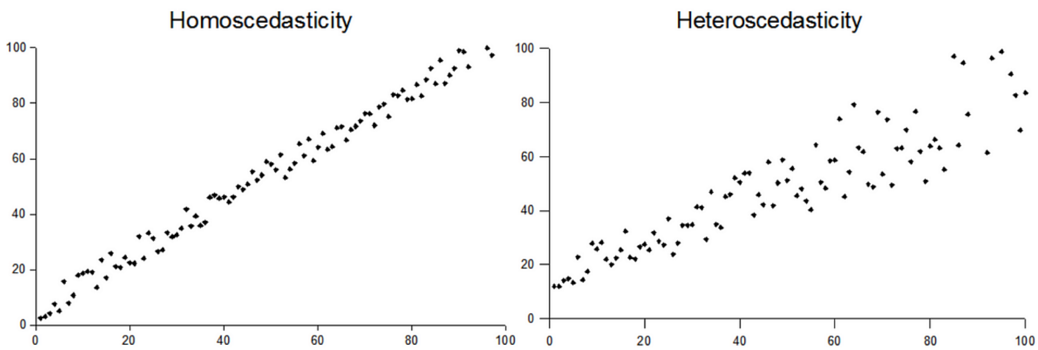
3. Sum of Squares Error (SSE)
   While SSE is the sum of the squared differences between the observed dependent variable and the predicted value. The formula of SSE is:
   $$SSE = \sum_{i=1}^{n}(y_i - \hat{y})^2$$

However, since we want to measure the differences between the predicted value and actual value, to measure the performance of MLR using the Mean Absolute Error (MAE), Mean Squared Error (MSE), Mean Absolute Percentage Error (MAPE), and Root Mean Squared Error (RMSE). But, if we want to measure how well the model fits the data, we could calculate it using the R-squared and Adjusted R-squared which are as follows:

- R-squared
  $R^2 = \frac{SSR}{SST}$
- Adjusted R-squared
  $Adj.R^2 = 1 - (1 - R^2) * \frac{n-1}{n-p-1}$

In MLR using the OLS method, we must obey the following assumptions:

1. Linearity
   This means that all the independent variable and dependent variable is linear. If this assumption is violated, we could use exponential transformation, log transformation, and non-linear regression.
2. No Endogeneity
   It refers to the prohibition of a link between the independent variables and the errors. This issue could lead to variable bias. For example, when we want to predict the house price, the bigger the house, the cheaper it is. This example does not make any sense right?
3. Normality and Homoscedasticity
   a. It is assuming the error of the dataset is normally distributed, however, if it's not we could apply the Central Limit Theorem (CLT).
   b. While homoscedasticity refers to having an equal variance. The illustration of homoscedasticity and heteroscedasticity can be seen as follows:



4. No Autocorrelation
   It refers to "The Day-of-The-Week Effect", it's simply about the correlation of error terms with each other over time.

5. No Multicollinearity
   Multicollinearity occurs whenever two or more variables have a high correlation. For example,

$$a = 2 + 5 * b \qquad b = \frac{a-2}{5}$$

$$\rho_{ab} = 1 \quad \text{perfect multicollinearity}$$

Data Cleaning

1. Data Type Cleaning

   In this phase, we will clean the data type such as `object` to `category`, `date` to `datetime`, and `Postcode` to `category`. This phase will result in this dataset as follows:

```
dataset.info()
✓ 0.0s

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34857 entries, 0 to 34856
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Suburb         34857 non-null  category
 1   Address        34857 non-null  category
 2   Rooms          34857 non-null  int64
 3   Type           34857 non-null  category
 4   Price          27247 non-null  float64
 5   Method         34857 non-null  category
 6   SellerG        34857 non-null  category
 7   Date           34857 non-null  category
 8   Distance       34856 non-null  float64
 9   Postcode       34856 non-null  category
 10  Bedroom2       26640 non-null  float64
 11  Bathroom       26631 non-null  float64
 12  Car            26129 non-null  float64
 13  Landsize       23047 non-null  float64
 14  BuildingArea   13742 non-null  float64
 15  YearBuilt      15551 non-null  float64
 16  CouncilArea    34854 non-null  category
 17  Lattitude      26881 non-null  float64
 18  Longtitude     26881 non-null  float64
 19  Regionname     34854 non-null  category
 20  Propertycount  34854 non-null  float64
dtypes: category(9), float64(11), int64(1)
memory usage: 5.0 MB
```

2. Duplicate Columns
   Two columns seem ambiguous; thus, we need to compare their differences as follows:

```
dataset['Duplicate Columns'] = dataset['Rooms'] - dataset['Bedroom2']
print("Rooms VS Bedroom2 (Average): ", dataset['Duplicate Columns'].mean())
print("Rooms VS Bedroom2 (Median): ", dataset['Duplicate Columns'].median())
dataset.head(10)
[10] ✓ 0.0s

Rooms VS Bedroom2 (Average):  0.016253753753753753
Rooms VS Bedroom2 (Median):  0.0
```

   As it's shown above, we decided to remove the `Bedroom2` feature.

3. Missing Values

Several columns have missing values, such as follows:

- Price
- Bathroom
- Car
- Landsize
- BuildingArea
- YearBuilt
- CouncilArea
- Lattitude
- Longtitude
- Regionname
- Propertycount

```
# Summary of missing values
dataset.isnull().sum()
✓ 0.0s

Suburb              0
Address             0
Rooms               0
Type                0
Price            7610
Method              0
SellerG             0
Date                0
Distance            1
Postcode            1
Bathroom         8226
Car              8728
Landsize        11810
BuildingArea    21115
YearBuilt       19306
CouncilArea         3
Lattitude        7976
Longtitude       7976
Regionname          3
Propertycount       3
dtype: int64
```

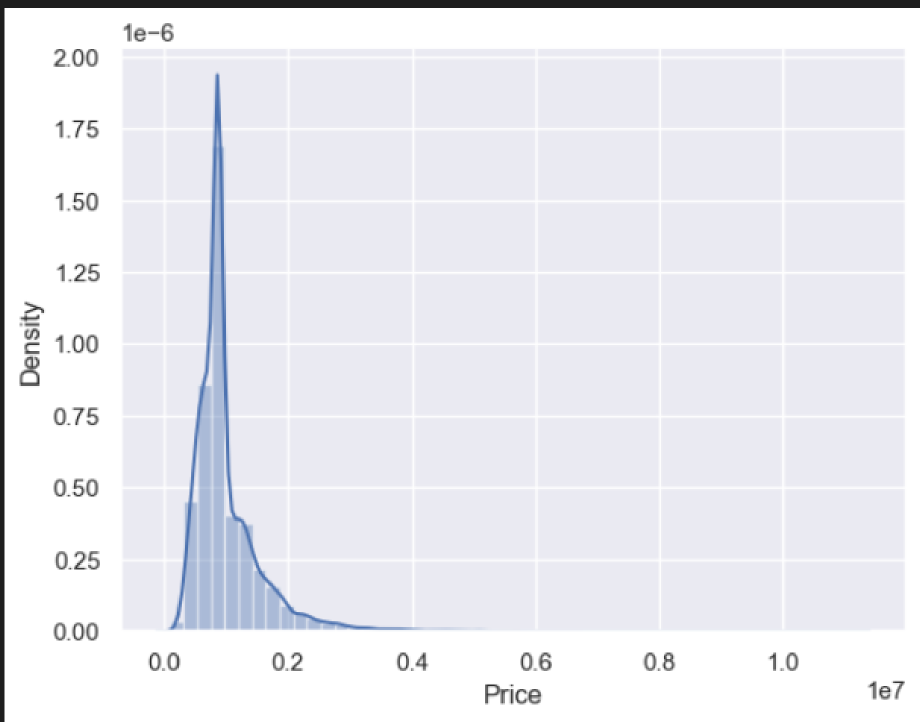
However, we will take the Central Tendency value using the median to fill them out, except for several columns, which are:

- Car
- CouncilArea
- Lattitude
- Longtitude
- Regionname

4. Outliers

We will convert the Probability Distribution Function for each feature to be normally distributed, as follows:

```
... <Axes: xlabel='Price', ylabel='Density'>
...
```

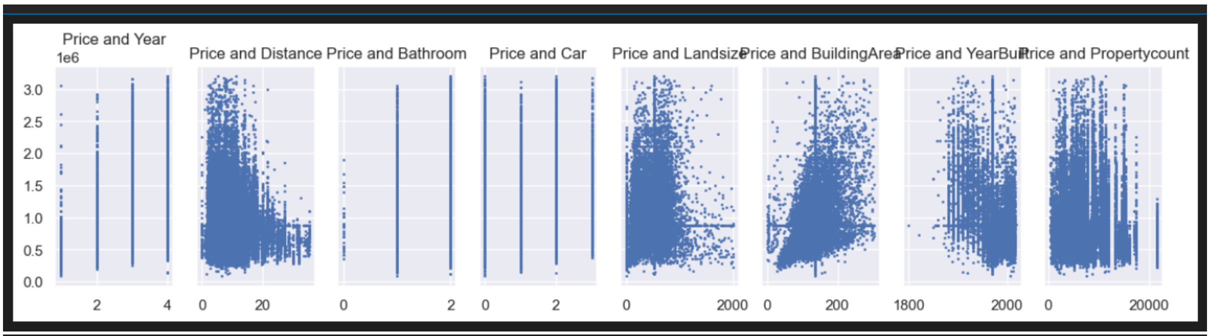


Thus, I need to clean all these numeric values by taking its 1% top quantile to be removed. This phase will help us in Homoscedasticity. In this phase, we will remove any value that is still odd.

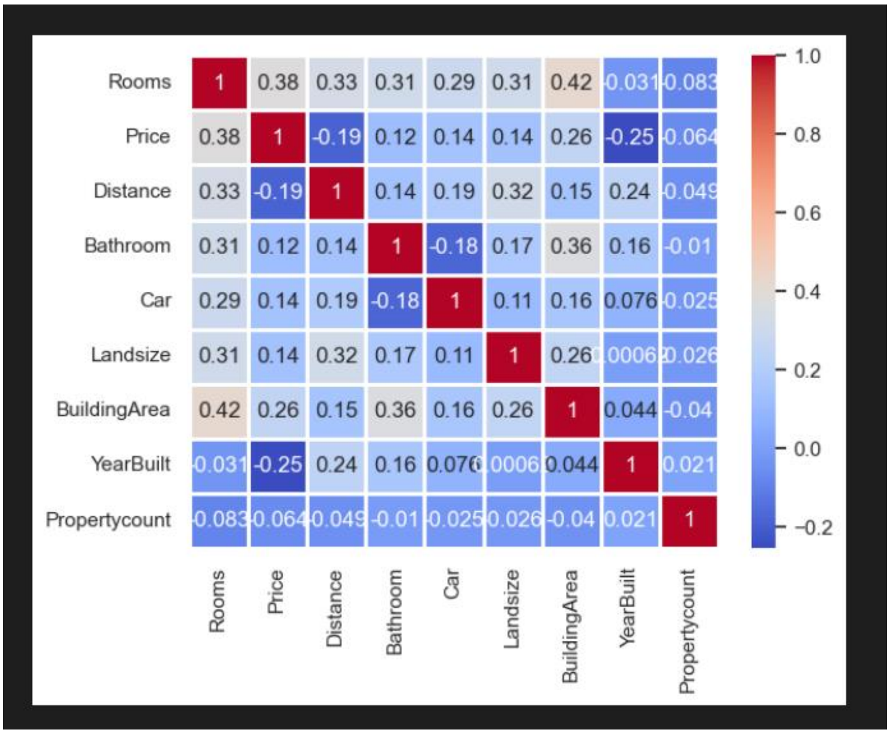
5. Data Reduction
   In this section, we will just remove the `Address` column to make this analysis more comprehensive.

OLS Assumptions Cleaning

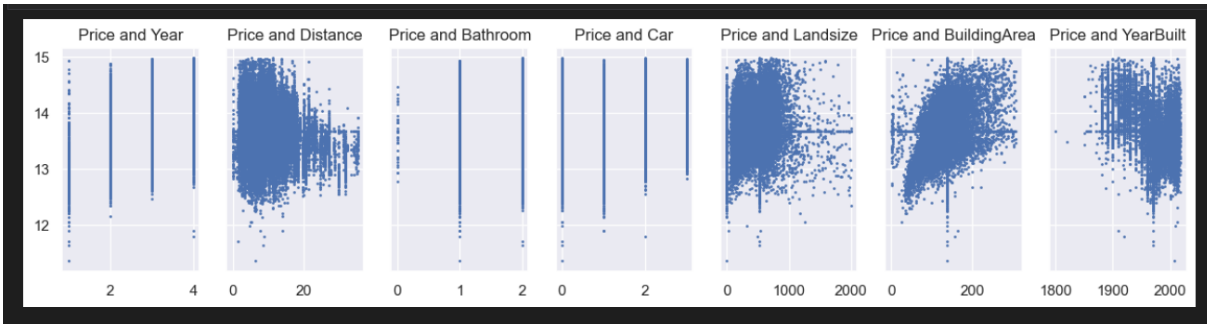
1. Linearity
   To check the linearity of `Price` with another variables, we could plot them using a scatter plot as follows:



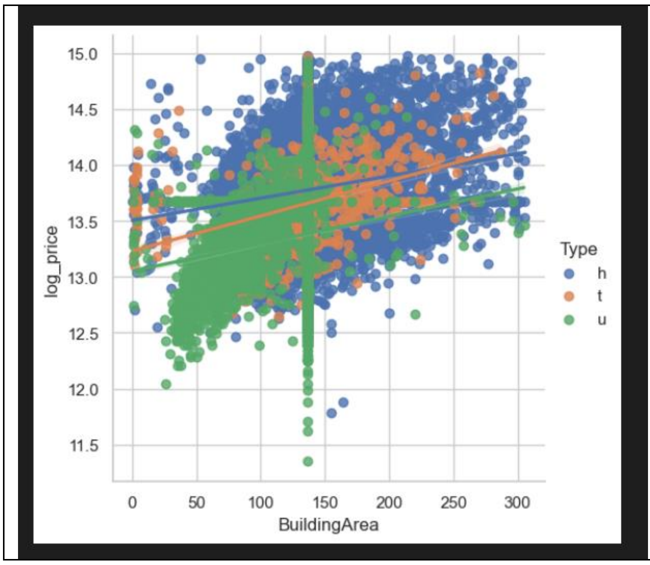
Other than that, we could find the correlation between each variable using the heatmap plot, as follows:



Based on the plot above, we can determine that this data is not linear, thus we will try to transform it using Log Transformation and below is the result:



Now we want to see using the category variable, which is the type of the property:



Type:

br - bedroom(s);

h - house,cottage,villa, semi,terrace;

u - unit, duplex;

t - townhouse;

dev site - development site;

o res - other residential.

2. No Endogeneity
   Based on the plot above, we could also determine that `Price` and `Distance` are endogenous. However, we will just include this feature since it has a big correlation with the price of the property.
3. Normality and Homoscedasticity
   We couldn't do much with non-linear data to obey this assumption, thus we will just let it be.
4. No Autocorrelation
   This data is not a time series or panel data, which means that this data has already obeyed this assumption.
5. No Multicollinearity
   This dataset also has multicollinearity, for example, we know that `Rooms` and `Bathroom` are correlated and there is no way that `Bathroom` value exceeds the value of `Rooms`. However, we could check this issue by finding the Variance Inflation Factors from the continuous variable, such as follows:
   - Distance
   - Landsize
   - BuildingArea
   - YearBuilt

   We could not accept if the VIF is more than 5, but we will still include them in our analysis.

	VIF	Features
0	4.571698	Distance
1	5.394516	Landsize
2	20.140854	BuildingArea
3	20.382215	YearBuilt

Dummy Variables

In this phase, we will take all of the data with a category data type to be encoded using One-Hot Encoding. This encoding will help us in our regression analysis further.

```python
new_dataset = pd.get_dummies(dataset, drop_first=True, dtype=int)
new_dataset
```

[22] Python

	Rooms	Distance	Bathroom	Car	Landsize	BuildingArea	YearBuilt	Lattitude	Longtitude	Propertycount	...	CouncilArea_Yarra City Council	CouncilArea_Yarra Ranges Shire Council	Regionname_Eastern Victoria	Regionname_MV	Regionname_Nort Metropo
0	2	2.5	1.0	1.0	126.0	136.0	1970.0	-37.80140	144.99580	4019.0	...	1	0	0	0	
1	2	2.5	1.0	1.0	202.0	136.0	1970.0	-37.79960	144.99840	4019.0	...	1	0	0	0	
2	2	2.5	1.0	0.0	156.0	79.0	1900.0	-37.80790	144.99340	4019.0	...	1	0	0	0	
3	3	2.5	2.0	1.0	0.0	136.0	1970.0	-37.81140	145.01160	4019.0	...	1	0	0	0	
4	3	2.5	2.0	0.0	134.0	150.0	1900.0	-37.80930	144.99440	4019.0	...	1	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
29233	4	6.3	1.0	3.0	593.0	136.0	1970.0	-37.81053	144.88467	6543.0	...	0	0	0	0	
29234	2	6.3	2.0	1.0	98.0	104.0	2018.0	-37.81551	144.88826	6543.0	...	0	0	0	0	
29235	2	6.3	1.0	2.0	220.0	120.0	2000.0	-37.82286	144.87856	6543.0	...	0	0	0	0	
29236	3	6.3	2.0	0.0	521.0	136.0	1970.0	0.00000	0.00000	6543.0	...	0	0	0	0	
29237	2	6.3	1.0	0.0	250.0	103.0	1930.0	-37.81810	144.89351	6543.0	...	0	0	0	0	

29238 rows × 991 columns

Additional (Neural Network)

While in Neural Network, we will just take the cleaned dataset (before OLS Assumption Cleaning) convert the `Postcode` into an object data type and encode the category dataset using One-Hot Encoding.

NOTE

However, all these datasets are not yet standardized. Thus, we will standardize this dataset using sklearn.StandardScaler() which scales the data using Z-score.

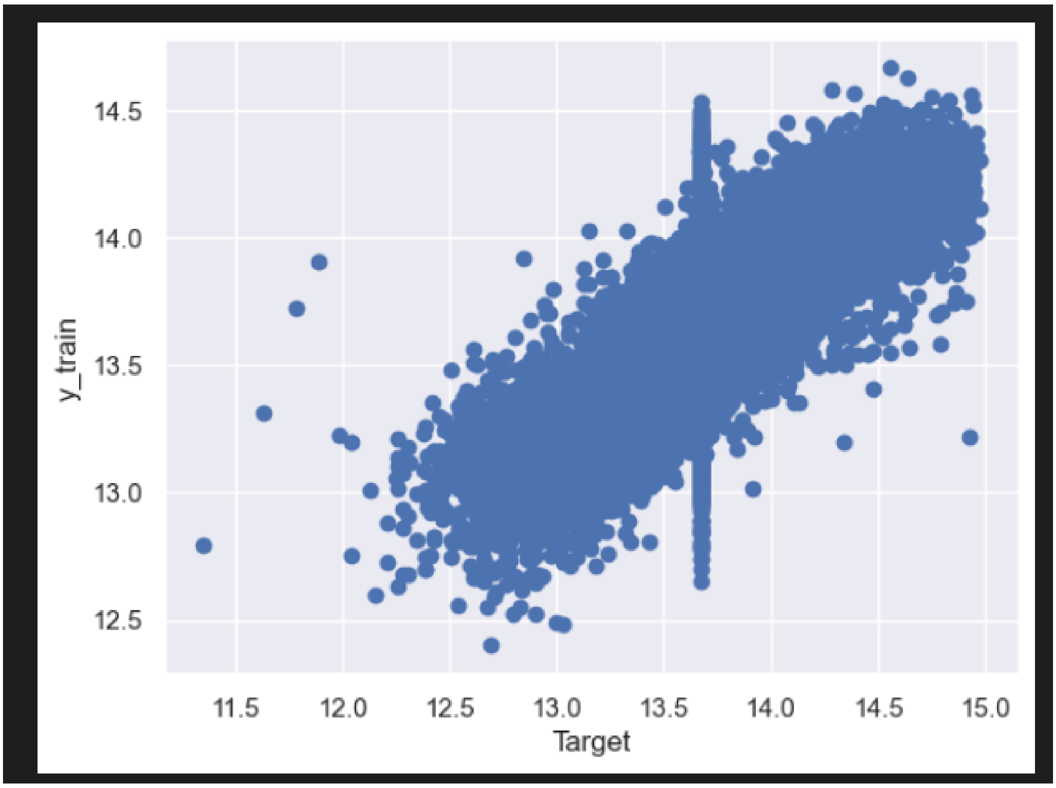
IV. Model Implementation

We have already documented this implementation using both models, which can be accessed from the link below:

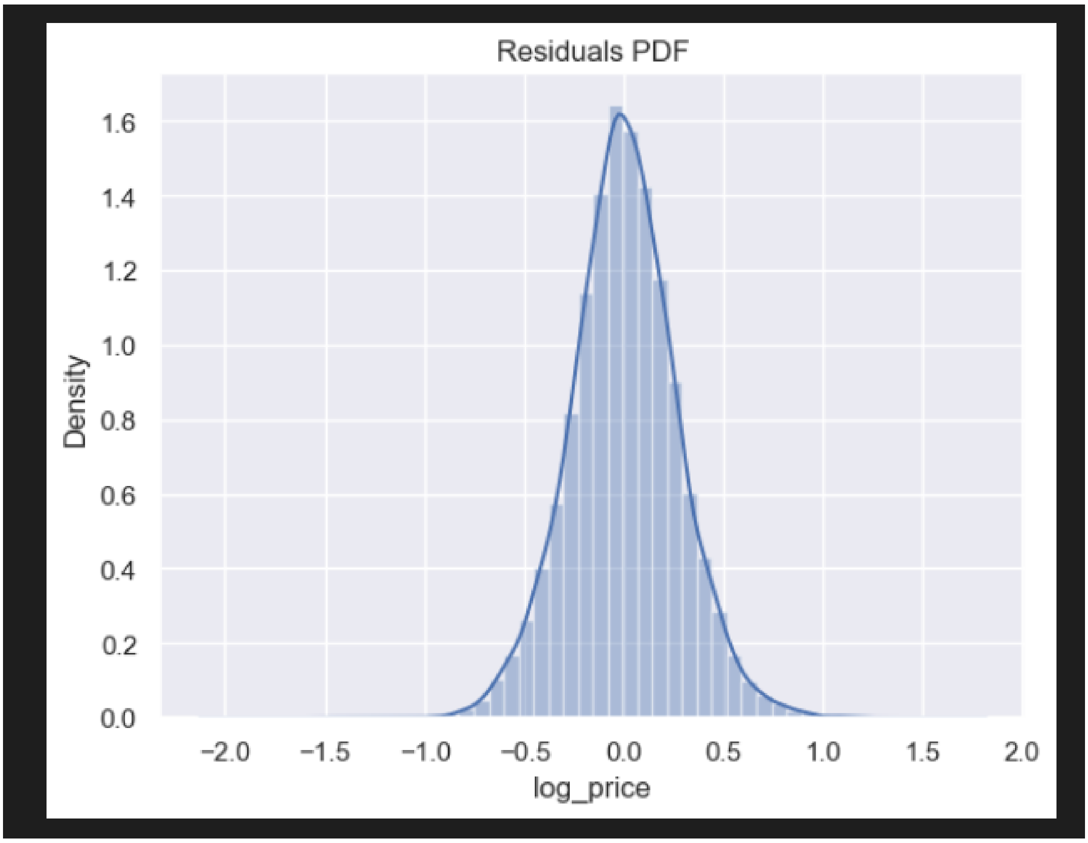
[Full Documentation of the Melbourne Housing Market Regression Analysis]

Multiple Linear Regression

1. Train Test Split
   We will split the data into 80:20, 80% training data and 20% testing data.
2. Regression
   After training the data, we could fit the data into the LinearRegression model and plot the training data as follows:



3. Evaluate
   After that, we need to plot the Residual Plot to validate our OLS, it helps us to check whether the residuals in our analysis are normally distributed and whether or not they exhibit heteroscedasticity.



After plotting the residual, we could evaluate whether the data fits the line or not using R-squared and Adjusted R-squared, thus:

Evaluating the Regression Mode

```
model.score(x_train, y_train)
```

```
0.5926448818238725
```

Calculate the Adjusted R-squared

Adjusted R-Squared Formula

$R^2_{adj.} = 1 - (1 - R^2) * \frac{n-1}{n-p-1}$

Where,

- n = number of observation
- p = number of predictors

```
n = x_train.shape[0]
p = x_train.shape[1]

adjusted_r2 = 1 - (1 - model.score(x_train
adjusted_r2
```

```
0.5746214457084808
```

Now, we want to find the weight and the coefficient of this analysis, and we have obtained as follows:

	Features	Weights
0	Unnamed: 0	4.899092e-03
1	Rooms	1.249806e-01
2	Distance	-4.676076e-02
3	Bathroom	9.818166e-03
4	Car	2.141773e-02
...	...	...
986	Regionname_Northern Victoria	-1.812458e+11
987	Regionname_South-Eastern Metropolitan	7.108649e+10
988	Regionname_Southern Metropolitan	1.882123e+11
989	Regionname_Western Metropolitan	-2.052852e+11
990	Regionname_Western Victoria	4.609471e+11

991 rows × 2 columns

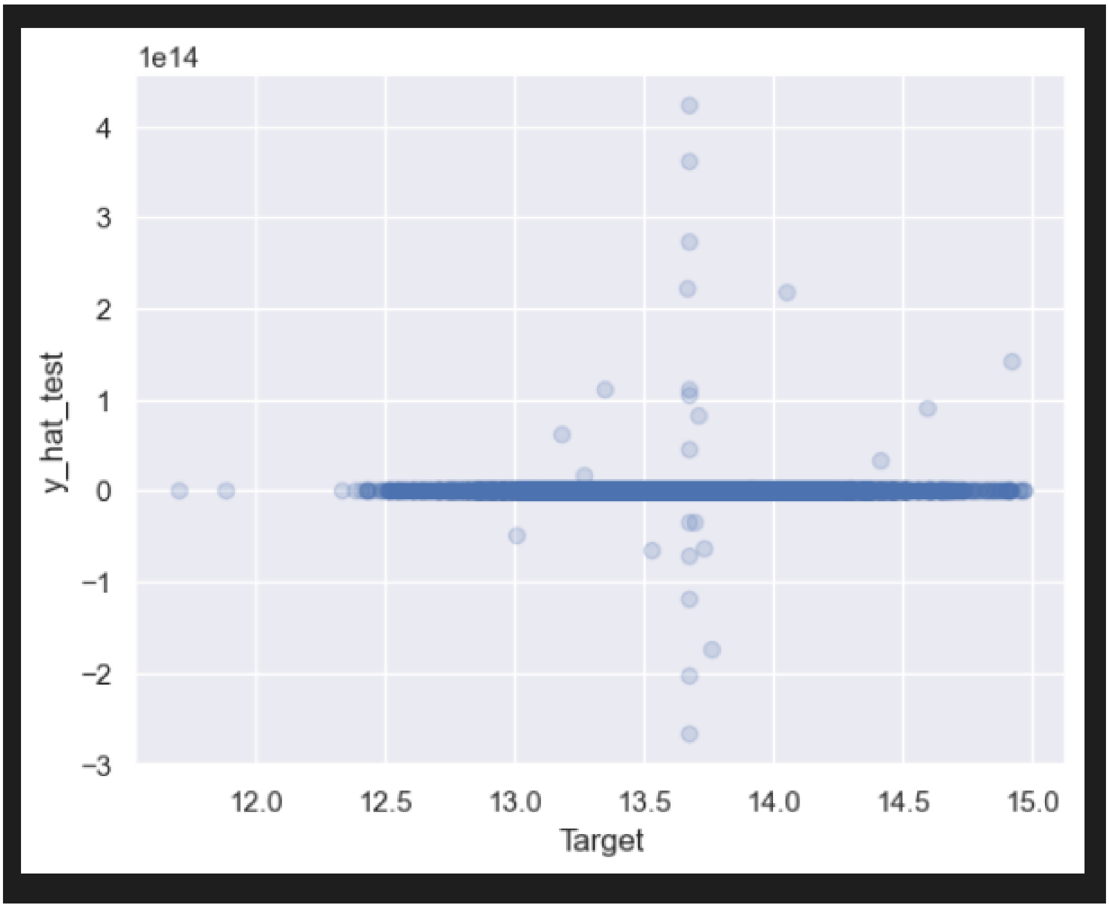
As you can see, those weights are some positives and some negatives. What is the meaning of weights?

- **POSITIVE WEIGHTS** = As a feature increases in value, so does the `log_price` or the price of the house, respectively
- **NEGATIVE WEIGHTS** = As a feature increases in value, the `log_price` or the price of the house will also **decreases**

While for the dummies, it means that,

- **POSITIVE WEIGHTS** shows that the respective category is more expensive rather than the benchmark (the first column of the category)
- **NEGATIVE WEIGHTS** the respective category is cheaper rather than the benchmark

4. Testing the data

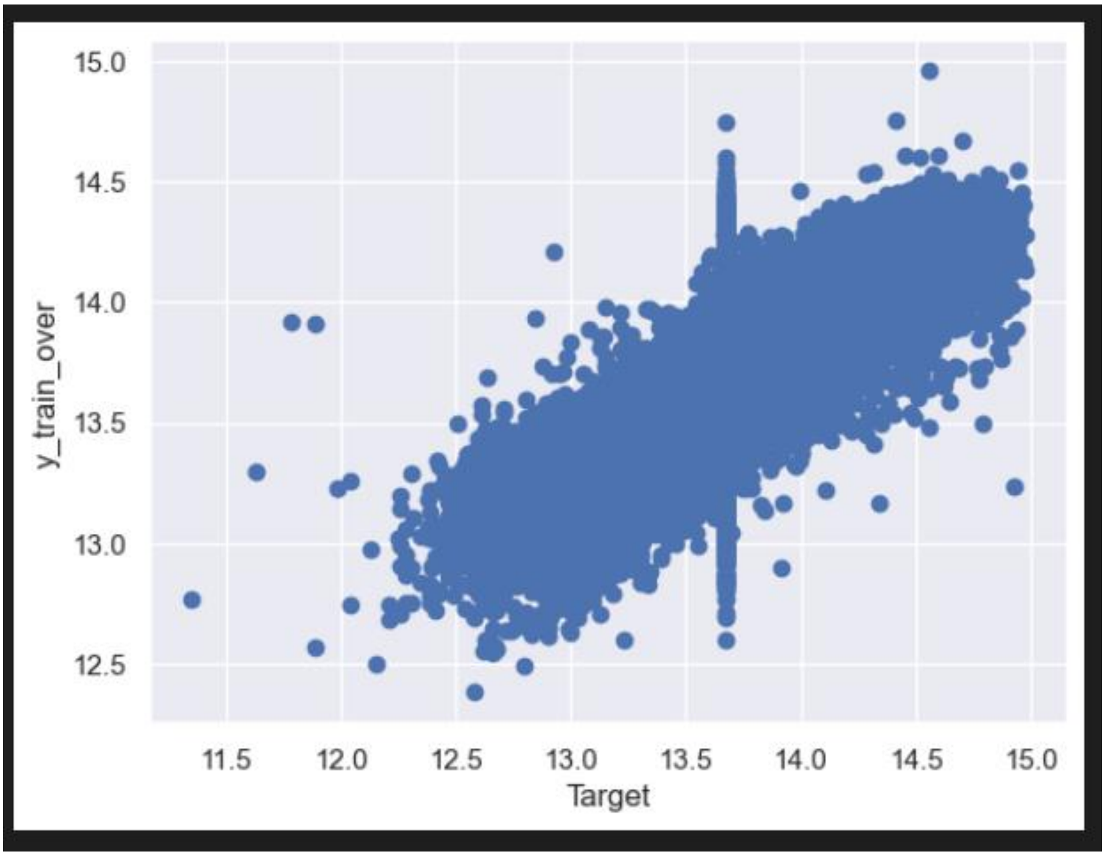


As shown in the figure above, we could say that this data is not linear, and the model is overfitting the data. However, let's see the prediction result:

```
C:\Users\LENOVO\AppData
  performance = pd.Data
```

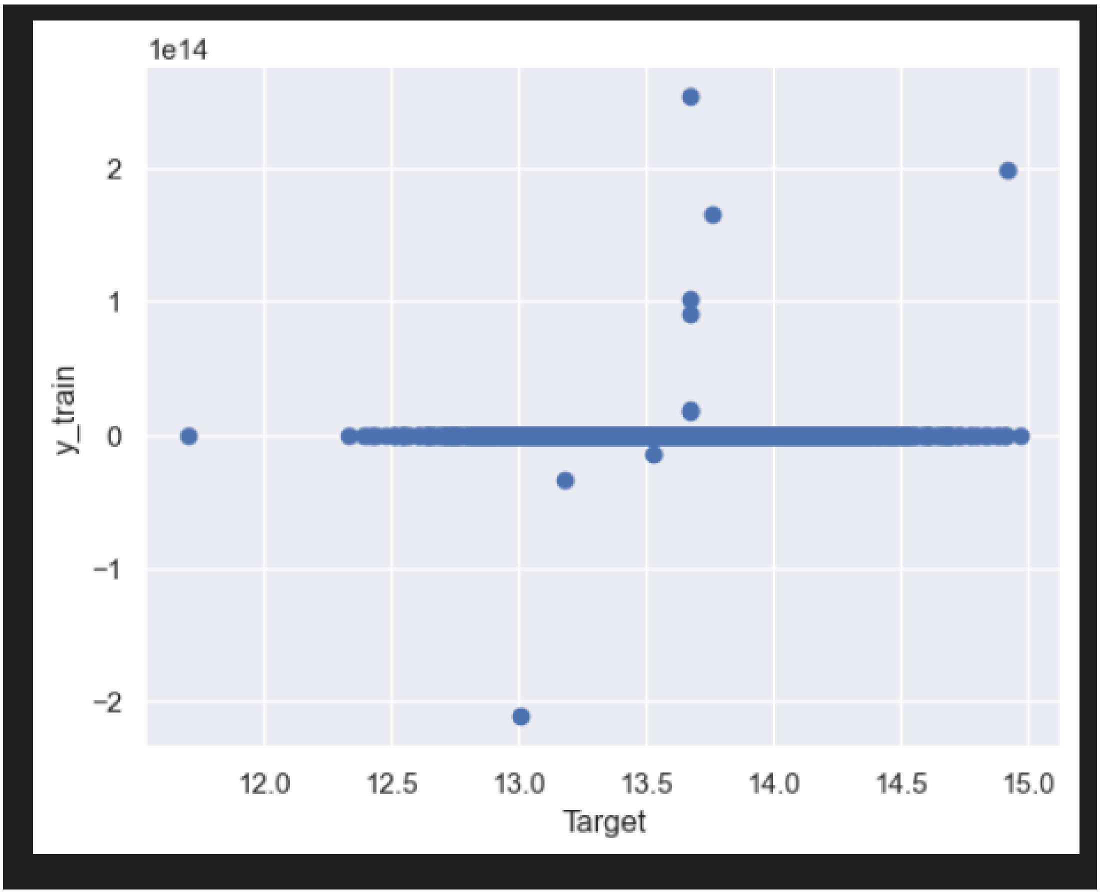
	Prediction
0	7.295277e+05
1	1.196317e+06
2	7.634019e+05
3	5.679010e+05
4	inf

5. Re-train Test Split
   Let's increase the data splitting into 90:10, 90% training data and 10% testing data.
6. Regression



As you can see, there are no significant changes in this model.

7. Re-Testing



However, we must acknowledge that this data is not linear and has been proven in this plot.

8. Evaluate

The performance of the model:

	Prediction	Target	Residuals	Difference%
count	5.848000e+03	5.848000e+03	5.848000e+03	5848.000000
mean	inf	9.384155e+05	-inf	inf
std	NaN	4.200663e+05	NaN	NaN
min	0.000000e+00	1.210000e+05	-inf	0.002362
25%	6.668717e+05	6.655000e+05	-1.183172e+05	7.905874
50%	8.554393e+05	8.700000e+05	6.590857e+03	16.996526
75%	1.099974e+06	1.100000e+06	1.671605e+05	28.778502
max	inf	3.175000e+06	1.937952e+06	inf

Highest coefficient (Most influential feature)

	Features	Weights
1	Rooms	1.249806e-01
6	BuildingArea	2.763003e-02
4	Car	2.141773e-02
5	Landsize	1.716589e-02
3	Bathroom	9.818166e-03
0	Unnamed: 0	4.899092e-03
7	YearBuilt	-1.907640e-02
2	Distance	-4.676076e-02
9	Longtitude	-1.781874e+00
8	Lattitude	-1.782309e+00
10	Propertycount	-4.684005e+11

9. Summary

Accuracy

```
# 80% Training Set
print('MAE  :',metrics.mean_abso
print('MAPE :',metrics.mean_abso
print('MSE  :',metrics.mean_squa
print('RMSE :',np.sqrt(metrics.m
```

```
MAE  : 579344653273.0378
MAPE : 42135270044.73972
MSE  : 1.2604419499246601e+26
RMSE : 11226940589157.227
```

```
# 90% Training Set
print('MAE  :',metrics.mean_abso
print('MAPE :',metrics.mean_abso
print('MSE  :',metrics.mean_squa
print('RMSE :',np.sqrt(metrics.m
```

```
MAE  : 378011750120.4941
MAPE : 27510340292.198647
MSE  : 6.701548838883953e+25
RMSE : 8186298821130.312
```

As you can see on the aside, the RMSE of this regression model is too high and indicates this model is not suitable for this data.

Neural Network

1. Train Test Split
   We will split the data into an 80:20 portion, 80% training dataset and 20% training dataset.

2. Neural Network Architecture

```
model = Sequential()
```
[15]

```
# Rectified Linear Unit (RELU)
model.add(Dense(128, input_dim = 998, activation='relu'))
model.add(Dense(64, activation='relu'))

# Output Layer
model.add(Dense(1, activation='linear'))
```
[16]

```
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mae'])
model.summary()
```
[17]

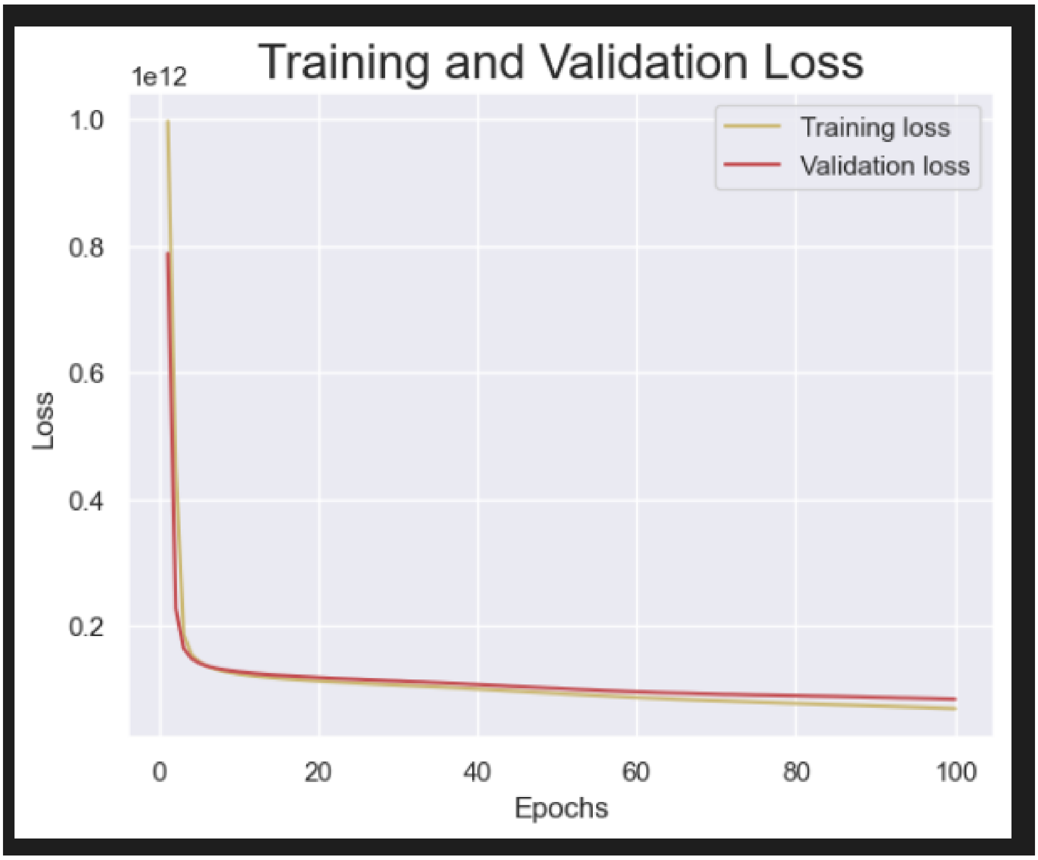
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               127872

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 136,193
Trainable params: 136,193
Non-trainable params: 0
_________________________________________________________________
```

In Neural Network architecture, we use 128 and 64 neurons as our hidden layer to train this data and use Adam optimizer to compile the dataset.

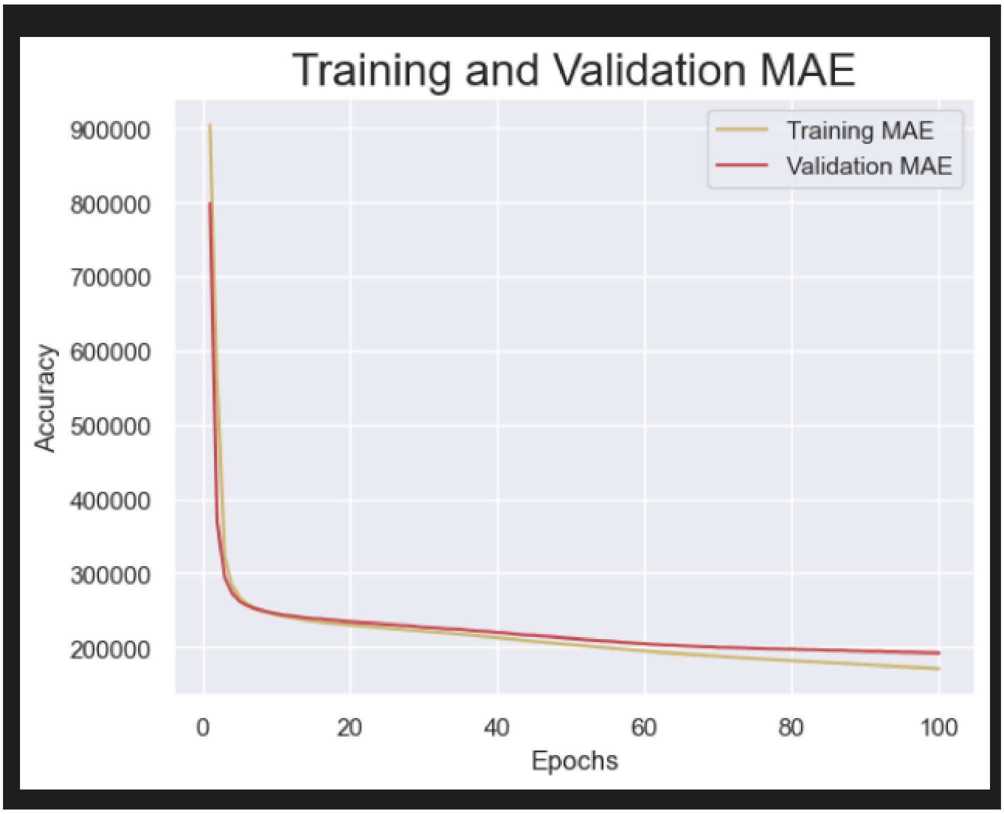
To fit the dataset into the model, we use 100 epochs to train the data resulting as follows:

```
Epoch 99/100
731/731 [==============================] - 1s 2ms/step - loss: 69772435456.0000 - mae: 171761.0312 - val_loss: 84391182336.0000 - val_mae: 192662.1719
Epoch 100/100
731/731 [==============================] - 1s 2ms/step - loss: 69345558528.0000 - mae: 171229.7969 - val_loss: 84018888704.0000 - val_mae: 192391.4219
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

3. Training Loss, Validation Loss, and Accuracy Rate
   The loss of the data is decreased significantly so fast, which means that the model fits the data very well.



While the accuracy of the data is:



As it is shown in the plot above, we could say that the model is quite overfit on 80 to 100 epochs. However, we know that the model accuracy is increased, which is shown by the decreasing MAE score.

4. Prediction

```
1/1 [==============================] - 0s 30ms/step
Predicted Values:  [[ 330009.78 ]
 [1047199.3  ]
 [ 860664.56 ]
 [ 489561.97 ]
 [   7318.636]]


Actual Values:  12547     985000.0
963      1425000.0
28609     920000.0
1412      502000.0
15961     870000.0
Name: Price, dtype: float64
```

As it's shown above, although the prediction is not accurately 100%, it's better than MLR.

5. Evaluation

```
183/183 [==============================] - 0s 1
MSE  :  84018888704.0
MAE  :  192391.421875
RMSE :  289860.11920234904
```

R-Squared

```
from sklearn.metrics import r2_score
```

```
y_true = np.array(y_test)
y_pred = np.array(model.predict(x_test))
```

```
183/183 [==============================] - 0s 8
```

```
r2_score(y_true, y_pred)
```

```
0.5237713739630026
```

The RMSE score is decreased significantly from the MLR, this means that the model fits the data very well. This result can be improved by tuning the hyperparameters.

Other than that, the R-squared score seems lower than the MLR, but it's not a big problem, since we want to find the performance of the model, not how well the data fits the line of regression.

V. Business Implementation

Based on our analysis, we must acknowledge that in real life no data is linear each other. Thus, we need a non-linear approach to leverage the machine learning model to its best performance.

In this analysis, we could help real-estate companies, banks, and other financial institutions to predict the house price accurately. This means that whenever a company want to open a new home cluster, they can predict what kind of house has the biggest Return on Investment (ROI). This means that they could determine what kind of feature or aspects in their home cluster to improve its price over time. For example, it's shown that the highest coefficients in this analysis are the rooms, cars, land size, etc. By leveraging these coefficients, the company could maximize its profitability and lower its production cost.