DSC 478 Programming in Machine Learning Applications

PROJECT

Project Topic: House Price Prediction with Data Analysis

Project Type: Solo

Project Guide: Professor Bamshad Mobasher

Project Date: 10/20/20

Project Done by: Vismay Bhavinkumar Patel

**INDEX**

|  |  |
| --- | --- |
| Project Goal | Page 1 |
| Data Description | Page 2-3 |
| KDD Procedure | Page 4-5 |
| Method | Page 6-7 |
| Tools Used | Page 8-9 |
| Results | Page 10-19 |
| Conclusion | Page 20 |
| Appendix | Page 21-42 |

**Section 1. Project Goal**

The main objective of the project is to perform and execute one Supervised Machine Learning algorithm, and one Unsupervised Machine Learning algorithm.

Supervised machine learning algorithm used in the project was Regression.

Unsupervised machine learning algorithm used in the project was Clustering.

By using the two machine learning techniques, the goal was to learn, understand and implement different libraries and packages on a dataset.

For Example: performing prediction on medical data, based on which season the people visit the hospital/clinic the most. Building recommender systems, which recommend the user, the type of choice of clothes they prefer or recommendation from a movie genre.

I have selected a prediction dataset, with the help of which, I would perform analysis to determine the factors or the best parameters which influence a specific characteristic or a variable in a dataset.

Melbourne Housing Market Dataset is one such dataset, where supervised and unsupervised machine learning techniques can be performed to find, the factors that influence the Price or the houses in Melbourne Housing Market.

On the other hand, visualizing the relation of variable with target attribute can help us understand what can be the factors that are responsible for providing better understanding of the house price rates.

Following are the primary objectives of the Project:

1. Finding the relation of attributes with Target variable by visualization
2. Generating a regression model which will help determine, the factors that best fit to predict house price, and which model gives the best accuracy / least RMSE, MAE values.
3. Performing clustering to see how well the clusters interpretation and validation of consistency with clusters of that data.

**Section 2. Dataset Description:**

The data set was taken from Kaggle, which is an opensource platform where numerous datasets are available, to explore and perform projects or analysis on.

**Data set name:** Melbourne Housing Market.

**Original Source:** This data was scraped from publicly available results posted every week from Domain.com.au

**Data Set Link:**

<https://www.kaggle.com/anthonypino/melbourne-housing-market?select=Melbourne_housing_FULL.csv>

The data set consists of two csv files. FULL and LESS, the FULL dataset consists of 21 columns and LESS dataset consists of 13 columns.

For the analysis in the project I have used FULL dataset, because there are more variables to look at, and sometimes they help in getting better insight about the data.

**Names of the Columns / Attributes:**

*Price:* Price in Australian dollars

*Rooms:* Number of Rooms

*Bedroom2:* Number of bedrooms

*CouncilArea*: Governing Council for the area

*Regionname:* General Region (West, North West, North, North East)

*YearBuilt:* Year the house was built

*Landsize*: Land Size in meters

*Longitude*: Geographic location

*Latitude*: Geographic location

*Type*: Here type is the type of house,

Br – bedroom(s), h = house, cottage, villa, semi, terrace;

u = unit, duplex, t = townhouse, dev; site = development site;

o res = other residential.

*Method*: Methods determine the way houses were sold.

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

*SellerG:* Real Estate Agent

*Date:* Date sold

*Distance*: Distance from CBD in Kilometres

*Postcode:* Post code

*Bathroom*: Number of bathrooms

*Car:* number of car spots

*BuildingArea*: Building Size in Meters

*Propertycount*: Number of properties that exists in the suburb

*Suburb*: Suburb

*Address*: Address

**Number of rows in dataset:**  There are 34857 rows in the entire dataset.

**Data Types:**

There are three data types in the dataset, i.e. integer, float and object. Following are the number of data types present in the dataset.

Int64 = 1, Float64 = 12, object = 8

Amongst all these data types some data types need to be changed due to appropriateness, following are the attributes:

Postcode, Bedroom2, Bathroom, Car, YearBuilt, Propertycount are the attributes which were changed from float to integer.

After changing the data types the total data types were as follows: int64 = 7, float64 = 6,

object = 8.

**Section 3. KDD Procedure:**

Let us discuss the Knowledge discovery process performed in this project step my step, based on the KDD hierarchy/structure.

**Selection:**

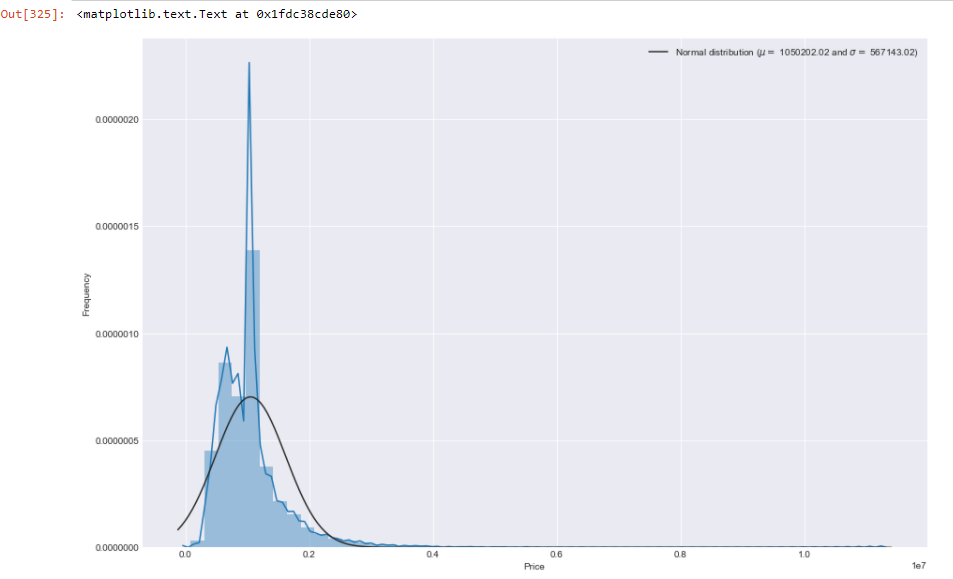
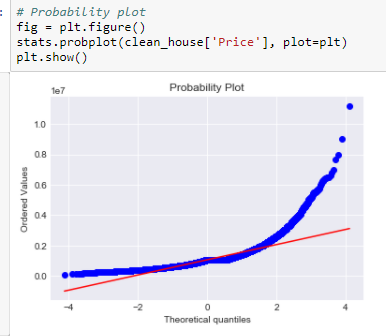
Structured data was selected, containing .csv file format. The data was selected from open source resource. Data with more than 10 Fractures/attributes had to be selected, in order to get sufficient features to explore or perform, pre-processing, Transformation and Data Mining tasks. Also, it is good to have about 1000 + rows, this allows the algorithm and model to train better and chances of getting efficient and precise output is higher. My data set consists of 21 columns and approximately 38450 rows.

**Preprocessing:**

The date set was about housing market prediction, so the most ideal target variable could be to predict the Price of the houses. The reason for this is, whenever and where-ever we observe, the price of the house is fluctuated because of certain parameters, which can bring ups and downs in the market. These features or parameters depend on the customer preference; thus, Price attribute was selected as Target variable. Moreover, initially the data types of some attributes had to be changed, in order to add meaning to the attribute. For example, YearBuilt and Propertycount were float values, YearBuilt and Propertycount do not need to be in float, because Years and Property are never calculated in decimals.

**Transformation:**

The transformation part was understood by the graphical visualizations of the categorical and numerical attributes, like creating bar chart, linear plots help to understand which variable needs to be transformed. The reason is because Price attribute has a very high value and also, it is because it is right skewed. Hence, while performing regression the value would be log transformed and normalized.

**Data Mining:**

For data mining, Regression and clustering methods were selected, regression was done by selecting features from correlation plot, and extracted features with positive correlation or moderately positive correlation between the attributes. Regression was performed by using different models like, Linear regression, Lasso Regression, ElasticNetRegressor, RandomForestRegressor, knnneighbourRegressor, GradientBoostingRegressor, AdaBoostRegressor, DecisionTreeRegresso

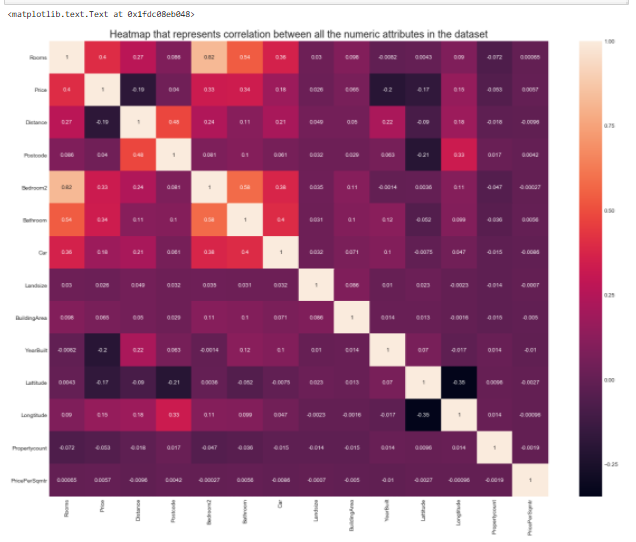
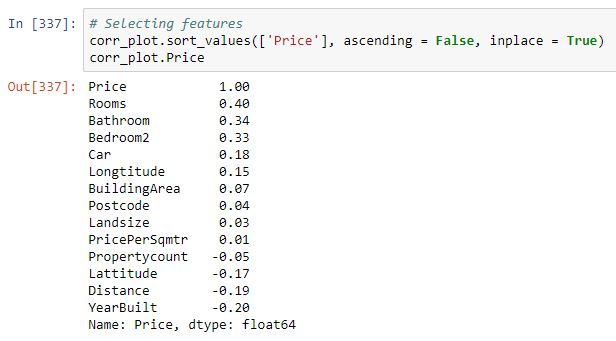
r. 

Figure 1.0

Feature selection done by corr\_plot.

Each time the model was run, the attributes and target variable were either normalized or they were not normalized. This was done to see if the model with normalized attributes and target variable gives good results or the model with non - normalized data gives good results.

After regression KMeans Clustering was done, using Silhouette analysis to evaluate the method interpretation and validation of consistency of the clusters in the data. The analysis was performed two times with different n number of clusters.

**Interpretation:**

The interpretation from Regression analysis was that, normalized attributes and target variables yield best results and for clustering, the clusters with more n values give higher homogeneity and completeness values than that of cluster with n value 5.

**Section 4. Methods:**

Method 1: Regression Analysis

The Regression was done on 9 different models:

LinearRegression with default parameters.

Lasso with default parameters

Ridge with default parameters

ElasticNet with default parameters

KNeighborsRegressor with default parameters

RandomForestRegressor with default parameters

GradientBoostingregressor with default parameters

AdaBoostregressor with n\_estimators = 10

DecisionTreeRegressor with max\_depth = 5, min\_samples\_split = 4, random\_state =1

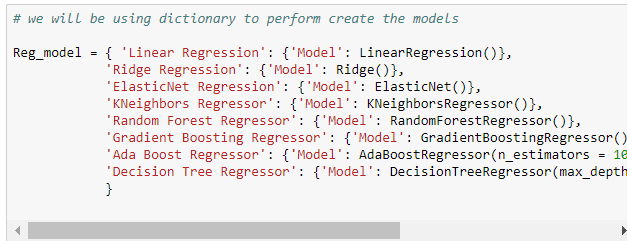


Figure 1.2

I used dictionaries to create the models and assign the keys and values later so that the operation could be much faster.



Figure 1.3

Method 2: KMeans Clustering using Silhouettes plot

The method was performed with two sets of n\_clusters, first time n\_clusters were set to 5, with 500 maximum iterations and verbose = 1.

Second time the n\_clusters were set to 8 with same max iteration 500 and verbose = 1

Here Verbose means, Verbose means that it will output messages which could be useful for debugging and for understanding how the training is doing.

The inertia is the sum of the squared distance for each point to its closest centroid, i.e., its assigned cluster.

**Section 5: Tools used**

There were different tools used to perform visualization, descriptive statistics, Regression and cluster analysis;

Descriptive Statistics and visualization:

Basic pandas library and mathematical functions were used to derive insights from the data. For example **.mean(),** **.describe()** were used to get the average mean of the numeric variables / attributes and to display the descriptive statistics, i.e. minimum, maximum values, standard deviation, quantile range (25%, 50%, 75%), number of unique values, etc.

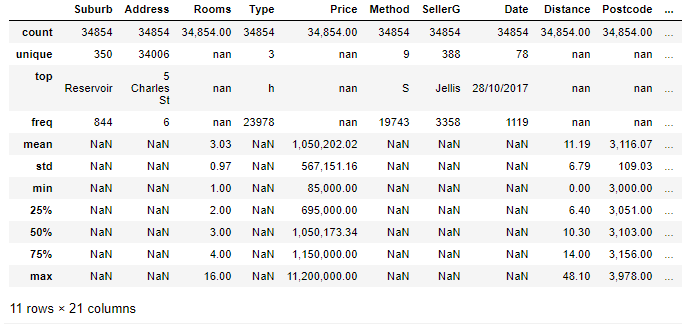
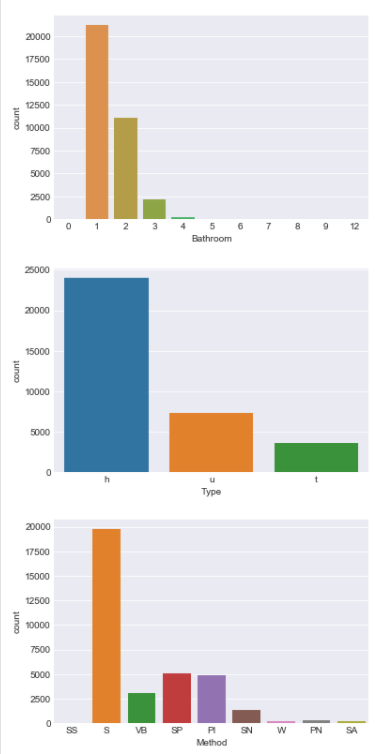
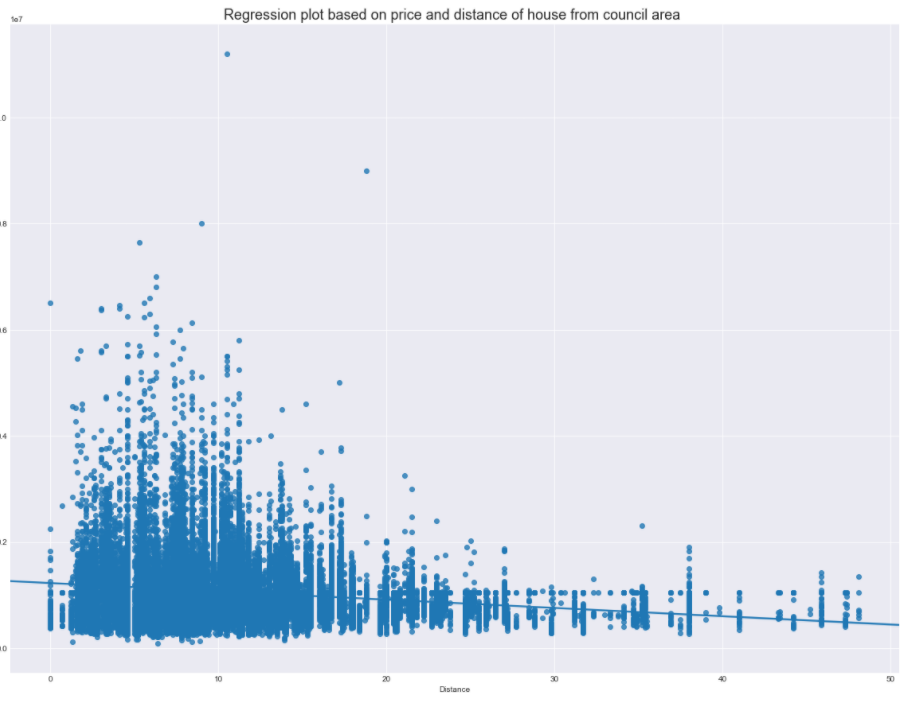


Figure 1.4

Seaborn library was used additionally to generate the descriptive statistic plots. From **seaborn** library. regplot**(),** **.countplot()** methods were used to display regression plots between Price and other attributes and display basic count plots of attributes.

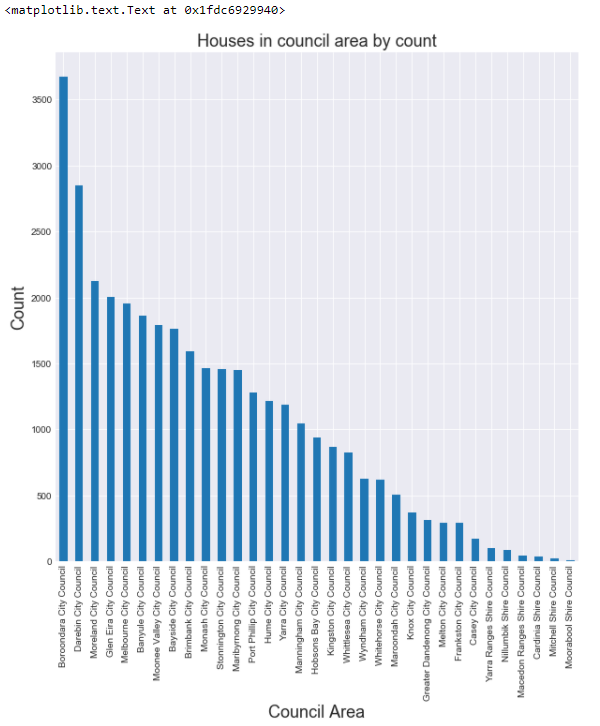


Figure 1.5

To display the normalized plot and probability plot of Price attribute, **stats** package was used from **scipy** module.

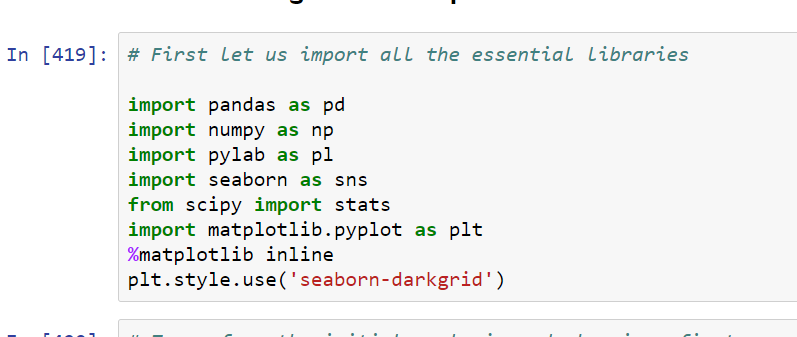


Figure 1.6

Data Analysis and Per-processing:

To convert categorical attributes Label Encoding package was used from sklearn module.

To perform pre-processing, regression and clustering the following tools were used shown in the image below:

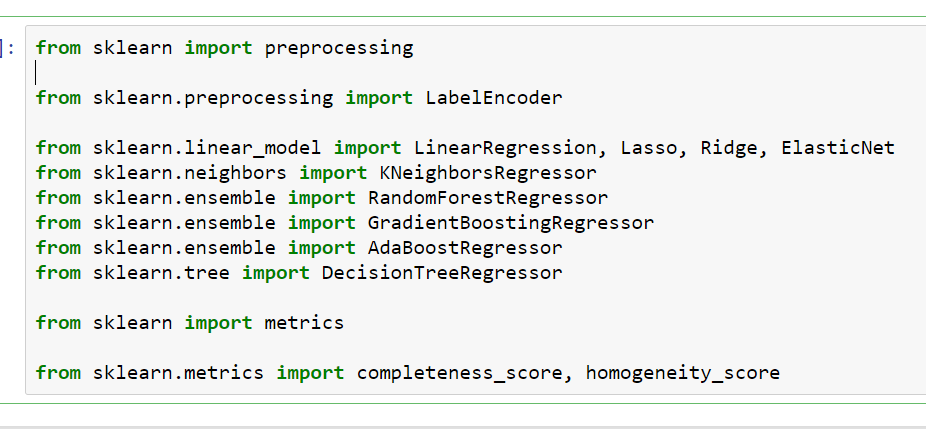


Figure 1.7

**Section 6: Result:**

Results of the analysis:

Regression Analysis:

The Results of Regression analysis are divided in four parts:

Part 1: Non – normalized attributes and target variable:

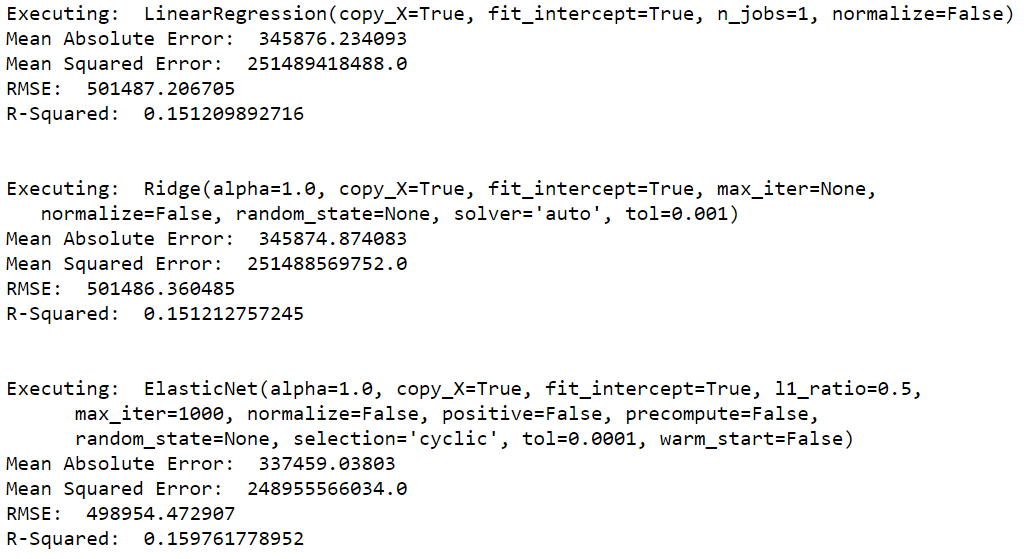


Figure 1.7

Here in the first three techniques we observe high error and low R-Squared value.

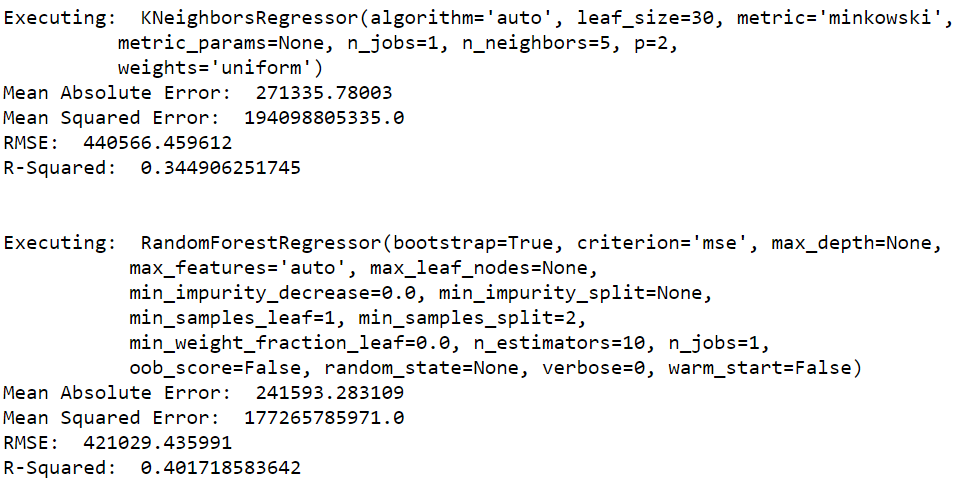


Figure 1.8

KNeighborsRegressor:

The target is predicted by local interpolation of the targets associated of the nearest neighbours in the training set. It is based on K nearest neighbours.

In the next two methods, RandomForestRegressor has done well compared to the first three models, but sill the error is much higher.



Figure 1.9

Gradient boost model builds the model in a stage wise pattern where the loss function is arbitrarily optimized for being differentiable. The negative gradient is fit to the regression tree to give the loss function.

Here, gradient boost gives us the optimization of 44.35 which is the best in all the models.

AdaBoost has performed bad in the part 1 analysis and DecisionTreeRegressor can be termed as average based on the current Part 1 outputs.

Part 2: Normalized data attributes, keeping Target variable as it is. (unchanged)

We obtain the following results:

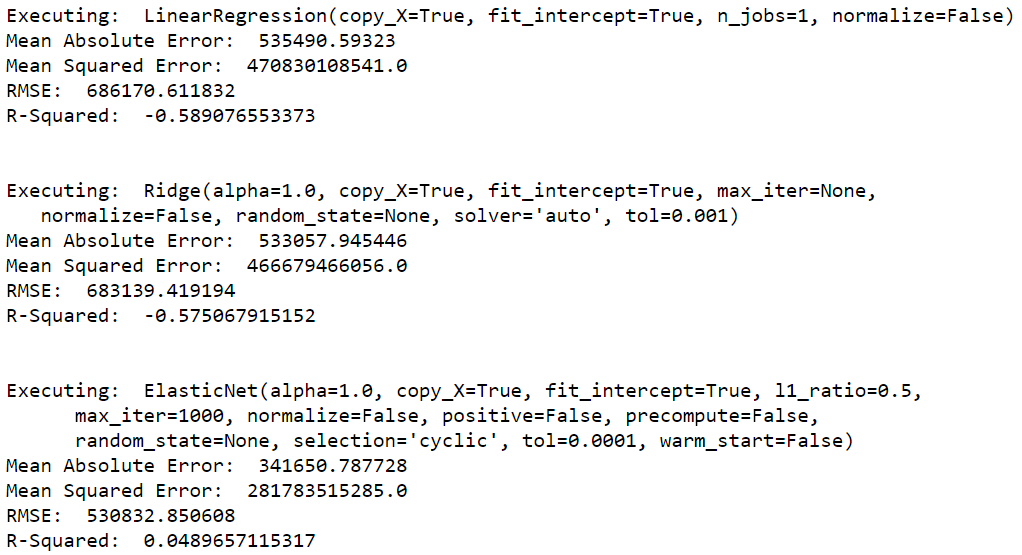


Figure 2.0

In this part none of the models have don’t well we can see the first three outputs

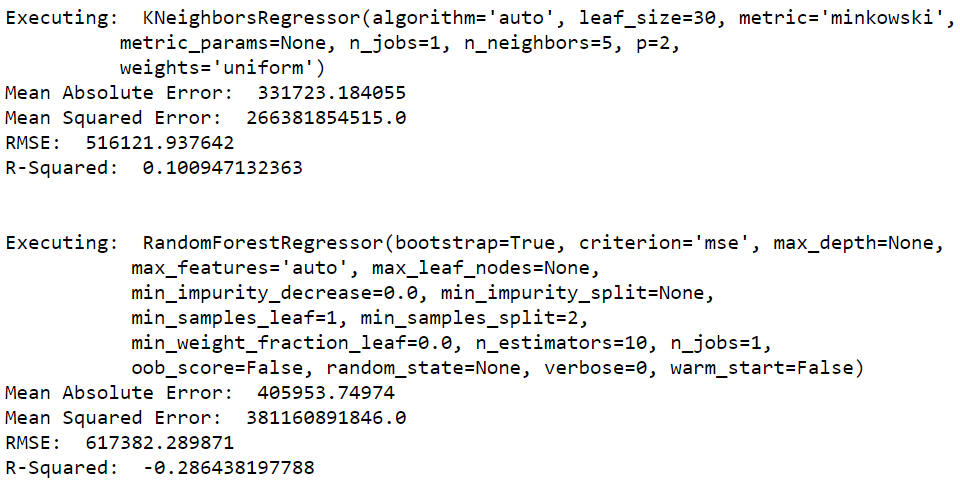


Figure 2.1

Both the models above have also not done well.

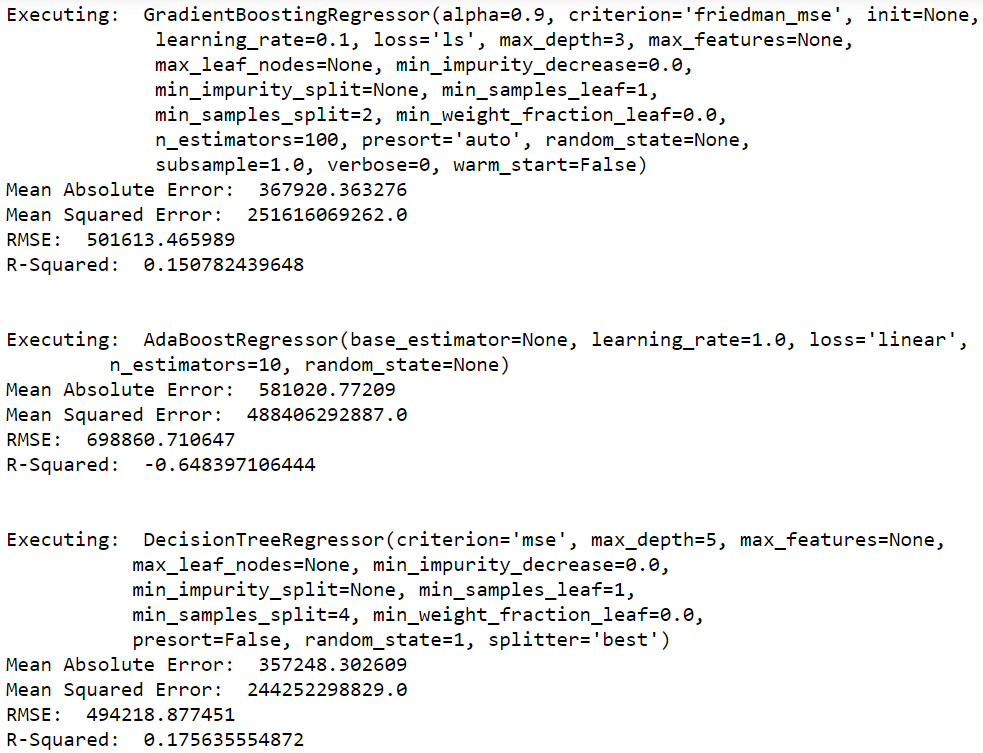


Figure 2.2

Here, GradientBoostingRegressor and Decision Tree Regressor have shown significantly better outputs than other models, though they are not valid or not good to be considered.

Part 3: Normalizing the Target attribute by using log transformation. Using both normalized test and training set to perform analysis

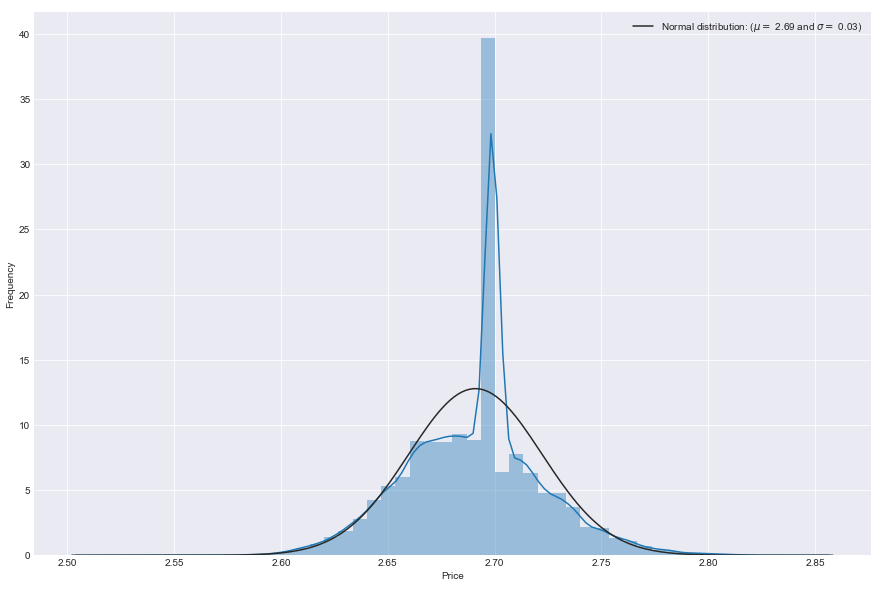


Figure 2.3

Log transform (normalized) Price target attribute.

Probability plot showing normalization line:

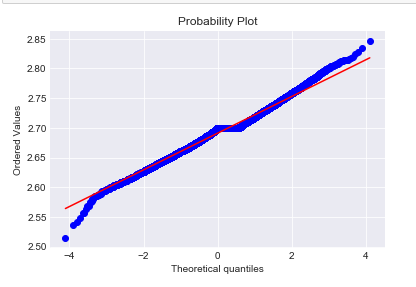


Figure 2.4

Results after making these changes in the modes are as follows:



Figure 2.5

Now, in these models after normalizing the target attribute and training data, we can see following changes.

Linear regression model shows 91% R-squared value, meaning it can define/predict about 91 percent of the target attribute.

Similar result is observed with Ridge regression, except Elastic net which has R-Squared going in negative value.

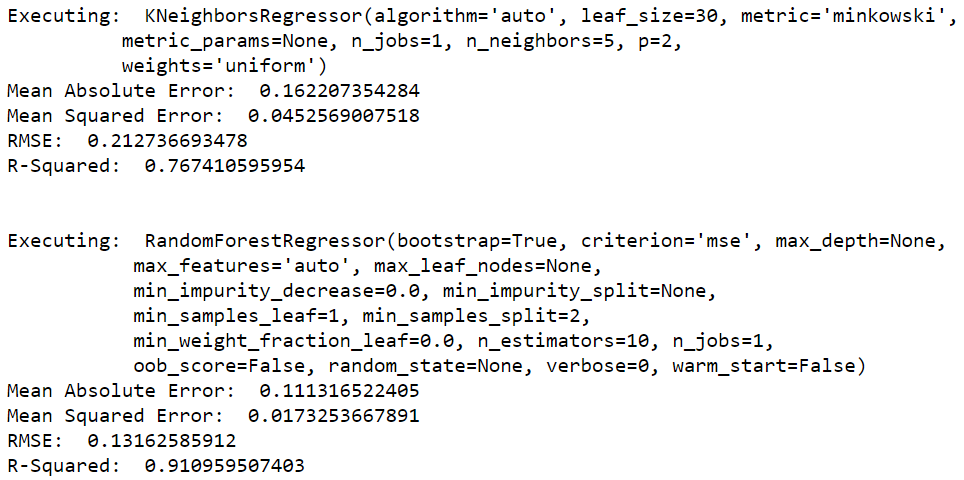


Figure 2.6

RandomForestRegression and KNeighboursRegressor have also done well, despite the fact that, KNeighboursRegressor has less R-squared compared to the three models.



Figure 2.7

GradientBoosting, AdaBoost, DecisionTree regressors have done well, and have almost similar values with the difference of 0.01.

If we review the performance of all the models, we can find out that, RandomForestRegressor, Linear Regression, Ridge Regression have done best. If we have to select any one from these, we can select RandomForestRegressor as the R-squared value is 91.09 % highest in these three models and comparing the error values, they are less in all the models.

Part 4: Reducing the number of attributes and keeping the other input same / unchanged.

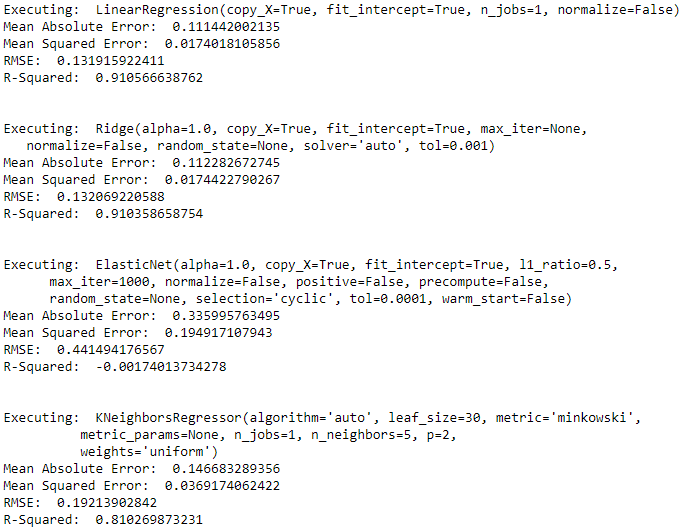


Figure 2.8

By changing the number of parameters / features we feed into the model, there is one change that we can find. KNeighboursRegressor model performance has increased by 4-5 percent. Other models however have similar values.

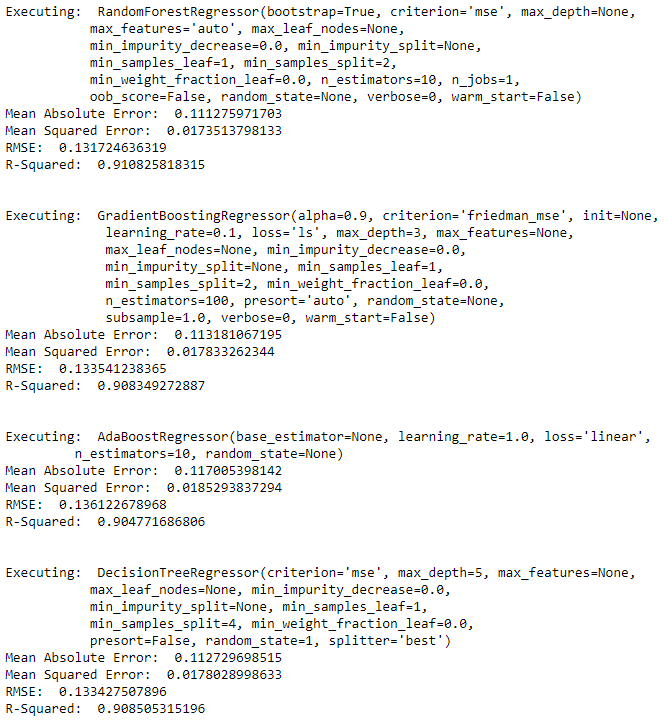


Figure 2.9

The models in this figure are almost similar to that of what we have seen in part 3.

The take away from Complete regression analysis is reducing the number of parameters has improved the performance of KNeighnoursRegressor.

Based on this we can say that Regression Analysis Part 4 with reduced parameters has the best models to be considered.

Cluster analysis with silhouette plot:

For Cluster Analysis I have selected integer attributes from the dataset and considered ‘Type’ attribute as the cluster class.

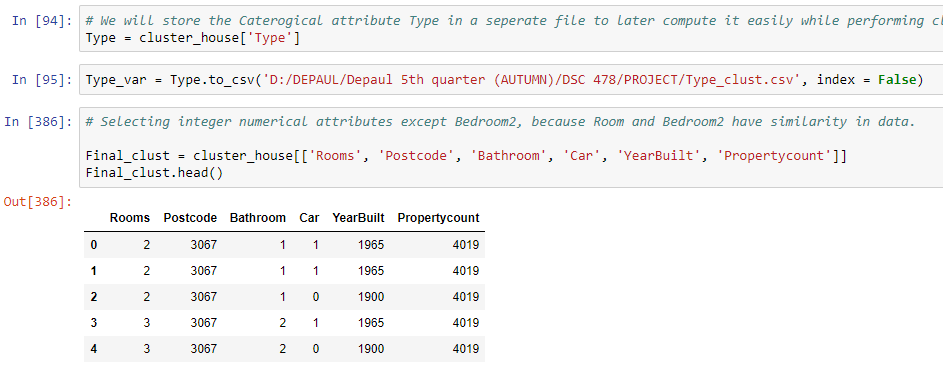
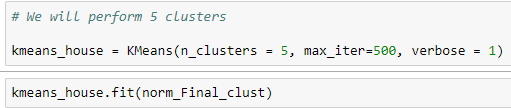


Figure 3.0

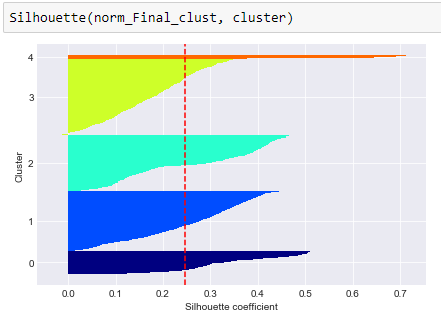
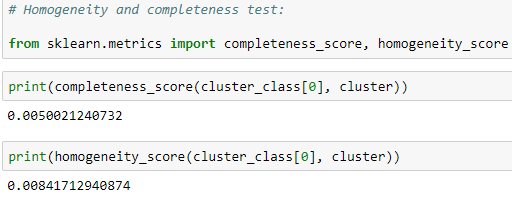
As shown in the figure, I have extracted the Type categorical variable into another csv file, and selected specific columns to perform clustering on the Type variable.

N\_clusters = 5 ,8



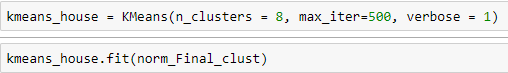
Clustering was performed two times, first by taking 5 clusters and them by takin 8 clusters.

Then homogeneity and completeness of the clusters was calculated for both the clusters including the silhouettes plot.

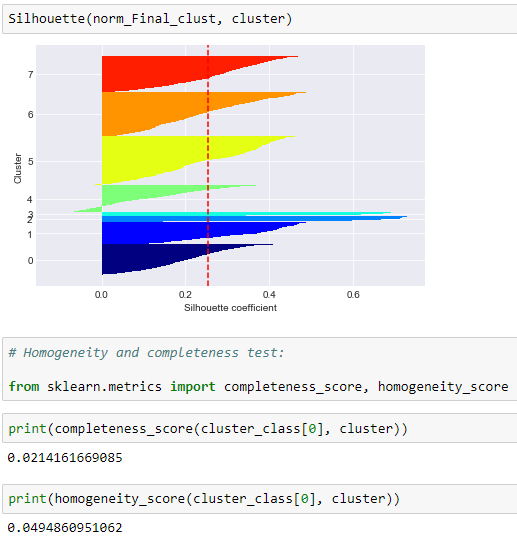
 

The Silhouette plot here gives us 5 clustes, i.e. silhouette coefficients

Cluster for n = 8 clusters:



Silhouette plot for n = clusters



The Silhouette coefficients give us 8 values.

Homogeneity and Completeness are evaluated on the scale of 0.0 to 1.0

So, talkinh about Homogeneity, all the observations having same class labels are within same clusters

Completeness means all the members of same class are in same cluster.

Based on these principles we will evaluate the homogeneity and completeness values that we have got.

Comparing Homogeneity from both n = 5 and n= 8 clustering, we get the following values:

N=5 (0.008417), N=8 (0.049860)

Both the values are at 0.0 and it shows there is homogeneity at the lowest, but if we consider the value up to 3 significant decimal place after decimal, than we can say that most of the clusters fall under homogeneity. Making all the observations have same class labels within same cluster.

**Section 7: Conclusion:**

Conclusion of Data Analysis:

Regression Analysis:

In Regression Analysis 7 models were generated by 4 different normalized and non-normalized training and testing sets.

In the Part 1, Part 2 and Part 3 all the 9 attributes were similar.

Selected attributes [‘Price’, ‘Rooms’, ’Bathroom’, ‘Type’, ‘Method’, ‘Regionname’, ‘CouncilArea’, ‘SellerG’, ‘Suburb’]

Out of which Price was not normalized by log transformation in Part 1 and Part 2.

The rest of the attributes were normalized in Part 2 and Part 3.

The models in Part 1 gave low R-Squared value and high Error rates, in Part 2 the models were having negative value of R-Square value. This was because only Price(target attribute) was not normalized and rest of the data was normalized.

From Part 3, after normalizing the Price by log transformation the models had the R-Squared value ot the prediction rate of 90-91% except elastic regression (negative value) and KNeighbiurRegressor(75% appx), which was good.

To try improvise the model and see if we can get 90-91 % and above in all the models, I removed a few Features and only selected the following featues: [‘Price’, ‘Rooms’, ‘Bathroom’, ‘Type’,’Method’, ‘CouncilArea’]

After making this change and running the model on normalized test and target sets the Prediction percentage of KNeighbourRegressor rised by 4-5 %, bringing the value to 81%. However, elestic net did not show any difference.

We can conclude that Models in Part 4 with attributes [‘Price’, ‘Rooms’, ‘Bathroom’, ‘Type’,’Method’, ‘CouncilArea’] performed the best, and Room, Bathroom, Type, Method, CouncilArea are the best predictors for predicting house prices in Melbourne Housing Market.

Commenting on the Clustering part the clusters were not giving the desired output in homogeneity and completeness, but considering decimals upto 3-4 significant places the clusters tilt towards homogeneity.

\*\*\* Note that Price is the training data so it gets poped when generating the target variable from the dataset. Despite shown in parameters selected features.

\*\*\* Also note that Bedroom2 despite having positive correlation with Price was dropped because both the valeus of room and bedroom2 were almost similar, hence dropped Bedroom2.

**References:** <https://scikit-learn.org/stable/>

**Section 8: Appendix**

**Data Cleaning and Descriptive statistics**

# First let us import all the essential libraries

import pandas as pd

import numpy as np

import pylab as pl

import seaborn as sns

from scipy import stats

import matplotlib.pyplot as plt

%matplotlib inline

plt.style.use('seaborn-darkgrid')

# To perform the initial analysis and cleaning, first we will import the FULL Melbourne Housing dataset.

# We have used na\_values = [?], to replace 'NaN' with the blank or empty values in the dataset.

house\_data = pd.read\_csv('C:/Users/visma/Anaconda3/DSC 478/DSC 478 Project/Melbourne\_housing\_FULL.csv', na\_values=['?'])

house\_data

# The shape of the dataset, consists of 34854 rows and 21 columns

house\_data.shape

**Details of the attributes before cleaning/filling missing values**

# Now we will look at the information function which will provide the details like, total entries in columns, etc.

house\_data.info()

# another way of finding the number of columns that fall into a specific datatype category is:

# house\_data.dtype.value\_counts()

# Anothe way of checking the missing values is as follows:

house\_data[house\_data.isnull().any(axis=1)]

**Removing missing values**

# The first column to be cleaned is Price, dtype = float

# We will replace all the missing values with the mean of the existing values

price\_mean = house\_data.Price.mean()

house\_data.Price.fillna(price\_mean, axis = 0, inplace = True)

# Next column Distance dtype = float

Dist\_mean = house\_data.Distance.mean()

house\_data.Distance.fillna(Dist\_mean, axis = 0, inplace = True)

# Column = Postcode, dtype = float

PC\_mean = house\_data.Postcode.mean()

house\_data.Postcode.fillna(PC\_mean, axis = 0, inplace = True)

# Column Bedroom2, dtype = float

Bedrm2\_mean = house\_data.Bedroom2.mean()

house\_data.Bedroom2.fillna(Bedrm2\_mean, axis = 0, inplace = True)

# Column Bathroom, dtype = float

bath\_mean = house\_data.Bathroom.mean()

house\_data.Bathroom.fillna(bath\_mean,axis = 0, inplace=True)

# Column Bathroom, dtype = float

bath\_mean = house\_data.Bathroom.mean()

house\_data.Bathroom.fillna(bath\_mean,axis = 0, inplace=True)

# Column Car, dtype = float

car\_mean = house\_data.Car.mean()

house\_data.Car.fillna(car\_mean, axis = 0, inplace = True)

# Column Landsize, dtype = float

landsiz\_mean = house\_data.Landsize.mean()

house\_data.Landsize.fillna(landsiz\_mean, axis = 0, inplace = True)

# Column Building Area, dtype = float

bldarea\_mean = house\_data.BuildingArea.mean()

house\_data.BuildingArea.fillna(bldarea\_mean, axis = 0, inplace=True)

# Column YearBuilt, dtype = float

yrbuilt\_mean = house\_data.YearBuilt.mean()

house\_data.YearBuilt.fillna(yrbuilt\_mean, axis = 0, inplace = True)

# Column CouncilArea, dtype = object(categorical variable)

# For categorical variable we delete or drop the missing values

house\_data.drop(house\_data[house\_data.CouncilArea.isnull()].index, axis = 0, inplace = True)

house\_data.drop(house\_data[house\_data.Regionname.isnull()].index, axis = 0, inplace = True)

# Column Latitude, dtype = float

Lat\_mean = house\_data.Lattitude.mean()

house\_data.Lattitude.fillna(Lat\_mean, axis = 0, inplace = True)

Column Longtitude, dtype = float

long\_mean = house\_data.Longtitude.mean()

house\_data.Longtitude.fillna(long\_mean, axis = 0, inplace = True)

PropertyCntmean = house\_data.Propertycount.mean()

house\_data.Propertycount.fillna(PropertyCntmean, axis = 0, inplace = True)

# Check the if the data still contains any missing value

house\_data[house\_data.isnull().any(axis=1)]

house\_data.info()

# changing data type

house\_data['Postcode'] = house\_data['Postcode'].astype('int64')

house\_data['Bedroom2'] = house\_data['Bedroom2'].astype('int64')

house\_data['Bathroom'] = house\_data['Bathroom'].astype('int64')

house\_data['Car'] = house\_data['Car'].astype('int64')

house\_data['YearBuilt'] = house\_data['YearBuilt'].astype('int64')

house\_data['Propertycount'] = house\_data['Propertycount'].astype('int64')

# Loading the data in another file

clean\_house = house\_data.to\_csv('D:/DEPAUL/Depaul 5th quarter (AUTUMN)/DSC 478/PROJECT/int64index\_CLEAN.csv', index = False)

# Declaring a new variable and assigning the clean data set to the variable.

clean\_house = pd.read\_csv('D:/DEPAUL/Depaul 5th quarter (AUTUMN)/DSC 478/PROJECT/int64index\_CLEAN.csv')

clean\_house

# Now the values of the index are in the right shap.

clean\_house.info()

Data type od the attributes

# Now let us find out the names of columns that fall into each datatype category, i.e integer, float, object.

# Column with data type integer.

clean\_house.select\_dtypes(include = ['int64']).head()

# Columns with data type float

clean\_house.select\_dtypes(include = ['float']).head()

# Columns with data type object

clean\_house.select\_dtypes(include = ['object']).head()

Descriptive Statistics

# computing describe.info

clean\_house.describe(include = 'all')

# Plotting two categorical attributes and numerical attributes

data = ['Rooms', 'Bathroom', 'Type', 'Method']

for i in data:

plt.figure()

sns.countplot(x = i, data = clean\_house)

# Plot of house by RegionName

clean\_house['Regionname'].value\_counts().plot(kind='bar', figsize=(10,10))

plt.xlabel('Region Name', size = '18')

plt.ylabel('House in region', size = '18')

plt.title('Houses in the region', size = '18')

# Plot of number of house counts by councial area

clean\_house['CouncilArea'].value\_counts().plot(kind='bar', figsize = (10,10))

plt.xlabel('Council Area', size = '18')

plt.ylabel('Count', size = '18')

plt.title('Houses in council area by count', size = '18')

# We will plot the details of the number of deals done by the real

# estate agents.

Agent = clean\_house['SellerG'].value\_counts()

Agent.head(n = 15).plot(kind = 'bar', figsize = (10,10), title = 'Number of deals done by top real estate agents')

plt.xlabel('SelleG', size = '18')

plt.ylabel('Count', size = '18')

# We will make a new column price per square meter area.

# This column will help us determine the value or price of the house per square meter area.

clean\_house['PricePerSqmtr'] = clean\_house['Price'] // clean\_house['BuildingArea']

clean\_house.info()

pricepersqmtr\_mean = clean\_house.PricePerSqmtr.mean()

clean\_house.PricePerSqmtr.fillna(pricepersqmtr\_mean, axis = 0, inplace = True)

# Plotting the highest price of the suburbs per square meter.

# Price vs Suburb

# Here we have taken the mean of Price per square meter to determine the house rates, based on different areas in suburbs.

sub\_price = clean\_house[['Suburb', 'PricePerSqmtr']].groupby(['Suburb']).agg(['mean'])

sub\_price.columns=['Mean PricePerSqmtr']

sub\_price.head()

sub\_price.nlargest(10,['Mean PricePerSqmtr']).plot(kind='bar', figsize = (15,15), title = 'Suburbs with highest Avg price per sq.mtr')

plt.ylabel('Price', size = '20')

plt.xlabel('Suburb', size = '20')

# Now we will look at the lowest average price rates per square meter

sub\_price.nsmallest(10,['Mean PricePerSqmtr']).plot(kind='bar', figsize = (15,15), title = 'Suburbs with lowest Avg price per sq.mtr')

plt.ylabel('Price', size = '20')

plt.xlabel('Suburb', size = '20')

# Now we will present the visualization of rooms and price

plt.figure(figsize = (20,15))

sns.regplot(x = 'Rooms', y = 'Price', data = clean\_house)

plt.xlabel('Room', size = 20)

plt.ylabel('Price', size = 20)

plt.title('Price of the room')

# The relation of number of bedrooms and their price

plt.figure(figsize = (20,15))

sns.regplot(x = 'Bedroom2', y = 'Price', data = clean\_house)

plt.title('Plot representing number of bedroom and their price', size = '18')

# Visualization of landsize and price, this will determine how the size of the

# land will affects price

plt.figure(figsize = (20,15))

sns.regplot(x = 'Landsize', y = 'Price', data = clean\_house)

plt.title('Regression plot of landsize and price', size = '18')

# Building Area and price relation.

plt.figure(figsize=(20,15))

sns.regplot(x = 'BuildingArea', y = 'Price', data = clean\_house)

plt.title('Regression plot for building area and price', size = '18')

# The relation of the price with the distance of the central district building

plt.figure(figsize = (20,15))

sns.regplot(x = 'Distance', y = 'Price', data = clean\_house)

plt.title('Regression plot based on price and distance of house from council area ', size = '18')

# Average square per meter price of the house based on the region.

region\_price = clean\_house[['Regionname', 'PricePerSqmtr']].groupby(['Regionname']).agg(['mean'])

region\_price

region\_price.plot(kind = 'bar', figsize = (20,15))

plt.title('Average square per meter price of house base on region', size = '18')

plt.ylabel('Price', size = '18')

plt.xlabel('Region Name', size = '18')

# Price of the house based on the year they were built on

year\_price = clean\_house[['YearBuilt', 'Price']].groupby(['YearBuilt']).agg(['mean'])

year\_price

year\_price.plot(figsize = (20,15))

plt.title('Average Price of the houses based on year built', size = '18')

plt.xlim([1800,2019])

plt.ylabel('Price', size = '18')

plt.xlabel('Year Built', size = '18')

# Relation of price and car, the price of the house based on the number of

# cars in the house

car\_price = clean\_house[['Car', 'Price']].groupby(['Car']).agg(['mean'])

car\_price.plot(kind = 'bar', figsize = (20,15))

plt.title('Average Price based on the number of car space in the house', size = '18')

plt.ylabel('Price', size = '18')

plt.xlabel('Car', size = '18')

# Plot : Total sale of the property by real estate agent based on price

estate\_price = clean\_house[['SellerG', 'Price']].groupby(['SellerG']).agg(['sum'])

# Change the column name

estate\_price.columns = ['Sale\_by\_sum']

estate\_price.nlargest(10,['Sale\_by\_sum']).plot(kind='bar', figsize = (20,15))

plt.title('TotalSale of property by real estate agent', size = '18')

plt.ylabel('Price', size = '18')

plt.xlabel('SelleG', size = '18')

# Plotting Dependant variable or target variable 'Price'

plt.subplots(figsize = (15,10))

sns.distplot(clean\_house['Price'], fit = stats.norm)

(mu, sigma) = stats.norm.fit(clean\_house['Price'])

plt.legend(['Normal distribution ($\mu=$ {:.2f} and $\sigma=$ {:.2f})'.format(mu, sigma)], loc='best')

plt.ylabel('Frequency')

# Probability plot

fig = plt.figure()

stats.probplot(clean\_house['Price'], plot=plt)

plt.show()

# Visualization of correlation plot with the help of heat map

corr\_plot = clean\_house.corr()

plt.figure(figsize = (20,15))

sns.heatmap(corr\_plot, annot = True)

plt.title('Heatmap that represents correlation between all the numeric attributes in the dataset', size = '18')

# Selecting features

corr\_plot.sort\_values(['Price'], ascending = False, inplace = True)

corr\_plot.Price

# Object / categorical variables in the dataset

clean\_house.select\_dtypes(include = ['object']).head()

# Assigning a variable to these values.

cat\_attribute = ('Type','Method','Regionname', 'CouncilArea', 'Date',

'Address', 'SellerG', 'Suburb')

# Now we will perform Label Encoding

from sklearn.preprocessing import LabelEncoder

for var in cat\_attribute:

label = LabelEncoder()

label.fit(list(clean\_house[var].values))

clean\_house[var] = label.transform(list(clean\_house[var].values))

# Let us check the values of categorical variables

clean\_house.head()

Regression Analysis

# Variable selection based on the correlation plot.

# Here we have removed Bedroom2 because Rooms and Bedroom2 have almost similar values which can generate duplication and

# overfitting

clean\_house1 = clean\_house[['Price', 'Rooms', 'Bathroom', 'Type', 'Method', 'Regionname', 'CouncilArea', 'SellerG', 'Suburb']]

clean\_house1.head()

target\_price = clean\_house1.pop('Price')

target\_price.head()

# Genereating testing and training data

percent = 0.8

size = int(percent \* len(clean\_house1))

ch1\_train = clean\_house1[:size]

ch1\_test = clean\_house1[size:]

tpercent = 0.8

tsize = int(tpercent \* len(target\_price))

target\_train = target\_price[:tsize]

target\_test = target\_price[tsize:]

# Performing multiple regression with non - normalized dataset and target variable

from sklearn.linear\_model import LinearRegression, Lasso, Ridge, ElasticNet

from sklearn.neighbors import KNeighborsRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.ensemble import AdaBoostRegressor

from sklearn.tree import DecisionTreeRegressor

from sklearn import metrics

# we will be using dictionary to perform create the models

Reg\_model = { 'Linear Regression': {'Model': LinearRegression()},

'Ridge Regression': {'Model': Ridge()},

'ElasticNet Regression': {'Model': ElasticNet()},

'KNeighbors Regressor': {'Model': KNeighborsRegressor()},

'Random Forest Regressor': {'Model': RandomForestRegressor()},

'Gradient Boosting Regressor': {'Model': GradientBoostingRegressor()},

'Ada Boost Regressor': {'Model': AdaBoostRegressor(n\_estimators = 10)},

'Decision Tree Regressor': {'Model': DecisionTreeRegressor(max\_depth = 5, min\_samples\_split = 4, random\_state = 1)},

}

for R\_model in Reg\_model:

Reg\_model[R\_model]['Prediction data'] = None

Reg\_model[R\_model]['Error value'] = { 'MAE': None,

'MSE': None,

'RMSE': None}

Reg\_model[R\_model]['Scores'] = {'R-Squared': None}

for R\_model in Reg\_model:

print('Executing: ', Reg\_model[R\_model]['Model'])

Reg\_model[R\_model]['Model'].fit(ch1\_train, target\_train)

Reg\_model[R\_model]['Predicted data'] = Reg\_model[R\_model]['Model'].predict(ch1\_test)

Reg\_model[R\_model]['Error value']['MAE'] = metrics.mean\_absolute\_error(target\_test, Reg\_model[R\_model]['Predicted data'])

Reg\_model[R\_model]['Error value']['MSE'] = metrics.mean\_squared\_error(target\_test, Reg\_model[R\_model]['Predicted data'])

Reg\_model[R\_model]['Error value']['RMSE'] = np.sqrt(Reg\_model[R\_model]['Error value']['MSE'])

Reg\_model[R\_model]['Scores']['R-Squared'] = metrics.r2\_score(target\_test, Reg\_model[R\_model]['Predicted data'])

print('Mean Absolute Error: ', Reg\_model[R\_model]['Error value']['MAE'])

print('Mean Squared Error: ', Reg\_model[R\_model]['Error value']['MSE'])

print('RMSE: ', Reg\_model[R\_model]['Error value']['RMSE'])

print('R-Squared: ', Reg\_model[R\_model]['Scores']['R-Squared'])

print('\n')

Part 2

# Normalizing the attributes

from sklearn import preprocessing

min\_max\_scaler = preprocessing.MinMaxScaler()

min\_max\_scaler.fit(ch1\_train)

min\_max\_scaler.fit(ch1\_test)

ch1\_train\_norm = min\_max\_scaler.fit\_transform(ch1\_train)

ch1\_test\_norm = min\_max\_scaler.fit\_transform(ch1\_test)

ch1\_train\_norm.shape

ch1\_test\_norm.shape

target\_train.shape

target\_test.shape

from sklearn.linear\_model import LinearRegression, Lasso, Ridge, ElasticNet

from sklearn.neighbors import KNeighborsRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.ensemble import AdaBoostRegressor

from sklearn.tree import DecisionTreeRegressor

from sklearn import metrics

# we will be using dictionary to perform create the models

Reg\_model = { 'Linear Regression': {'Model': LinearRegression()},

'Ridge Regression': {'Model': Ridge()},

'ElasticNet Regression': {'Model': ElasticNet()},

'KNeighbors Regressor': {'Model': KNeighborsRegressor()},

'Random Forest Regressor': {'Model': RandomForestRegressor()},

'Gradient Boosting Regressor': {'Model': GradientBoostingRegressor()},

'Ada Boost Regressor': {'Model': AdaBoostRegressor(n\_estimators = 10)},

'Decision Tree Regressor': {'Model': DecisionTreeRegressor(max\_depth = 5, min\_samples\_split = 4, random\_state = 1)},

}

for R\_model in Reg\_model:

Reg\_model[R\_model]['Prediction data'] = None

Reg\_model[R\_model]['Error value'] = { 'MAE': None,

'MSE': None,

'RMSE': None}

Reg\_model[R\_model]['Scores'] = {'R-Squared': None}

for R\_model in Reg\_model:

print('Executing: ', Reg\_model[R\_model]['Model'])

Reg\_model[R\_model]['Model'].fit(ch1\_train\_norm, target\_train)

Reg\_model[R\_model]['Predicted data'] = Reg\_model[R\_model]['Model'].predict(ch1\_test\_norm)

Reg\_model[R\_model]['Error value']['MAE'] = metrics.mean\_absolute\_error(target\_test, Reg\_model[R\_model]['Predicted data'])

Reg\_model[R\_model]['Error value']['MSE'] = metrics.mean\_squared\_error(target\_test, Reg\_model[R\_model]['Predicted data'])

Reg\_model[R\_model]['Error value']['RMSE'] = np.sqrt(Reg\_model[R\_model]['Error value']['MSE'])

Reg\_model[R\_model]['Scores']['R-Squared'] = metrics.r2\_score(target\_test, Reg\_model[R\_model]['Predicted data'])

print('Mean Absolute Error: ', Reg\_model[R\_model]['Error value']['MAE'])

print('Mean Squared Error: ', Reg\_model[R\_model]['Error value']['MSE'])

print('RMSE: ', Reg\_model[R\_model]['Error value']['RMSE'])

print('R-Squared: ', Reg\_model[R\_model]['Scores']['R-Squared'])

print('\n')

Part 3

# Log transforming the target attribute since it was right skewed.

clean\_house['Price'] = np.log1p(clean\_house['Price'])

# Now we will generate normalization plot.

plt.subplots(figsize=(15,10))

sns.distplot(clean\_house['Price'], fit=stats.norm)

(mu, sigma) = stats.norm.fit(clean\_house['Price'])

plt.legend(['Normal distribution: ($\mu=$ {:.2f} and $\sigma=$ {:.2f})'.format(mu, sigma)], loc = 'best')

plt.ylabel('Frequency')

fig = plt.figure()

stats.probplot(clean\_house['Price'], plot=plt)

plt.show()

Regression Using log Transformed target Attribute and normalized attribute

clean\_house1 = clean\_house[['Price', 'Rooms', 'Bathroom', 'Type', 'Method', 'Regionname', 'CouncilArea', 'SellerG', 'Suburb']]

clean\_house1.head()

# Genereating testing and training data

percent = 0.8

size = int(percent \* len(clean\_house1))

ch1\_train = clean\_house1[:size]

ch1\_test = clean\_house1[size:]

tpercent = 0.8

tsize = int(tpercent \* len(target\_price))

target\_train = target\_price[:tsize]

target\_test = target\_price[tsize:]

target\_price = clean\_house1.pop('Price')

target\_price.head()

min\_max\_scaler = preprocessing.MinMaxScaler()

min\_max\_scaler.fit(ch1\_train)

min\_max\_scaler.fit(ch1\_test)

ch1\_train\_norm = min\_max\_scaler.fit\_transform(ch1\_train)

ch1\_test\_norm = min\_max\_scaler.fit\_transform(ch1\_test)

for R\_model in Reg\_model:

print('Executing: ', Reg\_model[R\_model]['Model'])

Reg\_model[R\_model]['Model'].fit(ch1\_train\_norm, target\_train)

Reg\_model[R\_model]['Predicted data'] = Reg\_model[R\_model]['Model'].predict(ch1\_test\_norm)

Reg\_model[R\_model]['Error value']['MAE'] = metrics.mean\_absolute\_error(target\_test, Reg\_model[R\_model]['Predicted data'])

Reg\_model[R\_model]['Error value']['MSE'] = metrics.mean\_squared\_error(target\_test, Reg\_model[R\_model]['Predicted data'])

Reg\_model[R\_model]['Error value']['RMSE'] = np.sqrt(Reg\_model[R\_model]['Error value']['MSE'])

Reg\_model[R\_model]['Scores']['R-Squared'] = metrics.r2\_score(target\_test, Reg\_model[R\_model]['Predicted data'])

print('Mean Absolute Error: ', Reg\_model[R\_model]['Error value']['MAE'])

print('Mean Squared Error: ', Reg\_model[R\_model]['Error value']['MSE'])

print('RMSE: ', Reg\_model[R\_model]['Error value']['RMSE'])

print('R-Squared: ', Reg\_model[R\_model]['Scores']['R-Squared'])

print('\n')

Part 4

clean\_house1 = clean\_house[['Price', 'Rooms', 'Bathroom', 'Type', 'Method','CouncilArea']]

clean\_house1.head()

# Genereating testing and training data

percent = 0.8

size = int(percent \* len(clean\_house1))

ch1\_train = clean\_house1[:size]

ch1\_test = clean\_house1[size:]

tpercent = 0.8

tsize = int(tpercent \* len(target\_price))

target\_train = target\_price[:tsize]

target\_test = target\_price[tsize:]

target\_price = clean\_house1.pop('Price')

target\_price.head()

min\_max\_scaler = preprocessing.MinMaxScaler()

min\_max\_scaler.fit(ch1\_train)

min\_max\_scaler.fit(ch1\_test)

ch1\_train\_norm = min\_max\_scaler.fit\_transform(ch1\_train)

ch1\_test\_norm = min\_max\_scaler.fit\_transform(ch1\_test)

for R\_model in Reg\_model:

print('Executing: ', Reg\_model[R\_model]['Model'])

Reg\_model[R\_model]['Model'].fit(ch1\_train\_norm, target\_train)

Reg\_model[R\_model]['Predicted data'] = Reg\_model[R\_model]['Model'].predict(ch1\_test\_norm)

Reg\_model[R\_model]['Error value']['MAE'] = metrics.mean\_absolute\_error(target\_test, Reg\_model[R\_model]['Predicted data'])

Reg\_model[R\_model]['Error value']['MSE'] = metrics.mean\_squared\_error(target\_test, Reg\_model[R\_model]['Predicted data'])

Reg\_model[R\_model]['Error value']['RMSE'] = np.sqrt(Reg\_model[R\_model]['Error value']['MSE'])

Reg\_model[R\_model]['Scores']['R-Squared'] = metrics.r2\_score(target\_test, Reg\_model[R\_model]['Predicted data'])

print('Mean Absolute Error: ', Reg\_model[R\_model]['Error value']['MAE'])

print('Mean Squared Error: ', Reg\_model[R\_model]['Error value']['MSE'])

print('RMSE: ', Reg\_model[R\_model]['Error value']['RMSE'])

print('R-Squared: ', Reg\_model[R\_model]['Scores']['R-Squared'])

print('\n')

Clustering

# Inporting the dataset

cluster\_house = pd.read\_csv('D:/DEPAUL/Depaul 5th quarter (AUTUMN)/DSC 478/PROJECT/int64index\_CLEAN.csv')

cluster\_house.head()

# Selecting the integer datatypes

cluster\_house.select\_dtypes(include = ['int64']).head()

# Selecting object / catrgorical data types

cluster\_house.select\_dtypes(include = ['object']).head()

# We will store the Caterogical attribute Type in a seperate file to later compute it easily while performing clustering

Type = cluster\_house['Type']

Type\_var = Type.to\_csv('D:/DEPAUL/Depaul 5th quarter (AUTUMN)/DSC 478/PROJECT/Type\_clust.csv', index = False)

# Selecting integer numerical attributes except Bedroom2, because Room and Bedroom2 have similarity in data.

Final\_clust = cluster\_house[['Rooms', 'Postcode', 'Bathroom', 'Car', 'YearBuilt', 'Propertycount']]

Final\_clust.head()

Normalize the data so that clustering can become easier

from sklearn import preprocessing

min\_max\_scaler = preprocessing.MinMaxScaler().fit(Final\_clust)

norm\_Final\_clust = min\_max\_scaler.transform(Final\_clust)

norm\_Final\_clust = pd.DataFrame(norm\_Final\_clust, columns = Final\_clust.columns)

norm\_Final\_clust.head()

# Now we will perform clustering on the norm\_Final\_clust

from sklearn.cluster import KMeans

cluster\_class = pd.read\_csv('D:/DEPAUL/Depaul 5th quarter (AUTUMN)/DSC 478/PROJECT/Type\_clust.csv', header = None)

cluster\_class.head(10)

# We will perform 5 clusters

kmeans\_house = KMeans(n\_clusters = 5, max\_iter=500, verbose = 1)

kmeans\_house.fit(norm\_Final\_clust)

# Predicting clusters

cluster = kmeans\_house.predict(norm\_Final\_clust)

pd.DataFrame(cluster, columns=["Clusters"])

# Now we will generate the centroids that allow aggregation

# and characterization of each cluster

pd.options.display.float\_format='{:,.2f}'.format

centroid = pd.DataFrame(kmeans\_house.cluster\_centers\_, columns = norm\_Final\_clust.columns)

centroid

def size\_cluster(cluster):

size = {}

cluster\_label = np.unique(cluster)

n\_cluster = cluster\_label.shape[0]

for clust in cluster\_label:

size[clust] = len(norm\_Final\_clust[cluster == clust])

return size

size = size\_cluster(cluster)

for clust in size.keys():

print("Size pf cluster", clust, '=', size[clust])

Silhouette analysis

from sklearn import metrics

silhouettes\_method = metrics.silhouette\_samples(norm\_Final\_clust, cluster)

print(silhouettes\_method[:20])

print(silhouettes\_method.mean())

# Now we will plot the clusters on Silhouettes plot

def Silhouette(norm\_Final\_clust, cluster, metric='euclidean'):

from matplotlib import cm

from sklearn.metrics import silhouette\_samples

cluster\_labels = np.unique(cluster)

n\_clusters = cluster\_labels.shape[0]

silhouette\_values = metrics.silhouette\_samples(norm\_Final\_clust, cluster, metric = 'euclidean')

cluster\_ax\_lower, cluster\_ax\_upper = 0,0

cluster\_ticks = []

for a, b in enumerate(cluster\_labels):

cluster\_silhouette\_values = silhouette\_values[cluster == b]

cluster\_silhouette\_values.sort()

cluster\_ax\_upper += len(cluster\_silhouette\_values)

color = cm.jet(float(a)/n\_clusters)

pl.barh(range(cluster\_ax\_lower, cluster\_ax\_upper), cluster\_silhouette\_values, height = 1.0,

edgecolor = 'none', color = color)

cluster\_ticks.append((cluster\_ax\_lower + cluster\_ax\_upper) / 2)

cluster\_ax\_lower += len(cluster\_silhouette\_values)

silhouette\_average = np.mean(silhouette\_values)

pl.axvline(silhouette\_average, color = 'red', linestyle = '--')

pl.yticks(cluster\_ticks, cluster\_labels)

pl.ylabel('Cluster')

pl.xlabel('Silhouette coefficient')

pl.tight\_layout()

pl.show()

return

Silhouette(norm\_Final\_clust, cluster)

# Homogeneity and completeness test:

from sklearn.metrics import completeness\_score, homogeneity\_score

print(completeness\_score(cluster\_class[0], cluster))

print(homogeneity\_score(cluster\_class[0], cluster))