Melbourne Housing Market Analysis

by Adnan, Alvaro, Ikenna, Jasdeep, Jing, and Zach.

Abstract

This is a Real Estate related project that aims to provide a recommendation for a potential buyer about where to buy a 2 bedroom apartment in Melbourne, Australia. We looked to develop a model that would help us to predict the price variations in different areas of Melbourne considering different variables.

We conducted some data manipulation to determine the relevant input variables that have more impact on future pricing, we also dropped those who were not relevant, and then we analyzed all data points considering metropolitan areas in Melbourne for classification, thus being able to make a recommendation for our client.

Introduction

Our group will analyze the Melbourne Housing Market dataset with the goal of producing a predictive algorithm that will forecast the price of a subset of this data Our goal is to forecast the price of all homes within the Melbourne Housing Market at the beginning of 2019.

A regression algorithm will allow us to predict a continuous target variable, the future price of properties sold. We will focus on evaluating different types of regressions in Python to see if we can obtain valid train and test scores to increase the potential of a successful prediction.

A successful algorithm will allow us to compute an average of future home sales to understand if the overall market is increasing or decreasing in value. There is a consensus that the Melbourne housing market is heading into a cooling period, where the prices will decrease. We hope to build a model that confirms this and that might provide additional information to potential investors on areas of the city or types of home that might still be good investments in this market. The data covers sales from 2016 and 2017, our model will predict the value of these homes at the beginning 2019.

Our goal is to create a target variable that will allow to understand if a property is likely to see a decrease or an increase in value in 2019. Upon successful creation this variable, we can look to identify what important property attributes or location considerations can allow an investor to minimize the potential for a loss on their investment.

Business Understanding

Background

This data was scraped from publicly available results posted every week from Domain.com.au. The data set was previously cleaned but we still needed to conduct some data preparation and manipulation. The data set includes variables such as Address, Type of Real Estate, Suburb, Method of Selling, Rooms, Price, Real Estate Agent, Date of Sale, and distance from C.B.D.

The dataset used in this analysis is Melbourne housing clearance data from Kaggle.com. The data includes 21 attributes or columns and 34,857 rows of data.

Source: https://www.kaggle.com/anthonypino/melbourne-housing-market#Melbourne_housing_FULL.csv

Attribute	Description
Suburb	Suburb
Address	Address
Rooms	Number of rooms
Price	Price in Australian dollars
Method	S - property sold; SP - property sold prior; PI - property passed in; PN - sold prior not disclosed; SN - sold not disclosed; NB - no bid; VB - vendor bid; W - withdrawn prior to auction; SA - sold after auction; SS - sold after auction price not disclosed. N/A - price or highest bid not available.
Type	h - house, cottage, villa, semi-terrace; u - unit, duplex; t - townhouse; dev site - development site; o res - other residential.
SellerG	Real Estate Agent
Date	Date sold
Distance	Distance from CBD in Kilometres
Regionname	General Region (West, North West, North, North east ...etc)
Propertycount	Number of properties that exist in the suburb.
Bedroom2	Scraped # of Bedrooms (from different source)
Bathroom	Number of Bathrooms
Car	Number of carspots
Landsize	Land Size in Metres
BuildingArea	Building Size in Metres
YearBuilt	Year the house was built
CouncilArea	Governing council for the area
Latitude	Self explanatory
Longitude	Self explanatory
PropertyCount	Integer, unclear on purpose

Business Objectives

Primary Business Objective

To make a recommendation to our client about where in Melbourne buy a 2 bedroom apartment based on future return of investment.

Related Business Questions

- Are there a few areas in Melbourne that consistently present a higher Real Estate value?
- What are the features that impact the most the future valuation of an apartment?
- Will higher prices in the present ensure that the buyer will get a better ROI in the future?

Business Success Criteria

A successful model will allow us to predict future home pricing in Melbourne to understand if the overall market is increasing or decreasing in value. We will look to understand if an investor could maximize their profits by selecting properties with certain attributes or within certain areas of the city.

Assess Situation

As prerequisites the data set must have at least 10,000 rows of data, with enough values entered for comprehensibility and quality of results, and we have received clearance that we have authorization to use this data for the project.

As constraint of the project we have only this available data, so we might need to use external sources of information for future developments of this project.

Analytical Objectives

- To identify high correlations across variables to discard those irrelevant for the analysis.
- To determine the variables that impact the most, the output (pricing) variable.
- To find the regression model with the best fit to help us predict the kind of property with better ROI.

Project Plan

Given that our Team has already been provided with the data set for the project, most of our project plan will focus on description, exploration and quality assurance of the data.

For most of these tasks we will make use of Pandas functions to initially describe the data, as well as visualization tools such as plots and columns and bars' charts.

As part of the exploration and quality assurance tasks we will also use description functions and data frames to make sure that the transformed data is consistent and yields reliable results.

To read the data set and perform our analysis we loaded the libraries and set up the color coding below:

```
import numpy as np
import pandas as pd
import time
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt

mycolors = ["#771C19", "#AA3929", "#8E9CA3", "#556670", "#000000", \
            "#E25033", "#F27314", "#F8A31B", "#E2C59F", "#B6C5CC", \
            "#99CCCC","#FFCC99"]

%matplotlib inline

df_raw = pd.read_csv('./MELBOURNE_HOUSING_FULL.csv')
print(df_raw.shape)

# Define only two digits to display
pd.options.display.float_format = '{:,.2f}'.format

# Delete all missing price rows
df_raw = df_raw.dropna(subset=['Price'])
```

Numpy and Pandas

Numpy is an open source Python library used for scientific computing and provides a host of features that allow a Python programmer to work with high-performance arrays and matrices. In addition, pandas is a package for data manipulation that uses the DataFrame objects from R (as well as different R packages) in a Python environment. (Eric van Rees, n.d.)

Time

The time function was used as we were using a 'datetime' variable in our model to predict properties pricing over time.

Seaborn, Matplotlib and Matplotlib.pyplot

These three libraries were used for charting and plotting the data throughout our analysis.

Data Understanding

Data Collection

The data was previously collected and provided by the client.

Source: https://www.kaggle.com/anthonypino/melbourne-housing-market#Melbourne_housing_FULL.csv

We found that there were some cells with empty values and some data categories that would need further transformation for readability and use for analysis.

Data Selection

Target Variable: Price. The Melbourne housing market data shows property sales information from the period of January 2016 to March 2018. In this analysis the aim is to forecast the Price variable at a future date. This is a continuous variable, and our output will be Predicted Price.

Data Cleaning

Data upload and Rounding of decimals

Data uploading of the raw source .CSV file for our analysis was conducted through the pd.read_csv function of pandas. To reproduce this upload the user will need to ensure the source file and the Python workbook are opened from the same file path folder.

Deletion of Price Data

As the target variable was price and the main goal of this forecasting algorithm was to forecast price at a future date, a decision was made to delete all data that did not contain price in the original dataset.

Duplicate Data

A check for duplicate data in our code confirmed that no duplicate data rows existed. The function reviews the dataset for duplicate data then it counts the occournaces of eith "True" of "False" in the output. As there are no "True" items, this is confirmation of no duplicate data rows.

```
#check for duplicate data
duplicate = df_raw.duplicated()
duplicate.value_counts()
```

```
False    27247
dtype: int64
```

```
# Data preprocessing

from sklearn.impute import SimpleImputer

# Categroy variables
class_variable = [col for col in df_raw.columns if df_raw[col].dtypes == 'O']

# Numerical variables
numerical_variable = [col for col in df_raw.columns if df_raw[col].dtypes != 'O']
print('Catagory variables: %s' % class_variable, '\nNumerical variables: %s' % numerical_variable)

padding = SimpleImputer(strategy='mean')
# Fill the null numerical varibles with mean values
df_raw[numerical_variable] = padding.fit_transform(df_raw[numerical_variable])

# Fill the null category variables with None
df_raw[class_variable] = df_raw[class_variable].fillna('None')

# Convert Rooms, Bathroom, Car, YearBuilt to integer
df_raw[['Rooms', 'Bathroom', 'Car', 'YearBuilt']] = df_raw[['Rooms', 'Bathroom', 'Car', 'YearBuilt']].applymap(np.int64)
```

```
Catagory variables: ['Suburb', 'Address', 'Type', 'Method', 'SellerG', 'Date', 'CouncilArea', 'Regionname']
Numerical variables: ['Rooms', 'Price', 'Distance', 'Postcode', 'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea',
'YearBuilt', 'Lattitude', 'Longtitude', 'Propertycount']
```

Loading of Library

We loaded a new library called SimpleImputer which will be used to input values where we have missing numerical and object data. First we created two variables, one that obtained all of the Categorical variables which we identified based on a column datatype being equal to an "object" in Python; we called this first variable "class_variable". We then continued to create a second variable called "numerical_variable", we identified these as all columns that were not Python "Objects" this left only floats and integers.

Creating a function called "padding" we set the parameter for the SimpleImputer function to mean. We then used this function to input the mean of those columns into the rows with missing information. For our categorical variables we filled the missing information with "none".

We then converted some of the numerical data to integers. Items such as Rooms, Bathrooms, Car and YearBuilt were imported as floats however these variables are better represented as integers as they are counts of features of a home.

Data Attribute Summary

We ran the df_describe() function in Python to obtain some basic descriptive statistics on the numerical data within the dataset. The function shows the count of the data, the mean the interquartile ranges and the min and max values of each numerical column of data in the dataset.

```
# obtiain descriptive statistics to the above dataset
df_working.describe ()
```

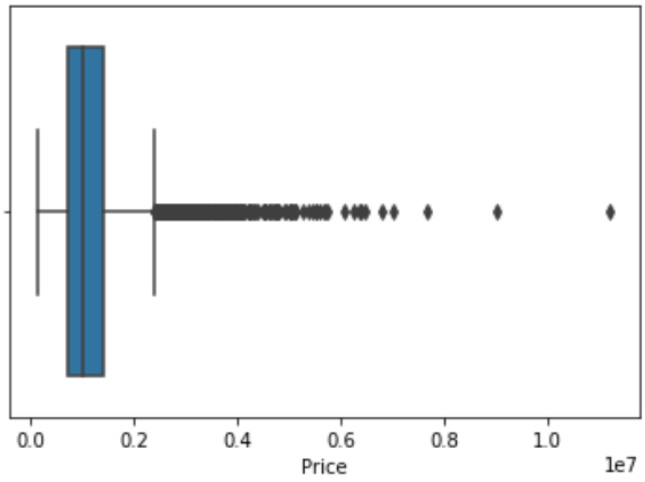
	Rooms	Price	Distance	Postcode	Bathroom	Car	Landsize	BuildingArea	YearBuilt
count	17,019.00	17,019.00	17,019.00	17,019.00	17,019.00	17,019.00	17,019.00	17,019.00	17,019.00
mean	3.18	1,154,715.17	12.32	3,115.60	1.49	1.68	619.68	157.87	1,962.38
std	0.65	621,689.48	7.20	126.45	0.62	1.00	3,707.88	71.84	26.05
min	2.00	131,000.00	1.20	3,000.00	0.00	0.00	0.00	0.00	1,196.00
25%	3.00	731,000.00	7.40	3,044.00	1.00	1.00	443.00	154.00	1,965.00
50%	3.00	995,000.00	11.20	3,082.00	1.00	2.00	593.49	156.83	1,966.00
75%	4.00	1,400,000.00	14.80	3,149.00	2.00	2.00	632.00	156.83	1,966.00
max	4.00	11,200,000.00	48.10	3,978.00	8.00	18.00	433,014.00	6,791.00	2,019.00

Data Preparation

We then generated a boxplot diagram to see if we had any outliers in our dataset. It was clear that outliers existed in our dataset that needed to be addressed.

```
sns.boxplot(x=df_working['Price'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1de5e59b0c8>
```



Outlier Handling

The Tukey Method, first introduced by John W. Tukey, was used to define the boundary for outliers and all data that fell outside this range was removed. The formula takes the interquartile range (IQR), the difference between Q3 and Q1 of the data, and applies a factor of 1.5 to the IQR. It then adds 1.5IRQ to the Q3 point to set the upper boundary and subtracts 1.5IQR from the Q1 to set the lower boundary. (Seo, 2002)

Afterwards we ran the boxplot function again on the data with the outliers removed based on the method above. We were able to show a more meaningful boxplot and outliers appears to have been properly identified and removed.

```
#formula used to calculate the interquartile range of data
Q1 = df_working.Price.quantile(0.25)
Q3 = df_working.Price.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

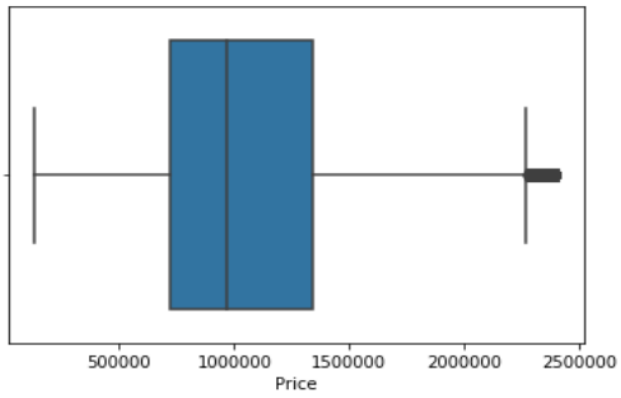
```
669000.0
```

```
# Functions to calculate the upper and lower limits that will define outliers
upIQR = Q3+(IQR*1.5)
lowIQR = Q1-(IQR*1.5)
print(upIQR)
print (lowIQR)
```

```
2403500.0
-272500.0
```

```
#New boxplot with outliers removed.
sns.boxplot(x=df_working2['Price'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1de5d89b8c8>
```



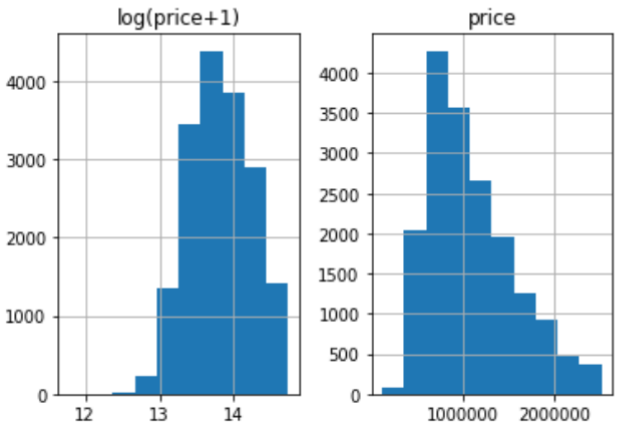

```
# Formula to drop all outliers from dataset.
df_working2=df_working.drop(df_working[(df_working.Price>upIQR)|(df_working.Price<lowIQR)].index)
```

Data Normalization

When we check a histogram of our price data, we noticed that our data was quite skewed. We decided to look at a histogram of our original price data and a histogram of the price data when a formula (Log +1) was applied to the price. The log normalization method produced a more normal distribution of our price data. We will apply this to our data when building our model.

```
prices = pd.DataFrame({"price":df_working2.Price, "log(price+1)":np.log1p(df_working2.Price)})
prices.hist()
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001EF9A2A9A88>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001EF9B918248>]],
      dtype=object)
```



Modeling

Based on the business requirements, we only looked at data for houses in the Melbourne Housing market. A formula was necessary to select only the necessary data:

```
# Create a working dataframe that only contains only houses.
df_working = df_resultdrop[(df_resultdrop.Type == 'h')]
```

```
df_working.head(5)
```

	Suburb	Address	Rooms	Type	Price	Date	Distance	Postcode	Bathroom	Car	Landsize	BuildingArea	YearBuilt	CouncilArea	Reg
1	Abbotsford	85 Turner St	2	h	1,480,000.00	03/12/2016	2.50	3,067.00	1	1	202.00	156.83	1966	Yarra City Council	Me
2	Abbotsford	25 Bloomburg St	2	h	1,035,000.00	04/02/2016	2.50	3,067.00	1	0	156.00	79.00	1900	Yarra City Council	Me
4	Abbotsford	5 Charles St	3	h	1,465,000.00	04/03/2017	2.50	3,067.00	2	0	134.00	150.00	1900	Yarra City Council	Me
5	Abbotsford	40 Federation La	3	h	850,000.00	04/03/2017	2.50	3,067.00	2	1	94.00	156.83	1966	Yarra City Council	Me
6	Abbotsford	55a Park St	4	h	1,600,000.00	04/06/2016	2.50	3,067.00	1	2	120.00	142.00	2014	Yarra City Council	Me

A check confirms that only items with Type equal to h, was included in the working dataset.

```
# Additional Data Pre-processing on dataset

df_new = df_working2[['Rooms', 'Price', 'Distance', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'YearBuilt']].copy()
```

In the data above, we created a new dataset that would be used in the predictive model. We selected 'Rooms', 'Price', 'Distance', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', and 'YearBuilt' as our input variables. This was based on our correlation matrix which showed that these variables were sufficiently correlated to price. This would ensure that they would provide benefit to the model when used as input variables. Variables with a correlation < .1 were not included as input variables. One numerical variable, titled "PropertyCount" did not provide any unique information. It was the same value across all fields, this was not included in the regression.

The following new variables were created:

```
# Extract year from Date
df_new.loc[:,'Year'] = pd.DatetimeIndex(df_new.Date).year
# The years between the predicted and the estimated, we assume the predicted years is 2019
df_new.loc[:,'N'] = 2019 - df_new.Year
# The age of house
df_new.loc[:,'Age'] =  df_new.Year - df_new.YearBuilt
```

Year – this was used to extract only the year of the sale, this would be used in a calculation of the Age of the property as well as to obtain information on when the house was sold. This will be used later in the model to calculate the per year change in price of the property values being predicted.

N – a column in the data that was created to calculate the difference between the predicted year and the sale of the property in the regression.

Age – The age of the house, a calculate that calculates the difference between the year the house was built and the year the house was sold.

Test Design

Defining Train and Test Data

```
from sklearn.model_selection import train_test_split

df_train = df_new[['Rooms', 'Distance', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', \
                   'Age', 'Type']].copy()

df_target = np.log1p(df_new.Price)

validation_size = 0.15

x_train, x_test, y_train, y_test = train_test_split(df_train, df_target, test_size=validation_size)
```

Utilizing the train_test_split library from sklearn.model_selection library required that we call the specific function within the library.

A new data frame is copied with all of our numeric input variables. In our model we call this df_train.

Our target variable, called df_target is defined, however we apply the np.log1p function to our variable, we had previously tested that doing this will normalize our data and provide for a more successful regression.

The validation_size variable that we created will allow us to split our data based on a percentage. In our initial tests of our models a test set of 15% and a training set of 85% seem to provide good results with the final model that was used to predict future prices.

The final line of code above splits the actual data into two sets, x_train, x_test and y_train, y_test. It defines are test size, based on the variable (validation_size).

Modeling Technique

Linear Regression Model

```
# Linear regression model test
from sklearn.linear_model import LinearRegression

lreg = LinearRegression()

lreg.fit(x_train,y_train)

pred_cv = lreg.predict(x_test)

mse = np.mean((pred_cv - y_test)**2)

print('The Linear regression mse:{0: .4f}'.format(mse))

train_score = lreg.score(x_train,y_train)
test_score = lreg.score(x_test,y_test)
print('train score:{0:.4f} test score:{0:.4f}'.format(train_score,test_score))
```

```
The Linear regression mse: 0.1105
train score:0.3855 test score:0.3855
```

For the linear regression model model test; we utilized the sklearn.linear_model library we imported the LinearRegression package

A variable was created to denote the Linear Regression function we then fit our x_train and y_train datasets that were previously created in the train test split.

A function was used call pred_cv which applied the linear regression formula to our test data. The error was calculated based on the difference between the predicted data and the actual data used in the test.

Finally the results were calculated for the train and test scores and they were displayed along with a print of the error or mse.

As we can see from the results of the test, this regression model is not quite ideal. We have a less than ideal error in this regression but more importantly, our train and test scores are only approaching 38%. If the issue with this model is that the data is not linear enough, we may have more success looking to a different model to achieve better results.

Elastic Net Regression

```
# Elastic Net regression model test
from sklearn.linear_model import ElasticNet


ENreg = ElasticNet(alpha=1, l1_ratio=0.5, normalize=False)

ENreg.fit(x_train,y_train)

pred_cv = ENreg.predict(x_test)

mse = np.mean((pred_cv - y_test)**2)

print('The Elastic Net regression mse:{0: .4f}'.format(mse))

train_score = ENreg.score(x_train,y_train)
test_score = ENreg.score(x_test,y_test)

print('train score:{0:.4f} test score:{0:.4f}'.format(train_score,test_score))
```

```
The Elastic Net regression mse: 0.6859
train score:0.2471 test score:0.2471
```

Our next model followed the same steps we called the function fit our data and tested our errors and calculated our train and test scores. We had an extremely high error with the elastic net regression and we continued to see poor train and test scores. Adjusting parameters had little impact on the overall test scores and we moved on to another regression test.

Random Forest Regression

```
# Random Forest regression model test
from sklearn.ensemble import RandomForestRegressor

RandForestReg = RandomForestRegressor(n_estimators=10, random_state=7)

RandForestReg.fit(x_train,y_train)

pred_cv = RandForestReg.predict(x_test)

mse = np.mean((pred_cv - y_test)**2)

print('The Random Forest mse:{0: .4f}'.format(mse))

train_score = RandForestReg.score(x_train,y_train)
test_score = RandForestReg.score(x_test,y_test)

print('train score:{0:.4f} test score:{0:.4f}'.format(train_score,test_score))
```

```
The Random Forest mse: 0.0692
train score:0.9013 test score:0.9013
```

The Random Forest Regression provided a low error score and high train and test scores. Our mean standard error score was .0682 and our train and test scores were .9013. The means that our training data can accurately predict based on our model what the test data provided as an output for our target variable.

Following these positive test scores we had selected the model for which to build our predictive algorithm.

```
pred_cv = RandForestReg.predict(df_train)
pred_price = np.expm1(pred_cv)
df_pred = pd.DataFrame(pred_price, columns=['PredictedPrice'])
df_pred = pd.concat([df_working2, df_pred], axis=1)
df_pred['PriceChange'] = (df_pred['PredictedPrice'] - df_new.Price)/ df_new.N
df_pred.drop(['Suburb', 'Address', 'Postcode', 'Date', 'Price', 'CouncilArea'], axis = 1, inplace = True)
```

Using the Random Forrest Regression function we applied this to our entire dataset. We had originally multiplied our data but (log+1) in our target variable, so in order to accurately present the predicted price, we need to apply the inverse of this to our pred_price variable. Which was done using the function np.exmp1(pred_cv).

We then add the column to the dataset to display all of the predicted price data and we concatenate this to our original dataset. A calculation was done to get the one year increase in the overall market from the end of 2018 to the beginning of 2019, this formula takes the predicted price, subtracts the sale price and divides by the number of year since the sale.

Other important attributes are added to the data in the final line of code. Overall the model did accomplish its goal and it was able to predict that overall the housing market declined by ($32,766.55).

```
#Average change accross all homes in the dataset.
print('The average change in price for homes in Melbourne is: {0: .2f}'.format(df_result['PriceChange'].mean()))
```

```
The average change in price for homes in Melbourne is: -32766.55
```

Model Development (Classification)

A classification model was created to understand which factors or property attributes were the most important in ensuring that an investor looking to purchase a home in Melbourne would not lose money on their investment over this period.

In python a function used that created a new column of data called ‘investment’ which gave a binary value of 1 if the price was greater than 0 and a value of 0 if the price was less than 0. This would later be used in the decision tree to predict which investments would result in positive ‘PriceChange’ values.

```
df_resultdrop['Investment'] = [1 if x > 0 else 0 for x in df_resultdrop.PriceChange]
df_resultdrop.head()
```

```
C:\Users\adnan\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: 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: http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

Rooms	Type	Price	Distance	Postcode	Bathroom	Car	Landsize	BuildingArea	YearBuilt	Regionname	PredictedPrice	PriceChange	Investment
2.00	h	1,480,000.00	2.50	3,067.00	1.00	1.00	202.00	156.83	1,966.00	Northern Metropolitan	1,022,685.44	-152,438.19	0
2.00	h	1,035,000.00	2.50	3,067.00	1.00	0.00	156.00	79.00	1,900.00	Northern Metropolitan	1,271,725.45	78,908.48	1
3.00	h	1,465,000.00	2.50	3,067.00	2.00	0.00	134.00	150.00	1,900.00	Northern Metropolitan	1,556,323.70	45,661.85	1
3.00	h	850,000.00	2.50	3,067.00	2.00	1.00	94.00	156.83	1,966.00	Northern Metropolitan	1,260,833.54	205,416.77	1
4.00	h	1,600,000.00	2.50	3,067.00	1.00	2.00	120.00	142.00	2,014.00	Northern Metropolitan	1,870,214.01	90,071.34	1

R was utilized to create a classification model that would outline important input variables and predict the best homes to invest in based on the variables in the dataframe. To extract the data from Python, we used the .to_csv to write a new .csv file from our dataset. This was saved and uploaded into R.

Prior to creating the .csv file the drop.na function was executed on the dataset, eliminating all rows with NaN or 'null' information. This was to ensure a proper execution of the decision tree within R.

```
df_resultdrop.to_csv(r'C:\Users\adnan\OneDrive\Documents\Big Data Analytics\Project Files\Melbourne\result.csv')
```

Model Assessment

Loading the Data and Libraries

Dataset was loaded into R and a seed was set to ensure reproducibility of results.

The following libraries were called to handle the creation of the decision tree and the corresponding testing of its accuracy:

```
Melbourne<-result
head(Melbourne)

#loading libraries necessary for the decision tree
library(rpart)
library(rpart.plot)

#setting seed to ensure reproducability of results
set.seed(100)
```

Defining Train and Test Data

The data was collected and indexed based on the date of sale for the rows of data within the dataset. In R it is necessary to first shuffle this data to ensure that the data split was taken as a random sample of all data.

A function to shuffle the data was applied to the dataset.

```
#Shuffling index to ensure that rows are selected randomly
shuffle_index <- sample(1:nrow(Melbourne))
head(shuffle_index)
Melbourne <- Melbourne[shuffle_index, ]
```

As the data was initially collected and indexed based on date, the splitting of the data in R required an initial shuffling of the data. Data was split into training and testing datasets based on an 80/20 split and the train and test datasets were defined.

```
#splitting data into a sample .80/.20 split
sample = sample.split(Melbourne$X, SplitRatio = .80)

#Defining the train and test data
train = subset(Melbourne, sample == TRUE)
test  = subset(Melbourne, sample == FALSE)
```

Decision Tree model

The training data was fit to the classification model and was plotted. The target variable was the binary value which represents either an increase in predicted property value (1) or a decrease in predicted property value (0). The input variables are Rooms, Distance, Regionname, and PredictedPrice.

```
#fitting the tree, defining input and output variables
fit <- rpart(Investment~Rooms+Distance+Regionname+PredictedPrice, data=train, method = 'class')

#plotting the decicion tree, additional parameters set.
rpart.plot(fit,type=01,yesno=2,tweak=.8,compress=TRUE)
```

Results

Accuracy and Testing of Model

Predict function was used to predict the efficacy of the model to the test data. The confusion matrix shows the successful predictions and the errors in their respective columns. 149 instances of a false positive (where property values were below 0 and the model predicted they were above 0. 161 instances showed that the values were predicted negative and were in fact, positive. The accuracy for the overall test was acceptable and our model proved to be accurate approximately 81.3% of the time

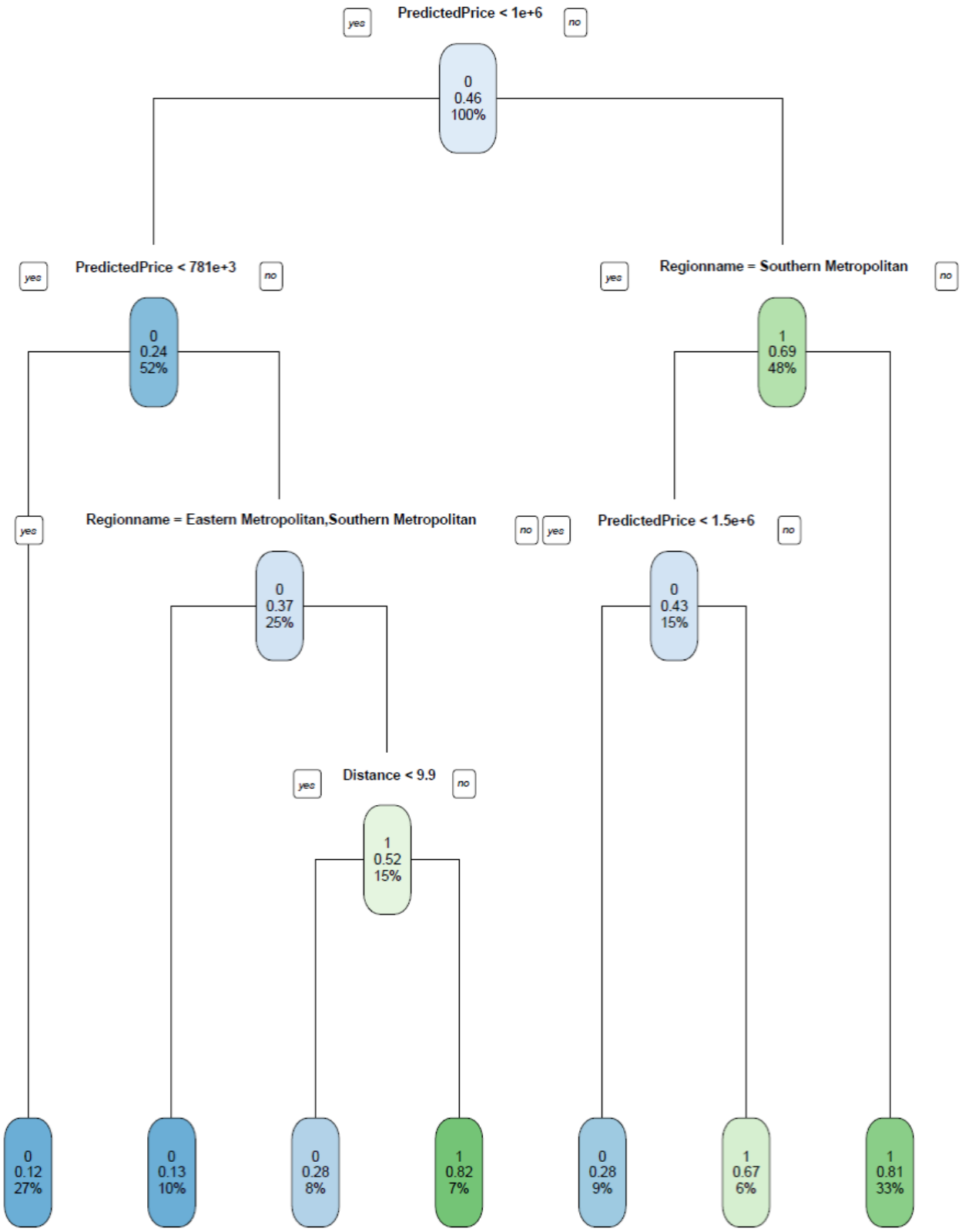
Input

```
#function to predict my test dataset based on the model
predict <-predict(fit, test, type = 'class')

#create a table of test results and prediction
table_mat <- table(test$Investment, predict)
table_mat

#calculate prediction success based on the sum of error divided by the sum of the entire table
accuracy_Test <- sum(diag(table_mat)) / sum(table_mat)
print(paste('Accuracy for test', accuracy_Test))
```

Output

```
table_mat
  predict
     0   1
0 738 149
1 161 609
```

```
> print(paste('Accuracy for test', accuracy_Test))
[1] "Accuracy for test 0.812914906457453"
```


PredictedPrice < 1e+6
yes
no
0
0.46
100%
PredictedPrice < 781e+3
yes
no
0
0.24
52%
Regionname = Southern Metropolitan
yes
no
1
0.69
48%
Regionname = Eastern Metropolitan,Southern Metropolitan
yes
no
0
0.37
25%
PredictedPrice < 1.5e+6
no
yes
no
0
0.43
15%
Distance < 9.9
yes
no
1
0.52
15%
0
0.12
27%
0
0.13
10%
0
0.28
8%
1
0.82
7%
0
0.28
9%
1
0.67
6%
1
0.81
33%

Model Conclusions

The model predicted that 81% of all houses purchased that had a PredictedPrice>1,000,000 AUD and were not located in the Southern Metropolitan area of Melbourne will not lose value as an investment within the Melbourne Housing market. The model also predicted that within the Southern Metropolitan area, homes that had a price greater than 1,500,000 AUD had a 67% chance of holding their value between the sold date and the predicted year, 2019.

Homes that were less than 1,000,000 and greater than 781,000 AUD that were not located in the Eastern Metropolitan and Southern Metropolitan areas and were greater than 9.9KM from the city centre had an 82% chance of at least holding their value through to the prediction year.

References

Eric van Rees. (n.d.). *pandas and NumPy arrays explained*. Retrieved from medium.com: https://medium.com/@ericvanrees/pandas-series-objects-and-numpy-arrays-15dfe05919d7

Seo, S. (2002). *A Review and Comparison of Methods for Detecting Outliers*. Kyunghee University.

Tukey, J. (n.d.). Exploratory Data Analysis. Addison-wesley, 1977.

Wikipedia. (n.d.). *Wikipedia.org*. Retrieved from https://en.wikipedia.org/wiki/Multicollinearity