Temasek Polytechnic School of Informatics & IT

Diploma in Applied Artificial Intelligence

Machine Learning for Developers (CAI2C08)

AY2024/2025 April Semester

Academic Declaration

Project

Practical Class:	P02
Submitted by: (list all students)	Wu Guan Yu / 2302258E
Date:	26/7/2024

"By submitting this work, I am declaring that I am the originator(s) of this work and that all other original sources used in this work have been appropriately acknowledged.

I understand that plagiarism is the act of taking and using the whole or any part of another person's work and presenting it as mine without proper acknowledgement.

I also understand that plagiarism is an academic offence, and that disciplinary action will be taken for plagiarism."

Name and Signature of Student: Wu Guan Yu, Wu Guan Yu

Introduction

My topic is about predicting house prices for houses located in King County, Washington. This prediction model would help realtors, home owners, and buyers. Helping them to get a gauge price of the house's potential prices, enabling people to have a sense of the price market and use it to their own advantage.

The title of my chosen dataset is "House Sales in King County, USA". The dataset is created to show homes and their prices that is sold between May 2014 and May 2015 in King County, Washington. This dataset is obtained from Kaggle and this it the link to it: https://www.kaggle.com/datasets/harlfoxem/housesalesprediction/data. The data inside is provided by King County, Washington.

Original Dataset before cleaning(First five rows)

id	date	price	bedrooms	bathroom:	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_baser	yr_built	yr_renovat	zipcode	lat	long	sqft_living	sqft_lot15
7.1E+09	20141013	221900	3	1	1180	5650	1	0	0	3	7	1180	0	1955	0	98178	47.5112	-122.257	1340	5650
6.4E+09	20141209	538000	3	2.25	2570	7242	2	0	0	3	7	2170	400	1951	1991	98125	47.721	-122.319	1690	7639
5.6E+09	20150225	180000	2	1	770	10000	1	0	0	3	6	770	0	1933	0	98028	47.7379	-122.233	2720	8062
2.5E+09	20141209	604000	4	3	1960	5000	1	0	0	5	7	1050	910	1965	0	98136	47.5208	-122.393	1360	5000
2E+09	20150218	510000	3	2	1680	8080	1	0	0	3	8	1680	0	1987	0	98074	47.6168	-122.045	1800	7503

Dataset's shape before cleaning

Rows: 21613   Columns: 21

Dataset's datatypes

Command Used: house_sales_df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21613 non-null  int64  
 1   date           21613 non-null  object 
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64  
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64  
 6   sqft_lot       21613 non-null  int64  
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64  
 9   view           21613 non-null  int64  
 10  condition      21613 non-null  int64  
 11  grade          21613 non-null  int64  
 12  sqft_above     21613 non-null  int64  
 13  sqft_basement  21613 non-null  int64  
 14  yr_built       21613 non-null  int64  
 15  yr_renovated   21613 non-null  int64  
 16  zipcode        21613 non-null  int64  
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64  
 20  sqft_lot15     21613 non-null  int64  
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

Features

Target/Label: Price

Id	- Unique ID for each home sold
date	- Date of the home sale
price	- Price of each home sold
bedrooms	- Number of bedrooms
bathrooms	- Number of bathrooms, where .5 accounts for a room with a toilet but no shower
sqft_living	- Square footage of the apartments interior living space
sqft_lot	- Square footage of the land space
floors	- Number of floors
waterfront	- A dummy variable for whether the apartment was overlooking the waterfront or not
view	- An index from 0 to 4 of how good the view of the property was
condition	- An index from 1 to 5 on the condition of the apartment,
grade	- An index from 1 to 13, where 1-3 falls short of building construction and design, 7 has an average level of construction and design, and 11-13 have a high quality level of construction and design.
sqft_above	- The square footage of the interior housing space that is above ground level
sqft_basement	- The square footage of the interior housing space that is below ground level
yr_built	- The year the house was initially built
yr_renovated	- The year of the house’s last renovation
zipcode	- What zipcode area the house is in
lat	- Lattitude
long	- Longitude

sqft_living15 - The square footage of interior housing living space for the nearest 15 neighbors
sqft_lot15 - The square footage of the land lots of the nearest 15 neighbors

Data Exploration and Pre-processing of data

1. See the info of the dataset
    a. Understand the Columns
    There are 20 features. In the features, there are 20 continuous features and 1 discrete data
    b. Know the dataset's datatypes
    Datatypes composes of 5 float, 15 integers, and 1 object
    c. Figure out whether the problem is a continuous or discrete problem
    The target feature is price which is a continuous data, therefore making this a continuous problem.
2. Check for missing data
    a. I would need to decide what to do with the missing data
    Command used: house_sales_df.isnull().sum()

```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

    There is no missing data, therefore I do not need to go through the two steps below
        i. Check for possible reasons(e.g could be a correlation)
        ii. Generally fill the missing data with the feature's average value unless remove
3. Check for weird data
    a. Outliers/Extreme data
        i. Figure out whether the data is important and could possibly badly influence the model

        To easily find out the features with outliers, I used a function called .describe(). This function shows me all the aggregations and statistics of the features.

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000	21613.000000
mean	4.580302e+09	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+04	1.494309	0.007542	0.234303	3.409430
std	2.876566e+09	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+04	0.539989	0.086517	0.766318	0.650743
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	0.000000	0.000000	1.000000
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.000000	3.000000
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000	3.000000
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.000000	4.000000
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000	5.000000

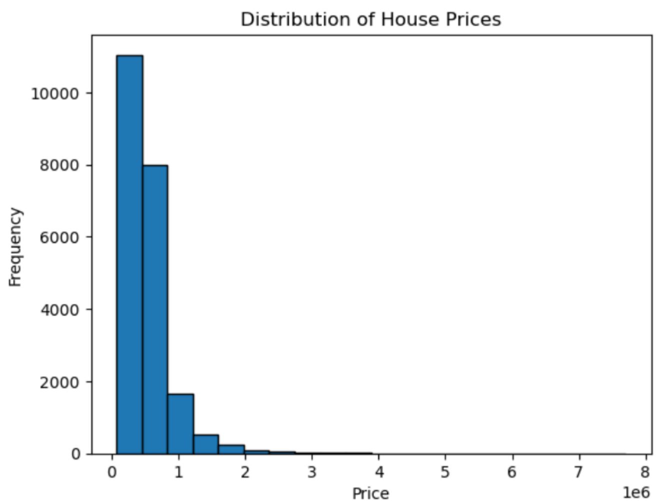
grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000
7.656873	1788.390691	291.509045	1971.005136	84.402258	98077.939805	47.560053	-122.213896	1986.552492	12768.455652
1.175459	828.090978	442.575043	29.373411	401.679240	53.505026	0.138564	0.140828	685.391304	27304.179631
1.000000	290.000000	0.000000	1900.000000	0.000000	98001.000000	47.155900	-122.519000	399.000000	651.000000
7.000000	1190.000000	0.000000	1951.000000	0.000000	98033.000000	47.471000	-122.328000	1490.000000	5100.000000
7.000000	1560.000000	0.000000	1975.000000	0.000000	98065.000000	47.571800	-122.230000	1840.000000	7620.000000
8.000000	2210.000000	560.000000	1997.000000	0.000000	98118.000000	47.678000	-122.125000	2360.000000	10083.000000
13.000000	9410.000000	4820.000000	2015.000000	2015.000000	98199.000000	47.777600	-121.315000	6210.000000	871200.000000

In here, one of the features with extreme data is bedrooms. The max value is 33, meaning a house that was sold had 33 bedrooms which seem implausible.

15872	2.4E+09	20140625T	640000	33	1.75	1620	6000	1	0	0	5	7	1040	580	1947	0	98103	47.6878	-122.331	1330	4700

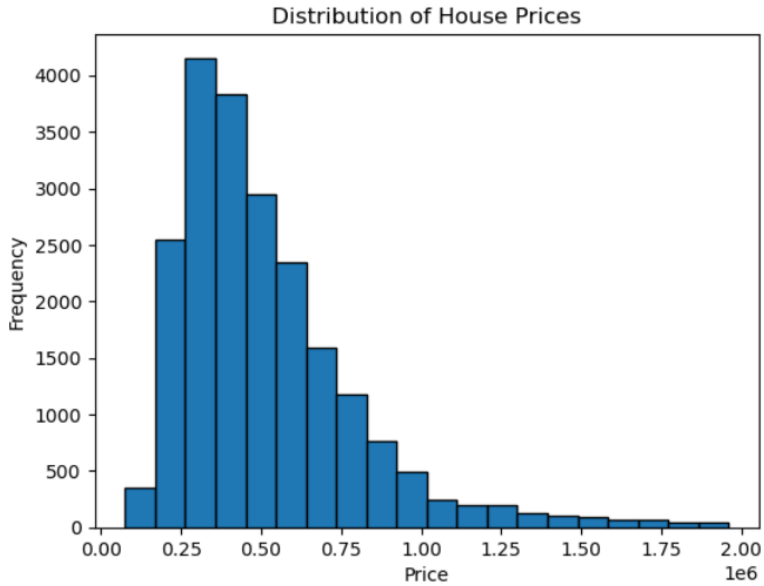
When you find the row, even the values in other features such as the sqft_living does not make sense with the number of bedrooms. It is not plausible to fit 33 bedrooms inside 1620sqft of space. The bedroom value seems to be a typo because when you find the actual house with the zip code it does not show a house with 33 bedrooms. Therefore, would remove this row from the dataset as an outlier.

Prices



The distribution of the house prices is right skewed. The median price value is 450000, whereas the highest price value is $7700000.

If I were to just remove the houses with prices in the 99th percentile, the distribution graph would improve greatly and be less right skewed.



However if I were to remove the outliers, the model's R^2 training and testing decreases, making it less accurate.

```
Training Set Mean Absolute Error: 43044.5190
Test Set Mean Absolute Error: 57791.3362
Test-Training Mean Absolute Error: 14746.8172
Training R2 Error: 0.9045
Test R^2 Error: 0.8992
```

Whereas if I were to not remove the outliers, the model's R^2 training and testing improves, making it more accurate and less overfitting.

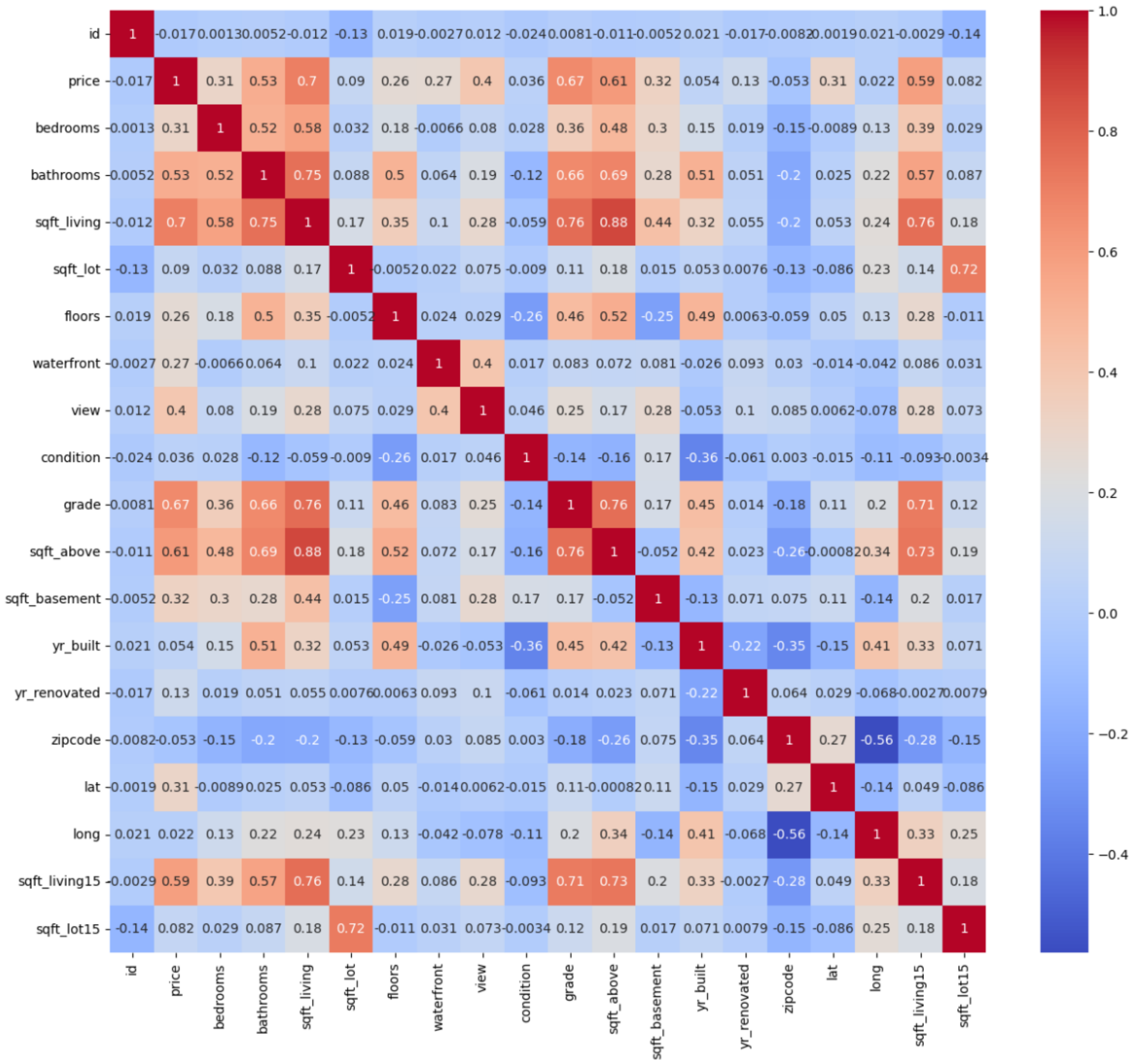
```
Training Set Mean Absolute Error: 46917.7556
Test Set Mean Absolute Error: 63705.9244
Test-Training Mean Absolute Error: 16788.1688
Training R2 Error: 0.9037
Test R^2 Error: 0.9035
```

The reason for this is because the houses with prices that seem to be outliers have other features that justifies the house's selling price.
For example(the highest selling price):

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
7252	6762700020	20141013T000000	7700000.0	6	8.0	12050	27600	2.5	0	3	4	13	8570	3480	1910	1987	98102	47.6298	-122.323	3940	8800

This 7.7million USD house has 6 bedrooms, 8bathrooms, 12050sqft of interior living space, and 27600sqft of outdoor space. Therefore, justifying the house's selling price and I would keep the outliers in the dataset

4. Plot out graphs and find the correlations between the features and the target



The features with the highest correlation to the prices are:

Bathrooms – 0.53

Sqft_living – 0.70

grade – 0.67

sqft_above – 0.61

sqft_living15 – 0.59

Data Cleaning and Pre-processing

Weird Data

I removed the data with 33 bedrooms because it is not possible as mentioned above.

Feature transformation

Combined yr_built and yr_renovated and date to create no_yrs_built and no_yrs_renovated. The no_yrs_built is how old the building is and the no_yrs_renovated is how old the renovation is.

The reason is because when gauging house prices, people do not look at the year it was built or the year it was renovated. They look focus on how old the building is base on the year the building was built and how old the renovation is since its last renovation. The machine would also be able to better pick out the underlying relationship between how old the building is and how old the renovation is to the price. This is better compared to making the machine find the underlying relationship between the year the building is built and the year the building is renovated.

no_yrs_built

I got no_yrs_built by finding the difference between the year it was built and the year it was sold. For no_yrs_built there are some values with -1, the buyers could have bought the house before the completion date and only -1 exists. Often there are price differences between buying a house before its completion date and buying after it is done. Therfore, I should leave all houses that were bought before completion date to -1.

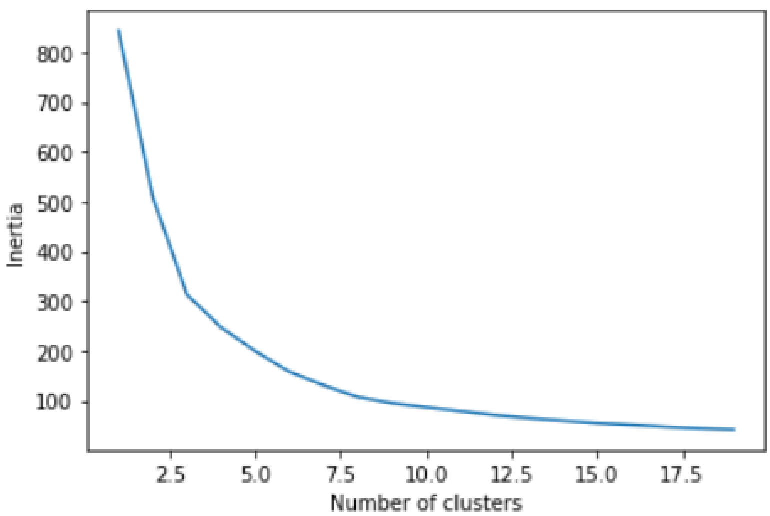
no_yrs_renovated

I got no_yrs_renovated by finding the difference between the year it was renovated and the year it was sold. For no_yrs_renovated, there were also some houses with no renovation making it possible for no_yrs_renovated a negative value, so I mad sure to convert all negative values to 0.

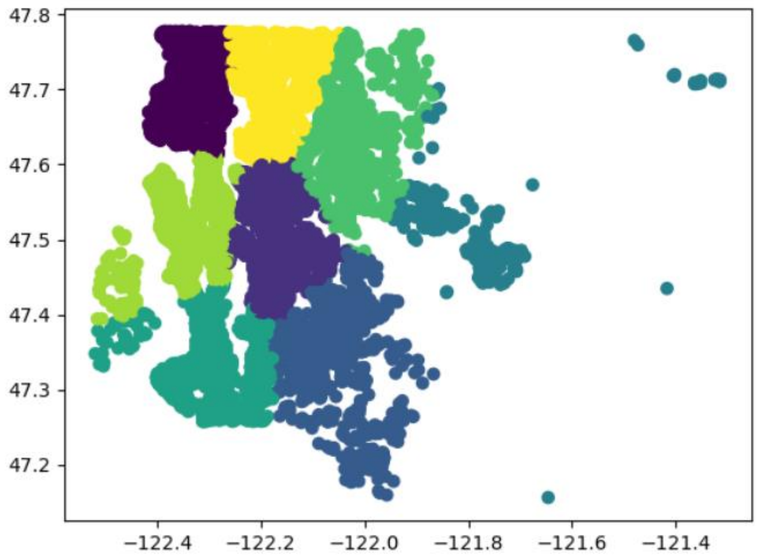
Grouped the houses based on their latitudes and longitudes with k-means clustering

The reason is because the house's price can be affected by the neighbourhood it is in. By using k-means to cluster the houses to their respective neighbourhoods, the houses would be properly grouped and the machine learning model would now be able to take the neighbourhood the house is in into account and accurately predict the house price.

To find the binning value I used the Elbow Method find the bin value before the rate of inertia changing decreases.



In the graph above it showed that rate of inertia changing decreases at 8 bins. Increasing the value pass 8 bins would be futile or may even decrease the model's performances



This is a graph of all the houses grouped in to their respective bins. The bins could also be represented as neighbourhoods. The shape plotted is also similar to the geographic shape of King County, Washington.

Remove zipcode

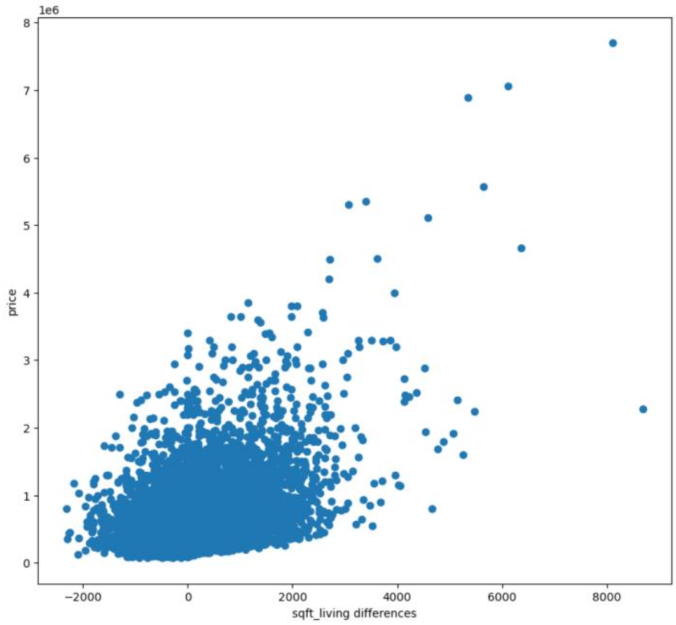
I removed zipcode because of two reasons. The zipcode is a categorical data and if I were to one-hot encode it, I would have 70 extra features. This would affect the curse of dimensionality and I would need way more rows to have a more accurate model prediction. The other reason is because I already clustered the houses into their respective neighbourhoods based on their longitude and latitude. This longitude and latitude is also more accurate in identifying the location of the house than the zipcode.

Remove yr_built and yr_renovated

I remove yr_built and yr_renovated because I have created no_yr_built and no_yr_renovated to replace this two features. Therefore, this two features are not needed anymore.

Combining the sqft_living and sqft_living15

I tried Combining the sqft_living, sqft_lot and sqft_living15, sqft_lot15 to find the differences. This is because the prices of 15 houses next to each other is generally the same and if the sqft_living of the house is higher than the average 15 neighbourhood houses, the price of the house is higher and vice versa. This can be seen in the graph below



Failed attempts

Binning grade

I tried binning grade into four bin where, low = 1-3, low-mid = 3-7, high-mid=7-11, high=11-13. I did this is because it helps to improve the accuracy of the predictive models by reducing the noises in the dataset and generalising the grade into four different categories instead of 1-13. However, it reduced my R^2 value by roughly 0.04.

```
Training Set Mean Absolute Error: 48762.4709
Test Set Mean Absolute Error: 65323.7964
Test-Training Mean Absolute Error: 16561.3255
Training R2 Error: 0.9048
Test R^2 Error: 0.9004
```

Best result(After cleaning):

```
Training Set Mean Absolute Error: 47063.1883
Test Set Mean Absolute Error: 63495.3186
Test-Training Mean Absolute Error: 16432.1302
Training R2 Error: 0.9058
Test R^2 Error: 0.9051
```

Methods and Improvements

Deciding which model to use

I have experimented using GradientBoostingRegressor(), BaggingRegressor(), RandomForestRegressor(), and LinearRegression().

GradientBoostingRegressor

```
GradientBoostingRegressor
Train Mse: 73000.906
Test Mse: 81577.557
Diff Mse: 8576.651
Train R^2: 0.902
Test R^2: 0.855
Diff R^2 0.047
```

BaggingRegressor

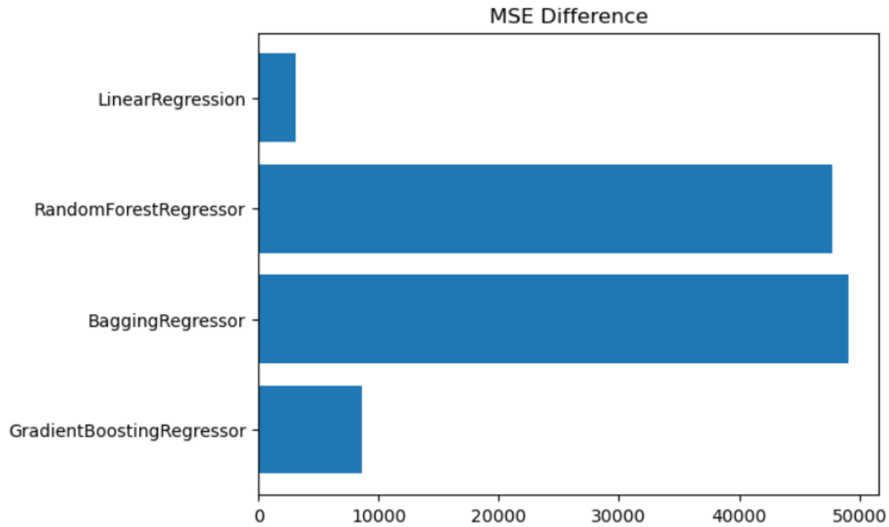
```
GradientBoostingRegressor
Train Mse: 73000.906
Test Mse: 81577.557
Diff Mse: 8576.651
Train R^2: 0.902
Test R^2: 0.855
Diff R^2 0.047
```

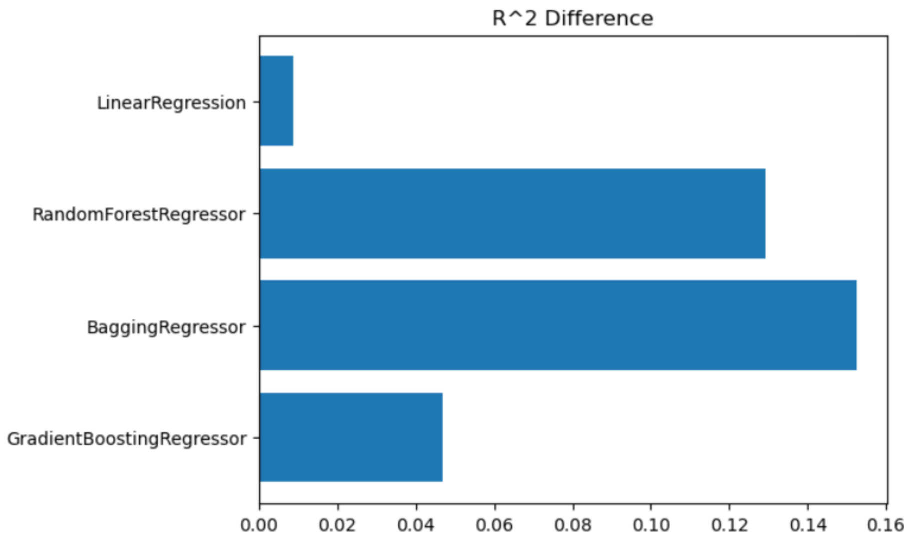
RandomForestRegressor

```
RandomForestRegressor
Train Mse: 26547.601
Test Mse: 74280.042
Diff Mse: 47732.441
Train R^2: 0.981
Test R^2: 0.851
Diff R^2 0.129
```

LinearRegression

```
LinearRegression
Train Mse: 118059.395
Test Mse: 121149.015
Diff Mse: 3089.619
Train R^2: 0.731
Test R^2: 0.722
Diff R^2 0.009
```





From the graphs above It looks like LinearRegression is the best choice for my dataset. However, the values of its mean absolute error are very high and its r^2 values are low. If you look at the second best, which is GradientBoostingRegressor, its mean absolute error values are very low compared to linear regression and its r^2 values are way higher. Therefore, I chose GradientBoostingRegressor for my model.

Finding the best hyper-parameters

I used GridSearchCV and used this parameters:

```
param_grid = {
    'n_estimators': [500,800,1000,2000,3000],
    'max_depth': [4,5,6,7,8,9],
    'min_samples_leaf': [3, 5, 9, 30, 40, 50],
    'max_features': [1.0, 0.3, 0.1],
    'loss': ['huber'],
    'learning_rate':[0.05, 0.1, 0.15, 0.2]
}
```

It took 36hours to run and this were my top three:

```
Top 3 best parameters:
1. {'learning_rate': 0.05, 'loss': 'huber', 'max_depth': 5, 'max_features': 0.3, 'min_samples_leaf': 3, 'n_estimators': 3000}
2. {'learning_rate': 0.05, 'loss': 'huber', 'max_depth': 6, 'max_features': 0.1, 'min_samples_leaf': 3, 'n_estimators': 3000}
3. {'learning_rate': 0.05, 'loss': 'huber', 'max_depth': 7, 'max_features': 0.3, 'min_samples_leaf': 3, 'n_estimators': 1000}
```

However when I ran them, I found that the models were very overfitted

1.
```
Training Set Mean Absolute Error: 30816.0255
Test Set Mean Absolute Error: 63594.9717
Test-Training Mean Absolute Error: 32778.9461
Training R2 Error: 0.9035
Test R^2 Error: 0.8963
```

2.
```
Training Set Mean Absolute Error: 30005.3554
Test Set Mean Absolute Error: 65225.3738
Test-Training Mean Absolute Error: 35220.0184
Training R2 Error: 0.8963
Test R^2 Error: 0.8877
```

3.
```
Training Set Mean Absolute Error: 29166.2857
Test Set Mean Absolute Error: 64314.6368
Test-Training Mean Absolute Error: 35148.3511
Training R2 Error: 0.8877
Test R^2 Error: 0.8890
```

The reason is because, GridSearchedCV was made to give me the best hyper-parameters for data it was given and I only gave it training data. Therefore, it gave me hyperparameters that are perfect for only the training data, leading to overfitting. To prevent overfitting, I started tweaking the hyper-parameters manually. I first decrease the learning_rate as it reduces overfitting and also decreased n_estimators as there would be lesser trees making the model less prone to overfitting. I also increased the min_samples_leaf and min_samples_split as it makes the decision trees in GradientBoostingRegressor less complexed and more generalised, reducing the risks of overfitting.

After tweaking I the hyperparameters with the least overfitting:

```
model = GradientBoostingRegressor(
    learning_rate = 0.01,
    loss = 'huber',
    max_depth = 6,
    max_features = 0.3,
    min_samples_leaf = 20,
    min_samples_split = 20,
    n_estimators = 2500,
)
```

Feature Selection

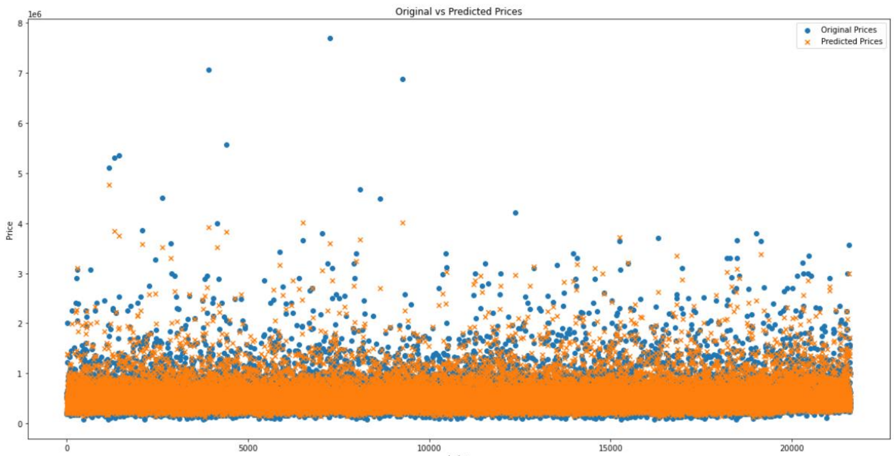
When I find the feature's importances none of them are 0.00%

```
neighbourhood_10 - 0.04%
neighbourhood_0 - 0.04%
neighbourhood_7 - 0.05%
neighbourhood_2 - 0.06%
neighbourhood_8 - 0.09%
neighbourhood_9 - 0.10%
neighbourhood_3 - 0.15%
no_yrs_renovated - 0.19%
neighbourhood_6 - 0.21%
yr_sold - 0.25%
waterfront - 0.40%
floors - 0.55%
neighbourhood_1 - 0.65%
sqft_basement - 0.69%
condition - 0.69%
neighbourhood_11 - 0.76%
bedrooms - 0.82%
neighbourhood_4 - 0.88%
sqft_lot15 - 1.28%
sqft_lot - 1.36%
view - 1.50%
bathrooms - 1.69%
neighbourhood_5 - 1.92%
no_yrs_built - 2.49%
long - 2.74%
sqft_above - 4.33%
sqft_living15 - 6.62%
sqft_living - 19.28%
grade - 22.09%
lat - 28.05%
```

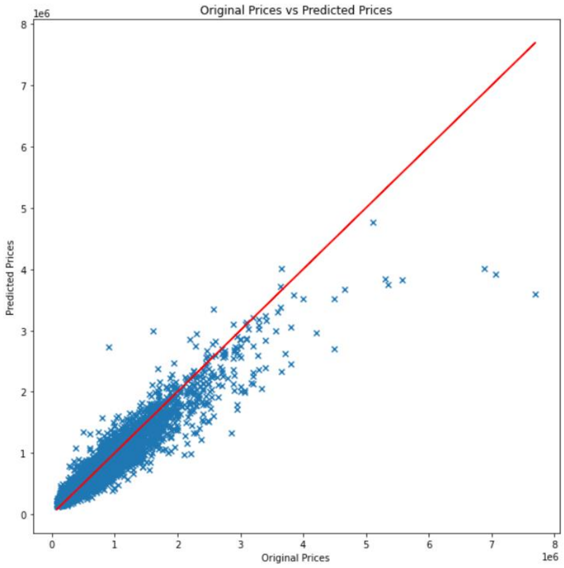
I would be using all the features in here as the features with lowest importance are all my neighbourhoods. They represent the neighbourhood the house is in categorically, which is very important. Therefore, I am keeping all the features.

Results and Analysis

The x-axis is row index and y-axis is price. The orange crosses represents the predicted prices and the blue circles represent the actual prices. This scatter plot shows that generally all the predicted prices are accurate as they are all roughly in the same area as the original prices.



The graph’s x-axis is Original Prices and the y-axis is Predicted. The red line represents what the original values are. The blue crosses represents the predicted prices. This graph shows that the model is quite accurate as the blue crosses are roughly following the red line. This means that most of the predicted prices are very close to the original prices.



Conclusion

In conclusion, my dataset has been through exploratory data and analysis(EDA) and it is pre-processed. The dataset is fitted into the models that best utilises the dataset which is GradientBoostingRegressor. I tuned the GradientBoostingRegressor’s hyper-parameters, to better fit the dataset. The model’s results

```
Training Set Mean Absolute Error: 47063.1883
Test Set Mean Absolute Error: 63495.3186
Test-Training Mean Absolute Error: 16432.1302
Training R2 Error: 0.9058
Test R^2 Error: 0.9051
```

References

Dataset features definitions:
https://www.kaggle.com/datasets/harlfoxem/housesalesprediction/discussion/207885

Advantages of binning: https://www.scaler.com/topics/machine-learning/binning-in-machine-learning/

Learning Rate: https://deepchecks.com/question/does-learning-rate-affect-overfitting/#:~:text=Reducing%20the%20learning%20rate%20improves,rate%2C%20but%20is%20not%20overfitting.

min_samples_leaf and min_samples_split: https://medium.com/@ompramod9921/decision-trees-8e2391f93fa7

K-means vs DBSCAN: https://geoffboeing.com/2014/08/clustering-to-reduce-spatial-data-set-size/