

ALY6140 Analytics System Technology

CAPSTONE PROJECT

FINAL REPORT

Prediction of House Pricing in Melbourne



Prepared By:

Name: Shreyansh Bhalodiya

NUID: 002664707

Name: Pui Man SIU

NUID: 002685051

Name: Megha Bhagat

NUID: 002673025

Presented to: Professor Dinesh

Date: April 01st, 2023

INTRODUCTION / OBJECTIVES

We have chosen a Melbourne House Pricing Data set for our group project and intend to perform pricing analytics using machine learning models. This dataset is a snapshot of the available results posted every week from Domain.com.au, which I found on Kaggle[1].

We would be implementing three different machine learning algorithms using the scikit learn Python library and then execute them to calculate the error between the actual and predicted house sale price using four different performance metrics.

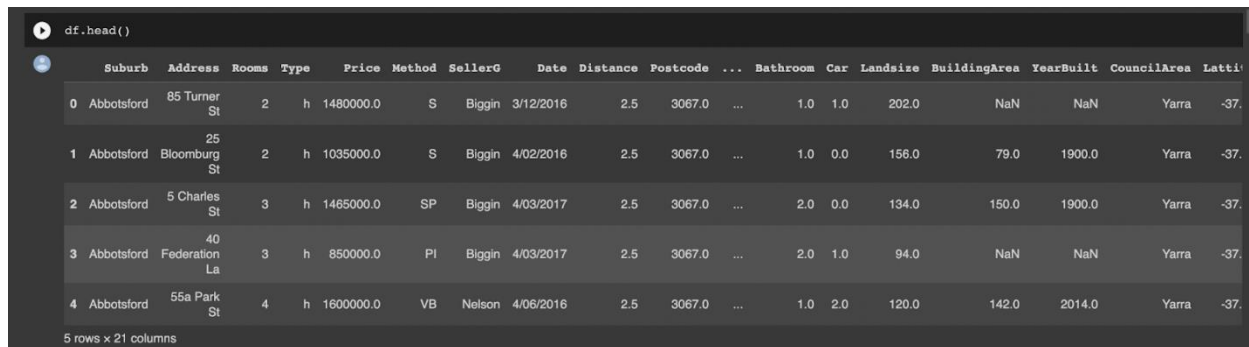
The key aim of our project would be to answer the following questions (2 to 6) through our models and Exploratory data analysis in an accurate manner.

1. What is the age distribution of properties in Melbourne?
2. Which area has the highest growth in property price?
3. What are the key drivers influencing the housing price in Melbourne?
4. What is the relationship between the above key drivers and housing price quantitatively?
5. Is there a significant difference in house price between different regions of Melbourne?
6. What is the predicted price of a house given the different features in the next few months?

EXPLORATORY DATA ANALYSIS

Overview of data structure

To begin with, we first have an overview of the data structure.



The screenshot shows a Jupyter Notebook interface with a code cell containing `df.head()`. Below the code, a table displays the first 5 rows of the dataset. The table has 21 columns: Suburb, Address, Rooms, Type, Price, Method, SellerG, Date, Distance, Postcode, ..., Bathroom, Car, Landsize, BuildingArea, YearBuilt, CouncilArea, and Lattitude. The first 5 rows are as follows:

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	Postcode	...	Bathroom	Car	Landsize	BuildingArea	YearBuilt	CouncilArea	Lattitude
0	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin	3/12/2016	2.5	3067.0	...	1.0	1.0	202.0	NaN	NaN	Yarra	-37.0
1	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin	4/02/2016	2.5	3067.0	...	1.0	0.0	156.0	79.0	1900.0	Yarra	-37.0
2	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin	4/03/2017	2.5	3067.0	...	2.0	0.0	134.0	150.0	1900.0	Yarra	-37.0
3	Abbotsford	40 Federation La	3	h	850000.0	PI	Biggin	4/03/2017	2.5	3067.0	...	2.0	1.0	94.0	NaN	NaN	Yarra	-37.0
4	Abbotsford	55a Park St	4	h	1600000.0	VB	Nelson	4/06/2016	2.5	3067.0	...	1.0	2.0	120.0	142.0	2014.0	Yarra	-37.0

5 rows x 21 columns

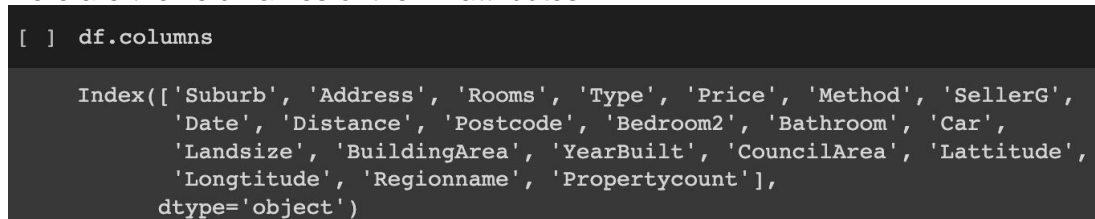
There are 13,580 observations and 21 attributes in the data set.



The screenshot shows a Jupyter Notebook interface with a code cell containing `df.shape`. Below the code, the output is displayed as `(13580, 21)`.

(13580, 21)	

Here are the field names of the 21 attributes.



The screenshot shows a Jupyter Notebook interface with a code cell containing `df.columns`. Below the code, the output is displayed as a list of 21 column names: Suburb, Address, Rooms, Type, Price, Method, SellerG, Date, Distance, Postcode, Bedroom2, Bathroom, Car, Landsize, BuildingArea, YearBuilt, CouncilArea, Lattitude, Longitude, Regionname, and Propertycount.

Index(['Suburb', 'Address', 'Rooms', 'Type', 'Price', 'Method', 'SellerG', 'Date', 'Distance', 'Postcode', 'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'YearBuilt', 'CouncilArea', 'Lattitude', 'Longitude', 'Regionname', 'Propertycount'], dtype='object')	

Handling missing values

Most machine learning algorithms work on complete observations of a data set, so the missing values must be handled before we can move on to the machine learning part. The number of missing values in each data field were counted. Missing values were mostly found in BuildingArea, YearBuilt and CouncilArea and a little in Car field. There are 47.5%, 39.6% and 10.1% of missing values out of total number of observations for Building Area, Year Built and Council Area respectively.

<code>df.isnull().sum()</code>	<code>1 [(df.isnull().sum()*100)/len(df)]</code>
Suburb 0	Suburb 0.000
Address 0	Address 0.000
Rooms 0	Rooms 0.000
Type 0	Type 0.000
Price 0	Price 0.000
Method 0	Method 0.000
SellerG 0	SellerG 0.000
Date 0	Date 0.000
Distance 0	Distance 0.000
Postcode 0	Postcode 0.000
Bedroom2 0	Bedroom2 0.000
Bathroom 0	Bathroom 0.000
Car 62	Car 0.457
Landsize 0	Landsize 0.000
BuildingArea 6450	BuildingArea 47.496
YearBuilt 5375	YearBuilt 39.580
CouncilArea 1369	CouncilArea 10.081
Lattitude 0	Lattitude 0.000
Longitude 0	Longitude 0.000
Regionname 0	Regionname 0.000
Propertycount 0	Propertycount 0.000
dtype: int64	dtype: float64

Since BuildingArea, YearBuilt and CouncilArea fields are critical attributes that we believe to influence property price for our model development, we can hardly drop all these fields. Therefore, we are going to cleanse and impute values one by one.

First we are going to cleanse the data field of "BuildingArea". It was found that 1,323 observations of BuildingArea is greater than the value of Landsize which is not possible, it's data error that needed to be cleansed.

```
nf = df[df["BuildingArea"] > df["Landsize"]]
nf.shape

## Here we have few columns where Building Area is greater then land size which is not possible so we will replace
## those numbers with null first becasue those values are not correct.

(1323, 21)
```

We first calculate the building area percentage by dividing BuildingArea with Landsize, the value is stored in 'Built area percent'.

```
df["Built_area_percent"] = df["BuildingArea"]/df["Landsize"]
```

Some records are having Built_area_percent > 1, we update the value and limit the maximum value to be 1.

Before cleansing			After cleansing		
Regionname	Propertycount	Built_area_percent	Regionname	Propertycount	Built_area_percent
Northern Metropolitan	4,019.000	NaN	Northern Metropolitan	4,019.000	NaN
Northern Metropolitan	4,019.000	0.506	Northern Metropolitan	4,019.000	0.506
Northern Metropolitan	4,019.000	1.119	Northern Metropolitan	4,019.000	1.000
Northern Metropolitan	4,019.000	NaN	Northern Metropolitan	4,019.000	NaN
Northern Metropolitan	4,019.000	1.183	Northern Metropolitan	4,019.000	1.000

After updating the maximum value to 1, we calculate the average Built_area_percent – 0.4926 and then impute the value to records with null value in Built_area_percent.

```
df["Built_area_percent"].mean()
0.49261336938459177

[ ] # replacing na values in college with No college
df["Built_area_percent"].fillna(df["Built_area_percent"].mean(), inplace = True)
```

We then create a finalized built area field named "BuildingArea_final" by multiplying the Landsize with the cleansed Built_area_percent above.

```
df["BuildingArea_final"] = df["Landsize"]*df["Built_area_percent"]
```

Now the BuildingArea field was cleansed.

For CouncilArea and Car fields, there are 1,369 and 62 missing values respectively. We just replace those null value with zero by now since the records are minimal comparatively.

```
## For council area we have 1323 null values out of 1369
df["CouncilArea"].fillna('0', inplace = True)
df["Car"].fillna(0, inplace = True)
```

Next, let's move to "YearBuilt" field, we employed a KNN imputer to impute the missing values based on the nearest neighbor. The missing values are imputed with the mean value of YearBuilt of k nearest neighbors of similar latitude and longitude.

Missing values are checked again to make sure all of them are cleansed up.

```
df.isnull().sum()
Suburb      0
Address     0
Rooms       0
Type        0
Price       0
Method      0
SellerG     0
Date        0
Distance    0
Postcode    0
Bedroom2    0
Bathroom    0
Car         0
Landsize    0
YearBuilt   0
CouncilArea 0
Latitude    0
Longitude   0
Regionname  0
Propertycount 0
Built_area_percent 0
BuildingArea_final 0
dtype: int64
```

Outliers Analysis

Boxplots were produced for numerical variables that we are interested in. There were some outliers identified under Price, Landsize and YearBuilt variables.

Figure 1- Boxplot of Price

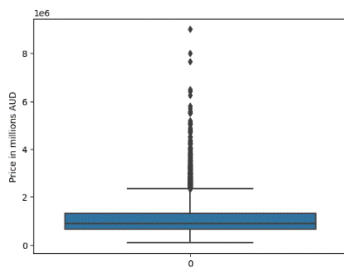


Figure 2 - Boxplot of Landsize

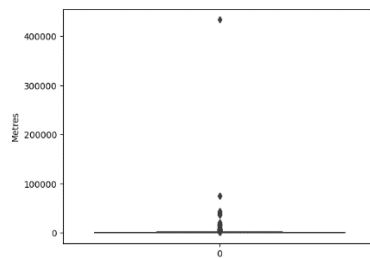
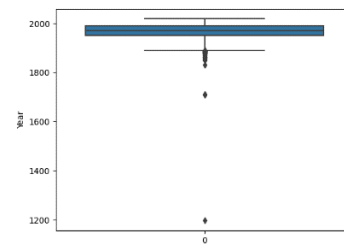


Figure 3 - Boxplot of YearBuilt



Our team has developed a self defined function `remove_outlier()` to remove outliers that are outside the range of $Q1 - 1.5 \text{ IQR}$ and $Q3 + 1.5 \text{ IQR}$.

```
### we saw that we have many variables with outlier let's drop outlier
def remove_outlier(df_in, col_name):
    q1 = df_in[col_name].quantile(0.25)
    q3 = df_in[col_name].quantile(0.75)
    iqr = q3-q1 #Interquartile range
    fence_low = q1-1.5*iqr
    fence_high = q3+1.5*iqr
    df_out = df_in.loc[(df_in[col_name] > fence_low) & (df_in[col_name] < fence_high)]
    return df_out
```

This function was applied to data fields to remove outliers including Price, Landsize and YearBuilt, which are the numerical fields that may affect our model.

Below is the result after removing the outliers.

Figure 4 - Boxplot of Price

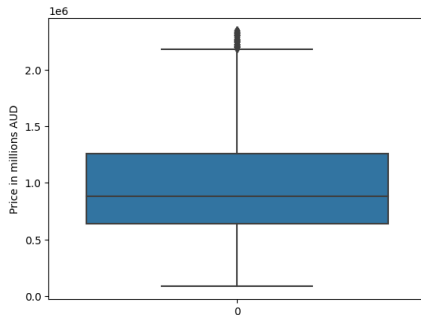


Figure 5 - Boxplot of Landsize

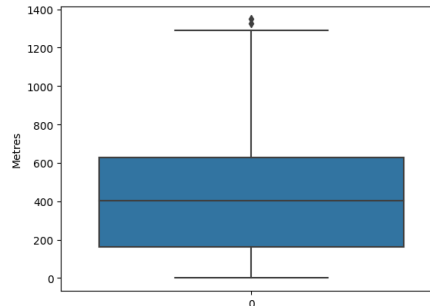
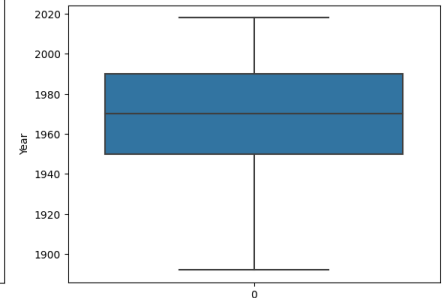


Figure 6 - Boxplot of YearBuilt



After cleansing the missing values and outliers, there are 12,022 rows and 22 variables remain for our upcoming model development.

```
### Hence we have
df.shape
### finally we have
(12022, 22)
```

Data preparation for machine learning models

Further data manipulation is needed so that it can be used for our model development.

First, we renamed the Price variable to 'TARGET'. This is our target variable for prediction. Furthermore, we have dropped variables including Date, Address, BuildingArea_final. Address is unique to each observation and has no use for machine learning, meanwhile we have transformed the variable Built_area_percent so we don't need the field BuildingArea_final.

```
df.rename(columns = {'Price':'TARGET'}, inplace = True)

[ ] df_model = df[['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
                  'Distance', 'Postcode', 'Bedroom2', 'Bathroom', 'Car',
                  'Landsize', 'YearBuilt', 'CouncilArea', 'Latitude', 'Longitude',
                  'Regionname', 'Propertycount', 'Built_area_percent', 'TARGET']]
```

Data conversion of YearBuilt field has been undertaken. This data field is converted from interger to string and later on will be converted into dummy variables as input variable for machine learning models.

```

1 convert_dict = {"YearBuilt":str
2                 }
3
4 df_model = df_model.astype(convert_dict)
5 print(df_model.dtypes)

```

```

Suburb      object
Rooms       int64
Type        object
Method      object
SellerG     object
Distance    float64
Postcode    float64
Bedroom2    float64
Bathroom    float64
Car         float64
Landsize    float64
YearBuilt   object
CouncilArea object
Latitude    float64
Longitude   float64
Regionname  object
Propertycount float64
Built_area_percent float64
TARGET      float64
dtype: object

```

Also, the dummy variables for categorical variables are created which can be fitted to the model later on. Originally, there are 7 categorical variables. After conversion, there are in total 741 dummy variables. The number of categories in each categorical variable is as below.

```

1 # No of levels in each categorical variable
2 df_model_cat = df_model.select_dtypes(exclude=['int', 'float'])
3 for col in df_model_cat.columns:
4     print(col,df_model_cat[col].nunique()) # to print categories name only
5

```

```

Suburb 307
Type 3
Method 5
SellerG 261
YearBuilt 123
CouncilArea 34
Regionname 8

```

Those 1 and 0 are values in dummy variables.

```

df_model_dummies.head()

```

	Longitude	Propertycount	...	CouncilArea_Yarra	CouncilArea_Yarra Ranges	Regionname_Eastern Metropolitan	Regionname_Eastern Victoria	Regionname_Northern Metropolitan	Regionname_Northern Victoria	Regionname Metropolitan
16	144.9984	4019.0	...	1	0	0	0	1	0	
19	144.9934	4019.0	...	1	0	0	0	1	0	
13	144.9944	4019.0	...	1	0	0	0	1	0	
19	144.9969	4019.0	...	1	0	0	0	1	0	
12	144.9941	4019.0	...	1	0	0	0	1	0	

Visualization and answering business questions

First, we are going to explore our target variable – Housing Price with a boxplot diagram. Housing Price in Melbourne ranges from \$0.09m to \$2.3m in 2016 to 2017, while mean price is at \$0.98m and median price is at \$0.88m.

Figure 7 – Price Distribution of Housing

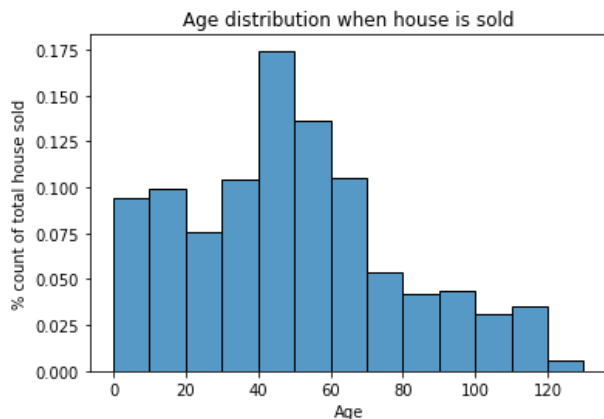


Moving on to the next step, we are trying to find the answers to the business questions stated in introduction.

What is the age distribution of price?

The age of house in Melbourne ranges from 0 to ~120 years old. Most of the house is between age 40-50 years old, accounted for 17.5%. Furthermore, more than half of the houses is below 50 years old.

Figure 8 – Age Distribution of Housing



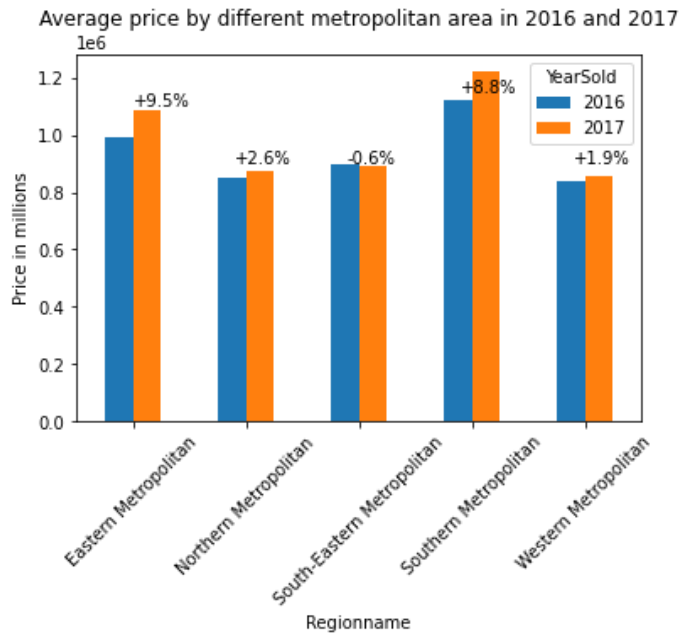
Which metropolitan area has the highest growth rate in property price?

It is shown in the bar plot (Figure 9 – Average Price of housing by Metropolitan regions in 2016 and 2017) that Eastern Metropolitan has the highest growth rate of 9.5%, followed by Southern Metropolitan area at 8.8%. However, minimal growth rate is observed in Northern and Western area.

Highest price regions in 2016 and 2017 remained the same as the highest growing metropolitan regions – Eastern and Southern area.

There was a drop in property price in South-Eastern Metropolitan area in 2017 vs previous year which declined by 0.6%.

Figure 9 – Average Price of housing by Metropolitan regions in 2016 and 2017

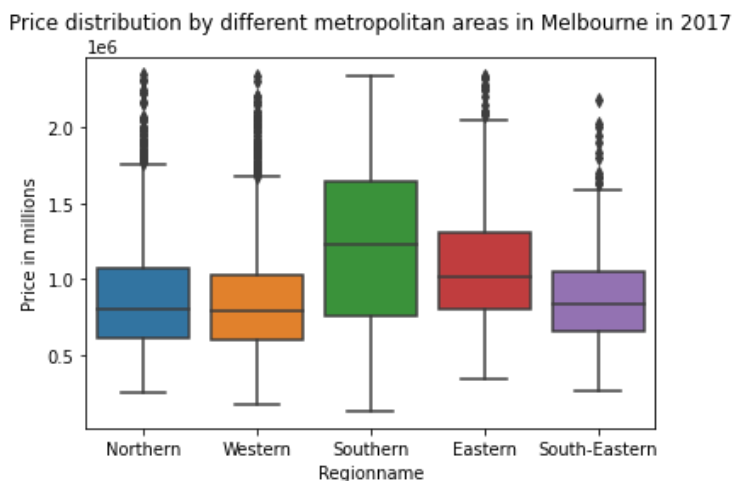


Does housing price differ between different metropolitan areas in Melbourne in 2017?

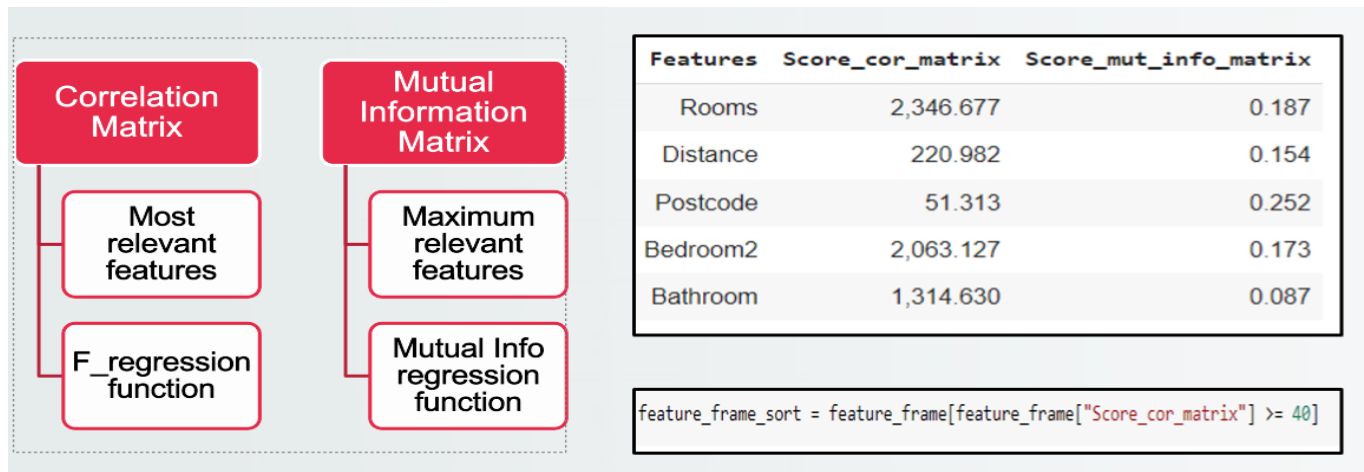
Southern Metropolitan region has the largest range vs other regions, indicating the largest dispersion of price. It has the longest box meaning middle 50% of observations scatter farther apart than other regions in price. In terms of median price, Southern region is the highest while Eastern region follows by.

Northern, Western and South-Eastern share some similarities in the price distribution regarding the median price, length of box - represents the middle 50% of sample and length of whiskers - represents the top and bottom 50% of sample.

Figure 10 – Price distribution by Metropolitan regions



FEATURE SELECTION



1. “For the **correlation statistic** we will use the [f_regression\(\) function](#). This function can be used in a feature selection strategy, such as selecting the top k most relevant features (largest values) via the [SelectKBest class](#).”
2. “The scikit-learn machine learning library provides an implementation of **mutual information** for feature selection with numeric input and output variables via the [mutual_info_regression\(\) function](#).”

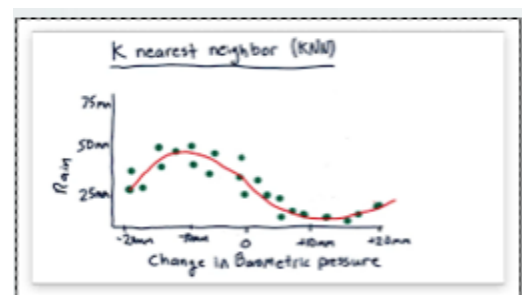
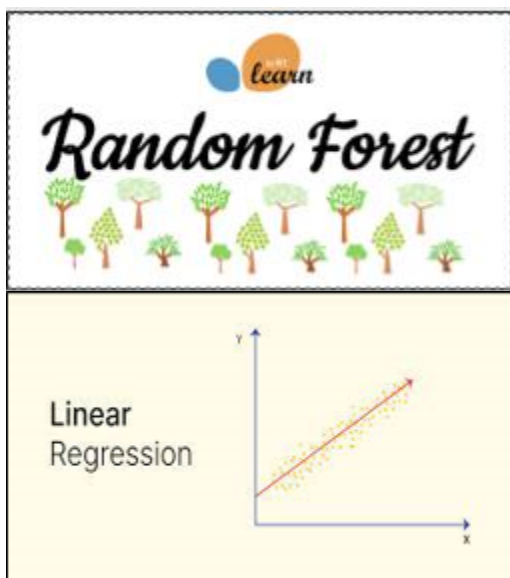
Let's try to understand these techniques more practically. Let's say I am buying a car. If I am buying a car with almost every feature, then its price will be almost double the one with minimum feature. Hence addition of each small feature has some major effect on price of car. But if I add some features in my house its price will go up, but it won't affect that much as major variables affecting the price of house are Land size, Location, Proximity and age of the house. Other factors like rooms, bathroom, built area, built type are the next features affecting the house price. But if I talk about the color of a house, it has no major effect on house prices. Hence, I must select a technique which can give me all the major features affecting the house price and that technique is correlation matrix. Hence, I am selecting all the features which has score greater than 40. This technique helped me to select all the major features affecting the house price.

What are the key drivers influencing the housing price in Melbourne?

- Rooms
- Distance
- Postcode
- Bedroom2
- Bathroom
- Car

- **Landsize**
- **Method** - 'MethodSP'
- **Latitude/Longitude**
- **Built_area_percent**
- **Suburb** - 'Suburb_Albert Park', 'Suburb_Ashburton', 'Suburb_Balwyn', 'Suburb_Balwyn North', 'Suburb_Brighton', 'Suburb_Brighton East', 'Suburb_Camberwell', 'Suburb_Glen Iris', 'Suburb_Glenroy', 'Suburb_Hampton', 'Suburb_Kew', 'Suburb_Kew East', 'Suburb_Malvern East', 'Suburb_Reservoir', 'Suburb_Sunshine West', 'Suburb_Surrey Hills'
- **Type** - 'Type_h', 'Type_u'
- **SellerG** - 'SellerG_Barry', 'SellerG_Buxton', 'SellerG_Fletchers', 'SellerG_Jellis', 'SellerG_Marshall', 'SellerG_Stockdale', 'SellerG_YPA'
- **YearBuilt** - 'YearBuilt_1900', 'YearBuilt_1910', 'YearBuilt_1920', 'YearBuilt_1925', 'YearBuilt_1930', 'YearBuilt_1960', 'YearBuilt_1970'
- **Council** - 'CouncilArea_Bayside', 'CouncilArea_Boroondara', 'CouncilArea_Brimbank', 'CouncilArea_Hume', 'CouncilArea_Manningham', 'CouncilArea_Maribyrnong', 'CouncilArea_Melton', 'CouncilArea_Moreland', 'CouncilArea_Whittlesea', 'CouncilArea_Wyndham'
- **Regionname** - 'Regionname_Northern Metropolitan', 'Regionname_Southern Metropolitan', 'Regionname_Western Metropolitan'

PREDICTIVE MODELS



House prices are reflective of the economic index and are of utmost importance to buyers as well as sellers. Real estate prices depend on a multitude of complex factors and are not always obvious.

This project aims to implement different regression models to solve the house price prediction problem, the models will be compared amongst one another based on performance metrics such as R-squared value, Mean Absolute Value, Root Mean Squared Value and Mean Squared Value. The model with the most accurate estimate and effectiveness will be identified to solve the problem statement.

The supervised machine learning algorithms are bifurcated into two categories –

- Regression algorithms – Used to predict a continuous value of dependent variables based on features of the training dataset.
- Classification algorithms - Used to predict a category for new observations based on the training dataset.

Since, our dependent variable ‘Price of a house’ is a continuous variable we will be implementing regression models to solve our problem. Below are the three models we intend to implement along with their rationale [\[2\]](#)[\[3\]](#)

1. Linear regression

- This classic supervised model is extremely simple to build and can be trained quickly. It is further classified into simple or multiple linear regression based on the number of independent variables.[\[4\]](#)
- It is proven to be accurate in pricing analysis in scenarios when a linear relationship exists between each independent variable and the dependent variable.
- We have implemented the linear regression model by splitting the dataset into train and test by 70:30.

```

1 train, test = train_test_split(df_model_dummies, test_size=0.30)
   ## train test split 70:30

[ ] df_model_dummies.columns
Index(['Rooms', 'Distance', 'Postcode', 'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'Latitude', 'Longitude', 'Propertycount',
      ...,
      'CouncilArea_Yarra', 'CouncilArea_Yarra Ranges', 'Regionname_Eastern Metropolitan', 'Regionname_Eastern Victoria', 'Regionname_Northern Metropolitan', 'Regionname_Northern Victoria', 'Regionname_South-Eastern Metropolitan',
      'Regionname_Southern Metropolitan', 'Regionname_Western Metropolitan', 'Regionname_Western Victoria'], dtype='object', length=754)

[ ] df_model_dummies.loc[:, df_model_dummies.columns != 'TARGET']

```

	Rooms	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	Latitude	Longitude	Propertycount	Built_area_percent	Suburb_Abbotsford	Suburb_Aberfeldie	Suburb_Airport West	Suburb_Albanvale	Suburb_Albert Park	Suburb_Albin	Suburb...
0	2	2.500	3.067.000	2.000	1.000	1.000	202.000	-37.800	144.998	4.019.000	0.493	1	0	0	0	0	0	
1	2	2.500	3.067.000	2.000	1.000	0.000	156.000	-37.808	144.993	4.019.000	0.506	1	0	0	0	0	0	
2	3	2.500	3.067.000	3.000	2.000	0.000	134.000	-37.809	144.994	4.019.000	1.000	1	0	0	0	0	0	
3	3	2.500	3.067.000	3.000	2.000	1.000	94.000	-37.797	144.997	4.019.000	0.493	1	0	0	0	0	0	
4	4	2.500	3.067.000	3.000	1.000	2.000	120.000	-37.807	144.994	4.019.000	1.000	1	0	0	0	0	0	
...
13574	3	16.500	3.049.000	3.000	2.000	2.000	256.000	-37.679	144.894	2.474.000	0.493	0	0	0	0	0	0	
13575	4	16.700	3.150.000	4.000	2.000	2.000	652.000	-37.906	145.168	7.392.000	0.493	0	0	0	0	0	0	
13576	3	6.800	3.016.000	3.000	2.000	2.000	333.000	-37.859	144.879	6.380.000	0.399	0	0	0	0	0	0	
13577	3	6.800	3.016.000	3.000	2.000	4.000	436.000	-37.853	144.887	6.380.000	0.493	0	0	0	0	0	0	
13579	4	6.300	3.013.000	4.000	1.000	1.000	362.000	-37.812	144.884	6.543.000	0.309	0	0	0	0	0	0	

12022 rows x 753 columns

- We then created X train, X test (input variables) and Y train Y test (output variables).

```

[ ] X_train = train.loc[:, train.columns != 'TARGET']
   y_train = train['TARGET']
   X_test = test.loc[:, test.columns != 'TARGET']
   y_test = test['TARGET']

1 X_train.head()

```

	Rooms	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	Latitude	Longitude	Propertycount	Built_area_percent	Suburb_Abbotsford	Suburb_Aberfeldie	Suburb_Airport West	Suburb_Albanvale	Suburb_Albert Park	Suburb_Albin	Suburb...
12950	3	14.000	3.047.000	3.000	1.000	2.000	500.000	-37.673	144.925	2.245.000	0.193	0	0	0	0	0	0	
5804	1	6.100	3.182.000	1.000	1.000	1.000	0.000	-37.961	144.963	13.240.000	0.493	0	0	0	0	0	0	
2538	2	1.600	3.065.000	2.000	1.000	1.000	0.000	-37.799	144.976	5.625.000	0.493	0	0	0	0	0	0	
6987	3	7.500	3.123.000	3.000	1.000	2.000	362.000	-37.834	145.050	6.482.000	0.359	0	0	0	0	0	0	
752	3	13.000	3.204.000	3.000	2.000	2.000	147.000	-37.932	145.035	6.795.000	1.000	0	0	0	0	0	0	

5 rows x 753 columns

```

[ ] y_train
12950    475,000,000
5804     475,000,000
2538    650,000,000
6987    1,420,000,000
752      827,000,000
...
9455    1,580,000,000
4231    1,002,500,000
2152     753,000,000
7562    1,800,000,000
18594    1,580,000,000
Name: TARGET, Length: 8415, dtype: float64

```

- We then trained the model using the training dataset and predicted Y (target variable - house price) and calculated the R-squared and P value metric

```

### Let's feed values in model

from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)

+ LinearRegression
LinearRegression()

[ ] y_predict_train = model.predict(X_train)

[ ] y_train[:10].ravel()

array([475000.000, 475000.000, 650000.000, 1420000.000, 825000.000,
       950000.000, 1255000.000, 1851000.000, 1285000.000, 968000.000])

[ ] y_predict_train[:10]

array([455264.952, 463496.731, 819121.554, 1550800.465, 1158461.438,
       758745.343, 1440720.635, 1364888.046, 1432087.668, 1255311.203])

[ ] y_train_list = y_train.tolist()
    y_pred_list = y_predict_train.tolist()

```

2. Random Forest

- This ensemble machine learning technique works by creating multiple decision trees and then returns the median prediction of individual decision trees as the outcome and preventing overfitting as compared to an individual decision tree, wherein the model uses a single data point and iteratively parses through the tree by answering true/false queries. Based on the outcome of the question distinct branches are created from the parent node, until the leaf node is reached. The entire tree is constructed using recursive partitioning.
- The effort required for data preparation is minimal when using Random Forest Model, as scaling or normalization is not required, it efficiently handles numerical, categorical, and binary features. Additionally, it also provides reliable indicators of key features and thus aids in performing implicit feature selection.
- Random Forest Models are immune to outliers and ignore statistical issues as opposed to other machine learning models which require normalization.

3. KNN (K-Nearest Neighbor)

- KNN comprises two hyperparameters K value (Number of neighbors) and a distance function (Euclidean, Manhattan etc).[\[5\]](#)
- In KNN regression a mean of the K nearest neighbors would be returned as the output. The algorithm is based on the assumption that any item in the data set should have a similar value for the prediction target if they share similar values for other features.
- In our particular case, the house price is the target variable that would be predicted by the k number of neighbors that are the most similar in terms of features, for example houses with similar land size and areas will have almost the same price.

INTERPRETATION / CONCLUSION

EDA Results

- The dataset contains 13,580 observations and 21 attributes in the data set. After missing value treatment, removing null values, outlier treatment and creating dummy variables - 12,022 rows and 22 variables remain for our upcoming model development.
- We have finalized three machine learning models for the price prediction problem – Linear Regression,
- Random Forest and KNN algorithm. We have implemented the below linear regression model and intend to build the other two models and compare them to study which model predicts house prices most accurately.

Evaluation of Modelling

Evaluation Matrix	Linear Regression	K-Nearest Neighbour	Random Forest Regressor
R-SQUARE	0.834	0.870	0.907
ADJUSTED R-SQUARE	0.833	0.817	0.906
MAE	186110.96	189667.65	131941.38

- The model produced the below R-squared and P-value on train and test data set
- R2 (R-square) - The ratio of the explained variance and the overall variance is known as the coefficient of determination, or R2. Another, possibly more intuitive way to think of it is as the percentage of variance that the model can predict or describe. It is frequently used to evaluate how well a regression model fits the data. The coefficient ranges from 0 to 1, with 1 being the highest., wherein 0 denotes no decision and better denotes perfect determination. This model gives a R2 value of 0.88 for train data and 0.86 for test dataset.
- Among the three models we can clearly see that all the three models have overall good R-value but if we look more closely, we can see that Random Forest regressor is one of the best algorithms as R-square value for train and test data both have almost same score also Adjust R-square value is in line with features hence Random Forest regressor is one of the best algorithms among three.

REFERENCES

- [1] <https://www.kaggle.com/datasets/dansbecker/melbourne-housing-snapshot>
- [2] <https://towardsdatascience.com/machine-learning-project-predicting-boston-house-prices-with-regression-b4e47493633d>
- [3] <https://www.sciencedirect.com/science/article/abs/pii/S0957417414007325>
- [4] <https://www.ibm.com/topics/linear-regression#:~:text=Linear%2Dregression%20models%20have%20become,can%20be%20trained%20very%20quickly>
- [5] <https://renanmf.com/data-science-machine-learning-house-prices/>
- [6] <https://stackoverflow.com/questions/59623109/find-out-unique-categories-in-all-categorical-variable>