!!Capstone Project!!

BY

> **1. Vignya Durvasula 20P61A6615**

> **2. Shreejit Cheela 20P61A6647**

Import Library

```
import pandas as pd#for data manipulation
import numpy as np#for working with arrays in the dataset
import seaborn as sns#for dtatistical data visualisation
import matplotlib.pyplot as plt#for dynamic data visualisation
```

Import Dataset

```
df=pd.read_csv('https://github.com/ybifoundation/Dataset/blob/main/Melbourne%20Housing%20Market.csv?raw=true')#importing the dataset
```

```
df#displaying the dataframe
```

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	Postcode	...	Bathroom	Car	Lands
0	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin	3/12/2016	2.5	3067.0	...	1.0	1.0	20
1	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin	4/02/2016	2.5	3067.0	...	1.0	0.0	15
2	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin	4/03/2017	2.5	3067.0	...	2.0	0.0	13
3	Abbotsford	40 Federation La	3	h	850000.0	PI	Biggin	4/03/2017	2.5	3067.0	...	2.0	1.0	9
4	Abbotsford	55a Park St	4	h	1600000.0	VB	Nelson	4/06/2016	2.5	3067.0	...	1.0	2.0	12
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
13575	Wheelers Hill	12 Strada Cr	4	h	1245000.0	S	Barry	26/08/2017	16.7	3150.0	...	2.0	2.0	65
13576	Williamstown	77 Merrett Dr	3	h	1031000.0	SP	Williams	26/08/2017	6.8	3016.0	...	2.0	2.0	33
13577	Williamstown	83 Power St	3	h	1170000.0	S	Raine	26/08/2017	6.8	3016.0	...	2.0	4.0	43
13578	Williamstown	96 Verdon St	4	h	2500000.0	PI	Sweeney	26/08/2017	6.8	3016.0	...	1.0	5.0	86

Data Preprocessing

```
df.shape#shape built-in function is used to know the total number of rows and columns in the dataset
```

```
(13580, 21)
```

```
df.nunique()#represents the number of unique values under each attribute
```

```
Suburb             314
Address          13378
Rooms                9
Type                 3
Price             2204
Method               5
SellerG            268
Date                58
Distance           202
Postcode           198
Bedroom2            12
Bathroom             9
Car                 11
Landsize          1448
BuildingArea       602
YearBuilt          144
CouncilArea         33
Lattitude         6503
Longtitude        7063
Regionname           8
Propertycount      311
dtype: int64
```

```
df.columns#displaying column names
```

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

```
df.info()#displays the basic information of the dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13580 entries, 0 to 13579
```

```
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Suburb         13580 non-null  object
 1   Address        13580 non-null  object
 2   Rooms          13580 non-null  int64
 3   Type           13580 non-null  object
 4   Price          13580 non-null  float64
 5   Method         13580 non-null  object
 6   SellerG        13580 non-null  object
 7   Date           13580 non-null  object
 8   Distance       13580 non-null  float64
 9   Postcode       13580 non-null  float64
 10  Bedroom2       13580 non-null  float64
 11  Bathroom       13580 non-null  float64
 12  Car            13518 non-null  float64
 13  Landsize       13580 non-null  float64
 14  BuildingArea   7130 non-null   float64
 15  YearBuilt      8205 non-null   float64
 16  CouncilArea    12211 non-null  object
 17  Lattitude      13580 non-null  float64
 18  Longtitude     13580 non-null  float64
 19  Regionname     13580 non-null  object
 20  Propertycount  13580 non-null  float64
dtypes: float64(12), int64(1), object(8)
memory usage: 2.2+ MB
```

```
df['Date']=pd.to_datetime(df['Date'])#converting the date attribute from object datatype to date datatype
```

Adjusting Missing Values

```
df.drop('CouncilArea',axis='columns', inplace=True)#Because its of object datatype moreover we already many other attributes that ref
```

```
df['BuildingArea'].fillna(df.groupby(['Type'])['BuildingArea'].transform('mean'), inplace=True)#filling the missing values in a parti
df['YearBuilt'].fillna(df.groupby(['Type'])['YearBuilt'].transform('mean'), inplace=True)
```

```
df['Car'].fillna(df.groupby(['Type'])['Car'].transform('mean'), inplace=True)
```

```
df.info()#One can observe that the missing values are all now filled up
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13580 entries, 0 to 13579
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Suburb         13580 non-null  object
 1   Address        13580 non-null  object
 2   Rooms          13580 non-null  int64
 3   Type           13580 non-null  object
 4   Price          13580 non-null  float64
 5   Method         13580 non-null  object
 6   SellerG        13580 non-null  object
 7   Date           13580 non-null  datetime64[ns]
 8   Distance       13580 non-null  float64
 9   Postcode       13580 non-null  float64
 10  Bedroom2       13580 non-null  float64
 11  Bathroom       13580 non-null  float64
 12  Car            13580 non-null  float64
 13  Landsize       13580 non-null  float64
 14  BuildingArea   13580 non-null  float64
 15  YearBuilt      13580 non-null  float64
 16  Lattitude      13580 non-null  float64
 17  Longtitude     13580 non-null  float64
 18  Regionname     13580 non-null  object
 19  Propertycount  13580 non-null  float64
dtypes: datetime64[ns](1), float64(12), int64(1), object(6)
memory usage: 2.1+ MB
```

```
df.describe()#displaying description of the data in the DataFrame
```

	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	BuildingAr
count	13580.000000	1.358000e+04	13580.000000	13580.000000	13580.000000	13580.000000	13580.000000	13580.000000	13580.0000
mean	2.937997	1.075684e+06	10.137776	3105.301915	2.914728	1.534242	1.610716	558.416127	152.4892
std	0.955748	6.393107e+05	5.868725	90.676964	0.965921	0.691712	0.960511	3990.669241	392.9573
min	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.000000	0.000000	0.000000	0.0000
25%	2.000000	6.500000e+05	6.100000	3044.000000	2.000000	1.000000	1.000000	177.000000	96.7500
50%	3.000000	9.030000e+05	9.200000	3084.000000	3.000000	1.000000	2.000000	440.000000	159.0000
75%	3.000000	1.330000e+06	13.000000	3148.000000	3.000000	2.000000	2.000000	651.000000	176.8662

```
df.nunique()
```

```
Suburb             314
Address          13378
Rooms                9
Type                 3
Price             2204
Method               5
SellerG            268
Date                58
Distance           202
Postcode           198
Bedroom2            12
Bathroom             9
Car                 13
Landsize          1448
BuildingArea       605
YearBuilt          147
Lattitude         6503
Longtitude        7063
Regionname           8
Propertycount      311
dtype: int64
```

Encoding

> Converting the object or human readable values to numeric or machine readable form.

```
df['Method'].value_counts()#displaying the number of unique values under each label of the attribute
```

```
S     9022
SP    1703
PI    1564
VB    1199
SA      92
Name: Method, dtype: int64
```

```
df.replace({'Method':{'S':1,'SP':2,'PI':2,'VB':2,'SA':2}},inplace=True)#grouping
```

```
df['Method'].value_counts()#one can observe the change in label and value count
```

```
1    9022
2    4558
Name: Method, dtype: int64
```

```
df['Type'].value_counts()
```

```
h    9449
u    3017
t    1114
Name: Type, dtype: int64
```

```
df.replace({'Type':{'h':1,'u':2,'t':2}},inplace=True)
```

```
df['Type'].value_counts()
```

```
1    9449
2    4131
Name: Type, dtype: int64
```

```
df['Regionname'].value_counts()
```

```
Southern Metropolitan         4695
Northern Metropolitan         3890
Western Metropolitan          2948
Eastern Metropolitan          1471
South-Eastern Metropolitan     450
Eastern Victoria                53
Northern Victoria               41
Western Victoria                32
Name: Regionname, dtype: int64
```

```
df.replace({'Regionname':{'Southern Metropolitan':1,'Northern Metropolitan':2,'Western Metropolitan':3,'Eastern Metropolitan':3,'Sout
```

```
df['Regionname'].value_counts()
```

```
3    4995
1    4695
2    3890
Name: Regionname, dtype: int64
```

Explaining why these Attributes are droped while defining our dependent and independent variable

```
df['SellerG'].value_counts()
```

```
Nelson           1565
Jellis           1316
hockingstuart    1167
```

```
Barry             1011
Ray                701
                  ...
Prowse               1
Luxe                 1
Zahn                 1
Homes                1
Point                1
Name: SellerG, Length: 268, dtype: int64
```

```
df['Suburb'].value_counts()
```

```
Reservoir          359
Richmond           260
Bentleigh East     249
Preston            239
Brunswick          222
                  ...
Sandhurst            1
Bullengarook         1
Croydon South        1
Montrose             1
Monbulk              1
Name: Suburb, Length: 314, dtype: int64
```

```
df['Address'].value_counts()
```

```
36 Aberfeldie St    3
2 Bruce St          3
5 Charles St        3
53 William St       3
14 Arthur St        3
                   ..
16 Alleford St      1
2/1073 Centre Rd    1
14 Columbia St      1
21 Hardy Ct         1
6 Agnes St          1
Name: Address, Length: 13378, dtype: int64
```

Explaination:

> Clearly, the lenght or the number of Categories of these Attributes is large and nearly impossible to Encode.

Representating Encoded dataset and information

```
df
```

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	Postcode	Bedroom2	Bathroom	Car	Lands
0	Abbotsford	85 Turner St	2	1	1480000.0	1	Biggin	2016-03-12	2.5	3067.0	2.0	1.0	1.0	2(
1	Abbotsford	25 Bloomburg St	2	1	1035000.0	1	Biggin	2016-04-02	2.5	3067.0	2.0	1.0	0.0	1!
2	Abbotsford	5 Charles St	3	1	1465000.0	2	Biggin	2017-04-03	2.5	3067.0	3.0	2.0	0.0	1:
		40						2017-						

```
df.describe()
```

	Rooms	Type	Price	Method	Distance	Postcode	Bedroom2	Bathroom	Car
count	13580.000000	13580.000000	1.358000e+04	13580.000000	13580.000000	13580.000000	13580.000000	13580.000000	13580.00000(
mean	2.937997	1.304197	1.075684e+06	1.335641	10.137776	3105.301915	2.914728	1.534242	1.61071(
std	0.955748	0.460084	6.393107e+05	0.472231	5.868725	90.676964	0.965921	0.691712	0.96051
min	1.000000	1.000000	8.500000e+04	1.000000	0.000000	3000.000000	0.000000	0.000000	0.00000(
25%	2.000000	1.000000	6.500000e+05	1.000000	6.100000	3044.000000	2.000000	1.000000	1.00000(
50%	3.000000	1.000000	9.030000e+05	1.000000	9.200000	3084.000000	3.000000	1.000000	2.00000(
75%	3.000000	2.000000	1.330000e+06	2.000000	13.000000	3148.000000	3.000000	2.000000	2.00000(
max	10.000000	2.000000	9.000000e+06	2.000000	48.100000	3977.000000	20.000000	8.000000	10.00000(

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 13580 entries, 0 to 13579
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Suburb         13580 non-null  object
 1   Address        13580 non-null  object
 2   Rooms          13580 non-null  int64
 3   Type           13580 non-null  int64
 4   Price          13580 non-null  float64
 5   Method         13580 non-null  int64
 6   SellerG        13580 non-null  object
 7   Date           13580 non-null  datetime64[ns]
 8   Distance       13580 non-null  float64
 9   Postcode       13580 non-null  float64
 10  Bedroom2       13580 non-null  float64
 11  Bathroom       13580 non-null  float64
 12  Car            13580 non-null  float64
 13  Landsize       13580 non-null  float64
 14  BuildingArea   13580 non-null  float64
 15  YearBuilt      13580 non-null  float64
 16  Lattitude      13580 non-null  float64
 17  Longtitude     13580 non-null  float64
 18  Regionname     13580 non-null  int64
 19  Propertycount  13580 non-null  float64
dtypes: datetime64[ns](1), float64(12), int64(4), object(3)
memory usage: 2.1+ MB
```

```
df.nunique()
```

```
Suburb             314
Address          13378
Rooms                9
Type                 2
Price             2204
Method               2
SellerG            268
Date                58
Distance           202
Postcode           198
Bedroom2            12
```

```
Bathroom            9
Car                13
Landsize         1448
BuildingArea      605
YearBuilt         147
Lattitude        6503
Longtitude       7063
Regionname          3
Propertycount     311
dtype: int64
```

```
df.groupby(['Type','Method']).count()#grouping categorical variables for feature selection
```

Type	Method	Suburb	Address	Rooms	Price	SellerG	Date	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	BuildingArea
1	1	6507	6507	6507	6507	6507	6507	6507	6507	6507	6507	6507	6507	6507
	2	2942	2942	2942	2942	2942	2942	2942	2942	2942	2942	2942	2942	2942
2	1	2515	2515	2515	2515	2515	2515	2515	2515	2515	2515	2515	2515	2515
	2	1616	1616	1616	1616	1616	1616	1616	1616	1616	1616	1616	1616	1616

Data Visualisation

```
sns.pairplot(df)#visualisation study
```

Part 1: Regression Analysis

Step 1: Define X and y

```
y=df['Price']#'Price' is our Regression target
np.log(y)#for increasing efficiency
```

```
0        14.207553
1        13.849912
2        14.197366
3        13.652992
4        14.285514
           ...
13575    14.034646
13576    13.846040
13577    13.972514
13578    14.731801
13579    14.066269
Name: Price, Length: 13580, dtype: float64
```

```
y.shape
```

```
(13580,)
```

```
X=df.drop(['Suburb','Address','SellerG','Date', 'Price'],axis=1)#defining X
```

```
X.shape
```

```
(13580, 15)
```

Dealing with Oversampling Data

```
'Price'
```

```
from imblearn.over_sampling import RandomOverSampler
```

```
r=RandomOverSampler(random_state=2408)
```

Before Oversampling

```
X.shape, y.shape
```

```
((13580, 15), (13580,))
```

```
y.value_counts()
```

```
1100000.0    113
1300000.0    109
650000.0     109
800000.0     109
600000.0     104
            ... 
1928000.0      1
2236000.0      1
601500.0       1
550500.0       1
1323000.0      1
Name: Price, Length: 2204, dtype: int64
```

```
X.value_counts()
```

```
Rooms  Type  Method  Distance  Postcode  Bedroom2  Bathroom  Car  Landsize  BuildingArea  YearBuilt    Lattitude  Longtitude  R
2      2     1       3.3       3141.0    2.0       2.0       2.0  17200.0   80.737121     2000.000000  -37.83610  144.99660   1
1      2     1       3.3       3141.0    1.0       1.0       1.0  0.0       80.737121     2000.000000  -37.83610  144.99660   1
3      1     1       7.5       3123.0    3.0       2.0       2.0  431.0     120.000000    1950.000000  -37.82690  145.04960   1
             2       9.2       3146.0    3.0       2.0       1.0  704.0     134.000000    1940.000000  -37.85200  145.09420   1
```

```
2    2    1    6.5     3071.0    2.0     1.0    1.0  0.0     80.737121    1970.000000  -37.75610  145.00670   2

3    1    1    3.8     3207.0    3.0     1.0    0.0  153.0   109.000000   1880.000000  -37.83820  144.94690   1
                                                     171.0   167.000000   1890.000000  -37.83850  144.94090   1
                                                     184.0   100.000000   1905.000000  -37.83480  144.94430   1
                                                     197.0   176.866248   1954.081176  -37.83430  144.93620   1
10   1    2    12.1    3083.0    10.0    3.0    2.0  313.0   176.866248   2006.000000  -37.71098  145.05381   2
Length: 13513, dtype: int64
```

After Oversampling

```
X, y = r.fit_resample(X,y)
```

```
X.shape, y.shape
```

```
((249052, 15), (249052,))
```

```
y.value_counts()
```

```
1480000.0    113
978500.0     113
920500.0     113
2633000.0    113
760500.0     113
            ...
2220000.0    113
667000.0     113
2105000.0    113
2177000.0    113
1323000.0    113
Name: Price, Length: 2204, dtype: int64
```

```
X.value_counts()
```

Rooms	Type	Method	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	BuildingArea	YearBuilt	Lattitude	Longtitude	R
2	2	1	3.3	3141.0	2.0	2.0	2.0	17200.0	80.737121	2000.000000	-37.83610	144.99660	1
1	2	2	11.8	3204.0	1.0	1.0	1.0	0.0	80.737121	1980.016708	-37.90320	145.04550	1
2	2	2	9.1	3015.0	2.0	1.0	1.0	0.0	80.737121	1970.000000	-37.82880	144.87110	3
3	1	2	7.0	3013.0	3.0	1.0	1.0	248.0	176.866248	1954.081176	-37.81570	144.89250	3
4	1	2	7.7	3015.0	4.0	2.0	0.0	389.0	158.000000	1990.000000	-37.82840	144.88610	3
3	1	1	12.6	3020.0	3.0	1.0	1.0	700.0	145.000000	1960.000000	-37.79330	144.84110	3
4	1	1	15.5	3038.0	4.0	2.0	2.0	713.0	164.000000	1982.000000	-37.72305	144.81074	3
3	1	1	12.6	3020.0	3.0	1.0	1.0	500.0	176.866248	1954.081176	-37.78210	144.84560	3
								0.0	126.000000	1950.000000	-37.77960	144.84550	3
10	1	2	12.1	3083.0	10.0	3.0	2.0	313.0	176.866248	2006.000000	-37.71098	145.05381	2

Length: 13513, dtype: int64

Step 2: Splitting the data

```
from sklearn.model_selection import train_test_split
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=2529)#since test size is given as 30%, train size is 70
```

```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

```
((174336, 15), (74716, 15), (174336,), (74716,))
```

Standard Scaling the dataset

```
from sklearn.preprocessing import StandardScaler
```

```
s=StandardScaler()
```

```
X_train_s=s.fit_transform(X_train)#Scaling train data
```

```
X_test_s=s.fit_transform(X_test)#Scaling test data
```

Visualisation and Impact of Scaling

> Reduced impact of outlier

```
X_train_s=pd.DataFrame(X_train_s,columns=X_train.columns)
X_test_s=pd.DataFrame(X_test_s,columns=X_test.columns)
```

```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 5))
ax1.scatter(X_train['YearBuilt'], X_train['Landsize'],color='green')
ax1.set_title("Before Scaling")
ax2.scatter (X_train_s['YearBuilt'], X_train_s['Landsize'],color="red")
ax2.set_title("After Scaling")
plt.show()
```

> Trying this code on different attributes will help you understand the impact of scaling

Step 3: Creating Model

```
#from sklearn.linear_model import LinearRegression
#model=LinearRegression()#Linear Regression Model
```

```
#from sklearn.neighbors import KNeighborsRegressor
```

```
#model=KNeighborsRegressor()#K-Nearest Neighbour Model
```

```
#from sklearn.tree import DecisionTreeRegressor
#model=DecisionTreeRegressor()#Decision tree Model
```

```
from sklearn.ensemble import RandomForestRegressor
model=RandomForestRegressor()#Random Forest Model
```

Step 4: Training Model

```
model.fit(X_train_s,y_train)#model that is used in the previous step is being trained in this step
```

```
RandomForestRegressor()
```

Step 5: Predicting Model

```
y_pred=model.predict(X_test_s)#Predicting the target for the given data
```

```
y_pred
```

```
array([1.93, 2.  , 2.  , ..., 2.  , 1.  , 1.  ])
```

Step 6: Accuracy

```
from sklearn.metrics import mean_absolute_percentage_error,mean_absolute_error,r2_score#checking for the accuracy of the prediction m
```

```
mean_absolute_percentage_error(y_test,y_pred)
```

```
0.05907422347720855
```

```
mean_absolute_error(y_test,y_pred)
```

```
0.07926986688180719
```

```
r2_score(y_test,y_pred)
```

```
0.8529766602228644
```

Sample Future Predictions Example

```
df_new=df.sample(1)#taking a sample set from the dataset
```

```
df_new#displaying sample set
```

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	Bu
8245	Prahran	303/10 Hillingdon Pl	2	2	645000.0	1	Gary	2017-05-13	4.5	3181.0	2.0	1.0	1.0	2842.0	

```
X_new=df_new.drop(['Suburb','Address','SellerG','Date','Price'],axis=1)#defining X or independent variables of samole set
```

```
X_new.shape
```

```
(1, 15)
```

```
y_pred_new=model.predict(X_new)#sample prediction
```

```
y_pred_new
```

```
array([2377000.])
```

Part 2: Classification Analysis

Step 1: Define X and y

```
y=df['Type']#'Type' is our Classification Target
```

```
y.shape
```

```
(13580,)
```

```
X=df.drop(['Suburb','Address','SellerG','Date', 'Type'],axis=1)
```

```
X.shape
```

```
(13580, 15)
```

Dealing with Undersampling Data

```
'Type'
```

```
from imblearn.under_sampling import RandomUnderSampler
```

```
r=RandomUnderSampler(random_state=2408)
```

```
X, y = r.fit_resample(X,y)
```

```
X.shape, y.shape
```

```
((8262, 15), (8262,))
```

```
y.value_counts()
```

```
1    4131
2    4131
Name: Type, dtype: int64
```

```
X.value_counts()
```

```
Rooms  Price      Method  Distance  Postcode  Bedroom2  Bathroom  Car  Landsize  BuildingArea  YearBuilt    Lattitude  Longtitu
2      965000.0   1       7.3       3146.0    2.0       2.0       1.0  704.0     140.046323    1998.988189  -37.85698  145.0468
       890500.0   2       2.4       3121.0    2.0       2.0       2.0  180.0     123.000000    1940.000000  -37.81443  144.9907
       435000.0   2       7.8       3124.0    2.0       1.0       1.0  896.0     77.000000     1960.000000  -37.84790  145.0958
4      1817000.0  2       4.2       3031.0    4.0       2.0       1.0  309.0     176.866248    1954.081176  -37.79100  144.9280
3      720000.0   2       7.8       3058.0    3.0       2.0       2.0  531.0     112.000000    2016.000000  -37.74240  144.9571

2      802000.0   1       5.4       3101.0    2.0       1.0       1.0  0.0       80.737121     1980.016708  -37.80456  145.0374
       801250.0   1       11.0      3018.0    2.0       1.0       1.0  245.0     80.737121     1980.016708  -37.87046  144.8348
       801000.0   1       14.5      3188.0    2.0       1.0       1.0  217.0     107.000000    1994.000000  -37.94320  145.0348
                          6.9       3039.0    2.0       1.0       0.0  133.0     176.866248    1954.081176  -37.77120  144.9254
8      2950000.0  1       11.0      3147.0    9.0       7.0       4.0  1472.0    618.000000    2009.000000  -37.87290  145.0788
Length: 8251, dtype: int64
```

Step 2: Splitting the data

```
from sklearn.model_selection import train_test_split
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,train_size=0.7,stratify=y,random_state=2408)
```

```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

```
((5783, 15), (2479, 15), (5783,), (2479,))
```

Standard Scaling the dataset

```
df.describe()
```

	Rooms	Type	Price	Method	Distance	Postcode	Bedroom2	Bathroom	Car
count	13580.000000	13580.000000	1.358000e+04	13580.000000	13580.000000	13580.000000	13580.000000	13580.000000	13580.00000(
50%	3.000000	1.000000	9.030000e+05	1.000000	9.200000	3084.000000	3.000000	1.000000	2.00000(

```
from sklearn.preprocessing import StandardScaler
```

```
s=StandardScaler()
```

```
X_train_s=s.fit_transform(X_train)
```

```
X_test_s=s.fit_transform(X_test)
```

Visualisation and Impact of Scaling

> Reduced impact of outlier

```
X_train_s=pd.DataFrame(X_train_s,columns=X_train.columns)
X_test_s=pd.DataFrame(X_test_s,columns=X_test.columns)
```

```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 5))

ax1.scatter(X_train['YearBuilt'], X_train['Price'],color='black')
ax1.set_title("Before Scaling")
ax2.scatter (X_train_s['YearBuilt'], X_train_s['Price'],color="red")
ax2.set_title("After Scaling")
plt.show()
```

Step 3: Creating Model

```
#from sklearn.linear_model import LogisticRegression
#model=LogisticRegression()#Logistic Regression Model


#from sklearn.neighbors import KNeighborsClassifier
#model=KNeighborsClassifier()#K-Nearest Neighbour Model


#from sklearn.tree import DecisionTreeClassifier
#model=DecisionTreeClassifier()#Decision Tree Model


#from sklearn.svm import SVC
#model=SVC() #Support Vector Machine


from sklearn.ensemble import RandomForestClassifier
model=RandomForestClassifier()
```

Step 4: Training Model

```
model.fit(X_train_s,y_train)
```

```
RandomForestClassifier()
```

Step 5: Predicting Model

```
y_pred=model.predict(X_test_s)
```

```
y_pred
```

```
array([2, 2, 2, ..., 2, 1, 1])
```

Step 6: Accuracy

```
from sklearn.metrics import confusion_matrix, classification_report
```

```
confusion_matrix(y_test,y_pred)#confusion matrix is the best way to represent the accuracy of a Classification Model. It represents T
```

```
array([[1178,   61],
       [  37, 1203]])
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           1       0.97      0.95      0.96      1239
           2       0.95      0.97      0.96      1240

    accuracy                           0.96      2479
   macro avg       0.96      0.96      0.96      2479
weighted avg       0.96      0.96      0.96      2479
```

Future Predictions Example

```
df_new=df.sample(1)#sample set
```

```
df_new
```

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	Postcode	Bedroom2	Bathroom	Car	Landsi
11596	Bentleigh	12 Wood St	5	1	1500000.0	2	hockingstuart	2017-07-22	11.4	3204.0	5.0	2.0	2.0	591

```
X_new=df_new.drop(['Suburb','Address','Type','SellerG','Date'],axis=1)
```

```
X_new.shape
```

```
(1, 15)
```

```
y_pred_new=model.predict(X_new)#sample prediction.
```

```
y_pred_new
```

```
array([1])
```

Link of the Colab file:

https://colab.research.google.com/drive/1nZuh04PYrE9kxCzCDbyYsH-b3czkKgpD?usp=sharing