ECS784P – Data Analytics – 2018/19

House Price Prediction

Submitted by: - MANEESHA MAJEED, Student ID – 180911929

Group size: 3

Table of individual contribution by each member of my group, based on my subjective opinion

Student Name	Effort
Leila Jalaly Chaleshtary	33.34%
Maneesha Majeed	33.33%
Himanshu Thakkar	33.33%

Contents

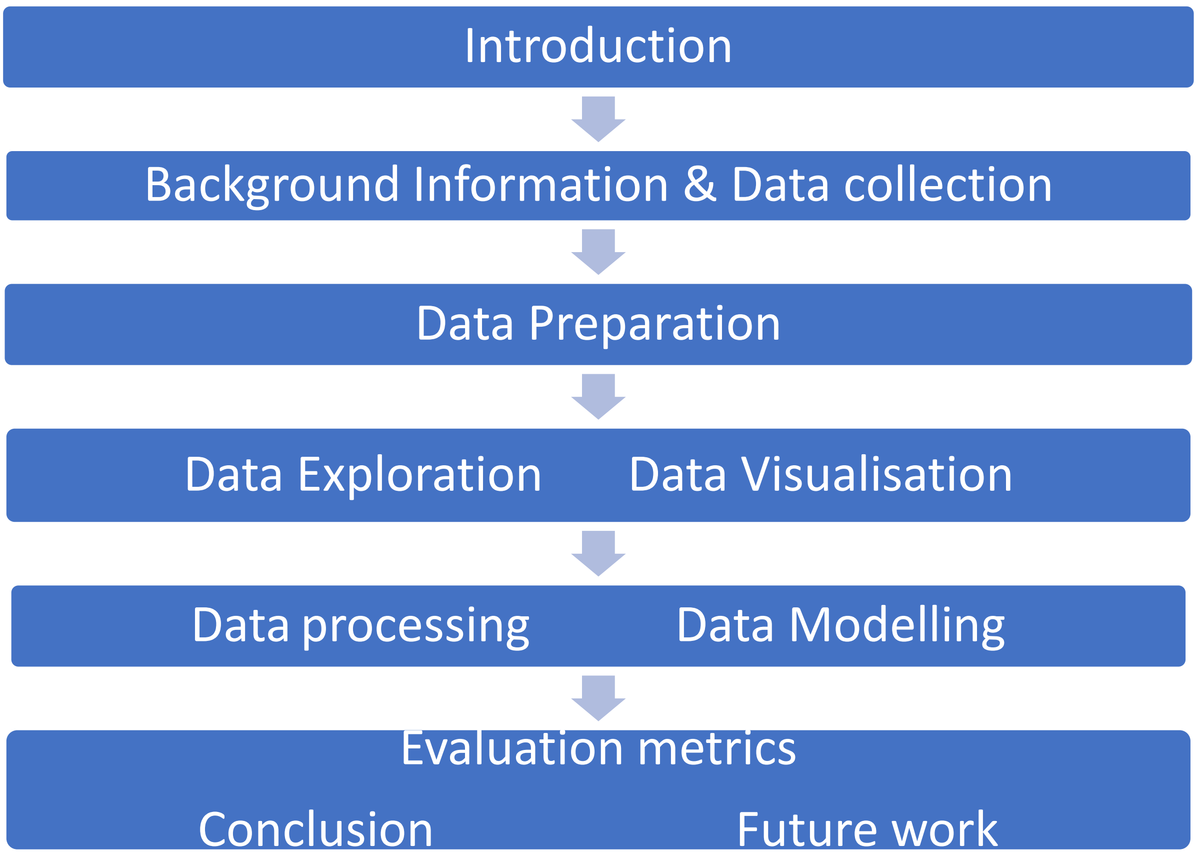
Introduction

Buying a property is an important decision in a person's life. A house is the most valuable thing many people will ever own. In Britain, two thirds of households own the house they live in. The house buying involves investing large sums of money and time and yet there is a persisting concern about the right deal. Besides the affordability of house, other factors such as the desirability of the location and the long-term investment prospects also affect decision making process. This coursework involves deriving a house price prediction model.

Stating the hypothesis that the house price depends on number of factors including its location, number of bedrooms, its lease type, distance from the station; this coursework makes use of supervised learning type of machine learning techniques in making a house price prediction model.

There are different machine learning algorithms to predict the house prices. In order to select a best performing prediction method, various regression methods are explored and compare to find an appropriate fit. Methods discussed includes Linear regression, Ridge, Kernel Ridge, Random forest Regressor, Lasso, XGBoosting Regressor and Gradient Boosting Regressor. We compare and assess the performance of these methods by examining R^2 score and RMSE. A cross validation is done to assess the achieved accuracy and thereby examine the scope to improve the accuracy score.

The coding is done in Python and implemented using Jupyter Notebook. The structure of report is as per the following flow chart.



Background Information

Area of Study

We considered many areas as study of interest. This largely included various parts of inner and outer London. Properties in inner London are dearer than properties located in outer London. The location of the property plays an important role in its price. The area of study for this project is Kingston Upton Thames. This place is filled with history, culture and top schools and located in an enviable position between Richmond park and River Thames [4]. The properties in this area are cheaper than Richmond-upon-Thames and Merton but expensive than Greenwich and Hounslow. The borough has a good selection of terraced and detached properties, as well as smart apartments for young professionals and student-friendly terraces [5]. The area has good transport links and low crime rates.

Most of the town centre is part of the KT1 postcode while some areas north of Kingston railway station have post code KT2 instead [6]. Properties in post codes KT1 and KT2 are covered in this assignment.

The concentration of different types of houses vary across London with houses in central London are flats while house in outside London are detached. Given the density of properties in the capital, it is prime for us to model the relationship between house prices and various factors that affect it.

Project Objective

The aim of the project is to build a model that can predict the price of the house by giving in the attribute that show high correlation value to the output price.

Another main goal of this thesis is to examine the importance of each predictor in explaining price variation for a given set of housing attributes.

Our approach is to prepare the collected raw data and do primary exploration analysis. After the basic visualisation is done the data is pre-processed. The data is then split into two parts. The first part trains the model and produces an inferred function. The second part of the data tests the accuracy of the model thus build. Both the data sets are subject to various regression models. Finally, the accuracy score is evaluated across the models.

External Libraries Used

Pandas

Pandas is an open source, BSD-licensed library that has easy to use data structures and data analysis tools for Python programming language. Pandas is great for data preparation, it has tools for reading and writing data between in-memory data structures and in different formats; this project involves data files in csv format. It has high performance merging and joining of data sets. The columns can be inserted and deleted from data structures [3].

NumPy

Numpy is the fundamental package for scientific computing with Python. Numpy has a powerful N-dimensional array object. It has useful linear algebra, Fourier transform, and random number capabilities. Besides, it can be also used as an efficient multi-dimensional container of generic data [10].

The libraries Scipy and Scikit-learn is based on Numpy

Matplotlib

Matplotlib is a Python 2D plotting library that produces publication quality features. Matplotlib aids in generating plots, histograms, power spectra, bar charts, error charts, scatterplots etc with just few lines of code. It can be used in variety of development environment including Jupyter Notebook [8].

Seaborn

Seaborn is a data visualisation library based on Matplotlib, closely integrated with Pandas data structures. It provides high level interface for drawing attractive and informative statistical graphics [9]. Its functionality includes

- Specialised support for using categorical variables to show aggregate statistics.
- Automatic estimation and plotting of linear regression models for different kind of dependent variables.
- Convenient views onto the overall structure of complex datasets.

Scikit-learn

Scikit-learn is a machine learning library that features various classification, regression and clustering algorithms including support vector machine (SVM), random forests, gradient boosting and k-means. It is designed to interoperate with libraries Numpy and Scipy [11].

Data Collection

Our first choice for data collection was 'HM Land Registry Open Data' [12]. The data available extends over number of years. When a sample of data was extracted from Land registry, it contained too fewer features. It represented us a risk of underfitting the prediction model and as such we start to search for sources with more features. One such website is Rightmove, it uses the sold house values from HM Land Registry and provides additional important information on the sold property such as number of Bedrooms and distance to station. Rightmove provides the data for last five years and this formed the basis of our search criteria. We used Parsehub to extract data from Rightmove. Other features of data include ward and location co-ordinates. Both the data sets are obtained separately and merged during data preparation process.

Extracting the data using Parsehub.

Parsehub is a web scrapping tool. It is an easy to learn, visual data extraction tool which is powerful and flexible, yet simple to use [2]. The collected data is downloadable in CSV format. Following figure shows a sample set up of data extraction using Parsehub.

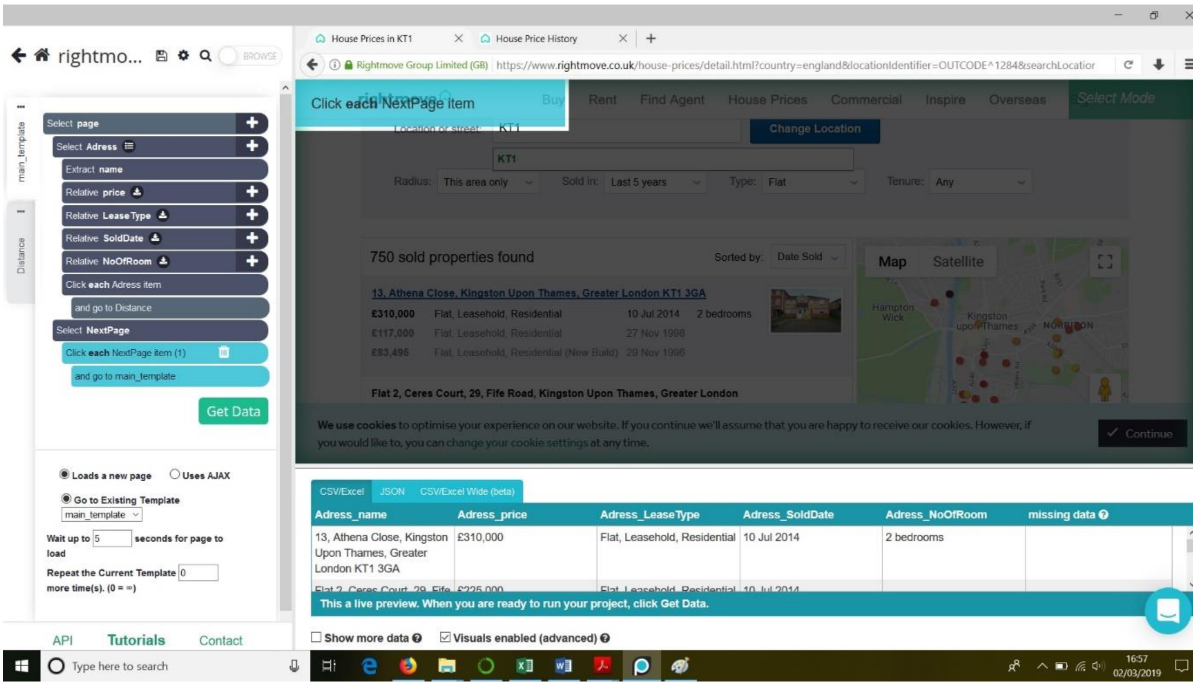


Figure: - Set up on Parsehub for extracting data

The selected features to extract are listed as follows

- Address of the property
- Sold Price
- Property Lease Type: - Leasehold or Freehold
- Number of Bedrooms
- Distance to the nearest station

The free version of the Parsehub has a limitation to collect the data from up to a maximum of 200 pages in a single run. Such a single run took 45 minutes to execute. The data extraction was split into multiple runs and finally data was merged into single file.

The figure below shows a snapshot of the data obtained, the file is saved as Combined_data_230219.csv.

1	selection1_name	selection1_prices	selection1_house_type	selection1_date	selection1_bedrooms	selection1_selectior
2	10, Lord Chancellor Walk, Kingston Upon Thames, Greater London KT2 7HG	Â£1,050,000	Detached, Freehold, Residential	19-Dec-18	5 bedrooms	(0.7 miles)
3	16, Clevedon Road, Kingston Upon Thames, Greater London KT1 3AD	Â£905,000	Detached, Freehold, Residential	04-Dec-18	5 bedrooms	(0.2 miles)
4	14, Albion Road, Kingston Upon Thames, Greater London KT2 7BZ	Â£1,225,000	Detached, Freehold, Residential	03-Dec-18	4 bedrooms	(0.5 miles)
5	Chase End, Fitzgeorge Avenue, New Malden, Greater London KT3 4SH	Â£1,030,000	Detached, Freehold, Residential	30-Nov-18	4 bedrooms	(0.9 miles)
6	55, Cobham Road, Kingston Upon Thames, Greater London KT1 3AE	Â£910,000	Detached, Freehold, Residential	22-Nov-18	4 bedrooms	(0.1 miles)
7	4a, Grove Lane, Kingston Upon Thames, Greater London KT1 2SU	Â£855,000	Detached, Freehold, Residential	16-Nov-18	3 bedrooms	(0.7 miles)
8	40, Burton Road, Kingston Upon Thames, Greater London KT2 5TF	Â£1,250,000	Detached, Freehold, Residential	14-Nov-18	5 bedrooms	(0.5 miles)
9	12, Brook Gardens, Kingston Upon Thames, Greater London KT2 7ET	Â£1,900,000	Detached, Freehold, Residential	14-Nov-18	4 bedrooms	(0.4 miles)
10	10, Chesham Road, Kingston Upon Thames, Greater London KT1 3AQ	Â£626,500	Detached, Freehold, Residential	09-Nov-18	2 bedrooms	(0.2 miles)
11	18, Derwent Avenue, London, Greater London SW15 3RD	Â£1,137,500	Detached, Freehold, Residential	05-Nov-18	5 bedrooms	(1.7 miles)
12	Wildcroft, Coombe Park, Kingston Upon Thames, Greater London KT2 7JB	Â£5,575,000	Detached, Freehold, Residential	31-Oct-18	7 bedrooms	(1.3 miles)
13	63, Canbury Avenue, Kingston Upon Thames, Greater London KT2 6JR	Â£1,000,000	Detached, Freehold, Residential	26-Oct-18	4 bedrooms	(0.4 miles)
14	46, Kings Road, Kingston Upon Thames, Greater London KT2 5HS	Â£720,000	Detached, Freehold, Residential	23-Oct-18		
15	8, Parkgate Close, Kingston Upon Thames, Greater London KT2 7LU	Â£820,000	Detached, Freehold, Residential	16-Oct-18	4 bedrooms	(0.8 miles)
16	Silverwood House, George Road, Kingston Upon Thames, Greater London KT2 7NR	Â£4,350,000	Detached, Freehold, Residential	15-Oct-18	5 bedrooms	(0.6 miles)
17	3, Manston Grove, Kingston Upon Thames, Greater London KT2 5GF	Â£1,013,000	Detached, Freehold, Residential	12-Oct-18	4 bedrooms	(1.0 miles)
18	2, Brough Close, Kingston Upon Thames, Greater London KT2 5DB	Â£654,000	Detached, Freehold, Residential	05-Oct-18		

Fig: - Sample of the data obtained

Other features useful in training a regression model are latitude, longitude, easting, northing and ward; the property is located in. We chose to represent an address by its postcode as it is a concise and unique reference to location of the property.

The postcodes and the location co-ordinates in KT1 and KT2 area were downloaded from Doogal website. The file is saved as 'geographical_postcode_kingston_upon_thames.csv'. The following figure shows a snapshot of the data

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Postcode	In Use?	Latitude	Longitude	Easting	Northing	Grid Ref	Ward	Introduced	Terminated	Altitude				
2	KT1 1AA	Yes	51.40869	-0.30447	518023	169104	TQ180691	Grove	01/02/1990		12				
3	KT1 1AB	No	51.41233	-0.30259	518144	169512	TQ181695	Grove	01/01/2005	01/08/2014	14				
4	KT1 1AD	Yes	51.41121	-0.30114	518248	169390	TQ182693	Grove	01/03/2010		13				
5	KT1 1AE	Yes	51.41261	-0.30736	517812	169535	TQ178695	Grove	01/02/2011		9				
6	KT1 1AF	Yes	51.41248	-0.30721	517823	169521	TQ178695	Grove	01/02/2011		10				
7	KT1 1AH	No	51.41076	-0.30771	517792	169329	TQ177693	Grove	01/06/2000	01/04/2009	12				
8	KT1 1AJ	Yes	51.41262	-0.30745	517806	169536	TQ178695	Grove	01/01/2013		9				
9	KT1 1AL	Yes	51.40915	-0.30735	517821	169151	TQ178691	Grove	01/06/2012		11				
10	KT1 1AN	Yes	51.41187	-0.30092	518261	169464	TQ182694	Grove	01/09/2012		12				
11	KT1 1AP	Yes	51.41264	-0.30112	518246	169549	TQ182695	Grove	01/01/2015		13				
12	KT1 1AZ	Yes	51.40876	-0.30786	517787	169107	TQ177691	Grove	01/12/2001		11				
13	KT1 1BA	No	51.40868	-0.30623	517900	169100	TQ179690	Grove	01/01/1995	01/06/1996	12				
14	KT1 1BB	Yes	51.40888	-0.30568	517938	169123	TQ179691	Grove	01/01/1980		12				
15	KT1 1BD	No	51.40868	-0.30623	517900	169100	TQ179690	Grove	01/01/1980	01/01/1995	12				
16	KT1 1BE	No	51.40868	-0.30623	517900	169100	TQ179690	Grove	01/01/1980	01/01/1995	12				
17	KT1 1BG	No	51.40866	-0.3048	518000	169100	TQ180690	Grove	01/01/1980	01/01/1995	12				
18	KT1 1BJ	Yes	51.40947	-0.3047	518005	169191	TQ180691	Grove	01/10/1984		15				

Fig: - Post code data from Doogal website

Further information on how we prepare the variables for use in our prediction model is explained in the following sections.

Data Preparation

The raw data is unsuitable for analysis. To be useful for predictive modelling the data must be first cleaned. The aim of data preparation is to remove unwanted features in the data and to rearrange the remaining data, wherever applicable, to consolidate raw data collected from different sources and to provide access to consistent and accurate data suitable for processing. 'Approximation' is applied to the missing data values. The data is available in two separate csv files, as described earlier: -

- Combined_data_230219.csv
- geographical_postcode_kingston_upon_thames.csv

Initially data cleaning is done on 'combined_data_230219.csv' file. It consists of following steps

1. The first step involves, renaming the data columns to a meaningful descriptive name. The post code from the address is split into new column. The house type column in the data represents a combined information. This is split into three columns as House Type, Freehold/Leasehold and Res_new/old. The code is shown in following snippet: -

```
#columns names obtained from the retrieved data which needed to be changed into the required form
data.columns = ['Address','Sale_price','House_types','Sale_date','Bedrooms','Stationdist_miles']
```

```
data.columns
```

```
Index(['Address', 'Sale_price', 'House_types', 'Sale_date', 'Bedrooms',
       'Stationdist_miles'],
      dtype='object')
```

```
#Splitting the house type into three seperate columns
data[['House_type','Freehold_leasehold','Res_new_old']] = data['House_types'].str.split(',', expand=True)
data = data.drop('House_types', axis=1)
```

```
#Cleaning for  ADDRESS column[0] and seperating POSTCODE to a new column
data['Postcode']=data['Address'].str.rsplit(',').str[-1]
data['Postcode'] = data['Postcode'].str.replace("Greater London","")
data['Address']=data['Address'].str.replace(' Kingston Upon Thames,','')
data['Address']=data['Address'].str.replace('Greater London','')
data['Address']=data['Address'].str.replace('London','')
data['Address']=data['Address'] .map(lambda x: str(x)[:-10])

# Cleaning for saleprice,Bedrooms and stationsdist_miles columns
data['Bedrooms'] = data['Bedrooms'].str.replace("bedrooms","")
data['Bedrooms'] = data['Bedrooms'].str.replace("bedroom","")
data['Stationdist_miles'] = data['Stationdist_miles'].str.replace("(","")
data['Stationdist_miles'] = data['Stationdist_miles'].str.replace("miles","")
data['Stationdist_miles'] = data['Stationdist_miles'].str.replace(")","")
data['Sale_price']=data['Sale_price'].str.replace(',','')
data['Sale_price'] = data['Sale_price'].str.replace("£","")
```

2. A new column containing the first part of post code is created (e.g. KT1, KT2). Two more new columns are created, these contain separated values of sold year and sold month.

```
#SPLITTING POSTCODE INTO KT1 AND KT2 region
data['KT1_KT2']=data['Postcode'].map(lambda x: str(x)[:-3])
```

```
#Seperating the sale date into two seperate columns with sale year and sale month
data['Sale_year']=pd.DatetimeIndex(data['Sale_date']).year
data['Sale_month']=pd.DatetimeIndex(data['Sale_date']).month
data.head()
```

3. Third step involved filling in the missing information. There was missing information on some of the entries with regards to number of bedroom and distance to station. To assume a suitable value for missing number of bedrooms following approximation technique is applied:

- The data is separated year wise and sorted by price sold for. For a given house that has a missing number of bedrooms, it is matched with a house with similar price tag that has known number of rooms in a given year. It is assumed that, at a give date, houses with similar price tags have same number of rooms. This method accounted for filling in 'number of bedrooms' for all the data records.
This is achieved in following code snippet: -

```
#Replacing the white spaces with nan
data = data.replace(r'^\s+$', np.nan, regex=True)
#This shows the number of null values in street_name,Bedrooms
null_columns=data.columns[data.isnull().any()]
data[null_columns].isnull().sum()
```

```
Bedrooms              1315
Stationdist_miles     1368
dtype: int64
```

```
#filling the null values for bedrooms
#year wise sorting with respect to sale price and filling values

data_2014=data[data['Sale_year']==2014]
data_2014_sorted=data_2014.sort_values('Sale_price')
data_2014_sorted['Bedrooms'].fillna(method='backfill',inplace=True)
data_2014_sorted['Bedrooms'].fillna(method='ffill',inplace=True)

data_2015=data[data['Sale_year']==2015]
data_2015_sorted=data_2015.sort_values('Sale_price')
data_2015_sorted['Bedrooms'].fillna(method='backfill',inplace=True)
data_2015_sorted['Bedrooms'].fillna(method='ffill',inplace=True)

data_2016=data[data['Sale_year']==2016]
data_2016_sorted=data_2016.sort_values('Sale_price')
data_2016_sorted['Bedrooms'].fillna(method='backfill',inplace=True)
data_2016_sorted['Bedrooms'].fillna(method='ffill',inplace=True)

data_2017=data[data['Sale_year']==2017]
data_2017_sorted=data_2017.sort_values('Sale_price')
data_2017_sorted['Bedrooms'].fillna(method='backfill',inplace=True)
data_2017_sorted['Bedrooms'].fillna(method='ffill',inplace=True)

data_2018=data[data['Sale_year']==2018]
data_2018_sorted=data_2018.sort_values('Sale_price')
data_2018_sorted['Bedrooms'].fillna(method='backfill',inplace=True)
data_2018_sorted['Bedrooms'].fillna(method='ffill',inplace=True)

data_update= pd.concat([data_2014_sorted, data_2018_sorted,data_2017_sorted,data_2016_sorted,data_2015_sorted])
len(data_update)
```

For data record that has missing value for distance to station, the approximation is rather straightforward. The data is sorted according to post code, the data for the missing value is then copied from a matching post code. This is done in following snippet.

```
#recheck null values count
null_columns=data_update.columns[data_update.isnull().any()]
data_update[null_columns].isnull().sum()
```

```
Stationdist_miles     1368
dtype: int64
```

```
#sorting according to postcode and filling stationdist_miles missing values
data_update=data_update.sort_values('Postcode')
#using backfill to fill missing values in dist_to_station
data_update['Stationdist_miles'].fillna(method='backfill',inplace=True)
```

In the second stage, data preparation is done on geographical_postcode_kingston_upon_thames.csv. This file contains Latitude, Longitude, Easting, Northing, Ward etc for all the post code in KT1 and KT2 area.

1. In the first step both the data sets are merged. The merge, by default is performed in the common column i.e. postcode. The motive to merge is to get the latitude, longitude ward etc parameters for every property sold. Obviously, there are multiple properties with the same post code, the ‘merge’ action makes all the geographical parameters available to every property sold. The following code snippet show this: -

```
#Merging the two datasets to form the final dataset
mer_data=pd.merge(df_readcsv,geo)
#Removing the columns which are not required for modelling
del mer_data['Grid Ref']
del mer_data['Terminated']
del mer_data['In Use?']
del mer_data['Introduced']
# mer_data['Introduced']=mer_data['Introduced'].str.replace('/','-')
# mer_data['Introduced']=mer_data['Introduced'].map(lambda x: str(x)[3:])
mer_data
```

	Address	POSTCODE	KT1_KT2	Sale_price	Bedrooms	House_type	Freehold_leasehold	Res_new_old	Stationdist_miles	Sale_date	Sale_year	Sale_
0	Flat 40, Elder House, 4, Water Lane	KT1 1AE	KT1	435000	2	Flat	Leasehold	Residential	0.4	05-Oct-15	2015	
1	Flat 50, Elder House, 4, Water Lane	KT1 1AE	KT1	1295000	4	Flat	Leasehold	Residential	0.4	05-Feb-16	2016	
2	Flat 35, Elder House, 4, Water Lane	KT1 1AE	KT1	550000	2	Flat	Leasehold	Residential	0.4	19-Oct-17	2017	
	Flat 47,											

2. Next, the unwanted data columns are deleted. These are ‘grid ref’, ‘Terminated’ and ‘In use’ columns. It is perceived that these features are not needed in modelling the solution for the hypothesis.
3. In the third step, the columns are assigned meaningful names. Any data other than KT1/KT2 postcode is deleted. The whitespaces are trimmed out as well.

The features obtained are shown in the following diagram

FEATURES OBTAINED

1.Address
2.Postcode
3.KT1/KT2
4.Sale price
5.Bedrooms
6.House_type
7.Freehold/Leasehold
8.Res_new/old
9.Staiondist_miles(Station distance in miles)
10.Sale_date
11.Sale_year
12.Sale_month
13.Latitude
14.Longitude
15.Easting
16.Northing
17.Ward
18.Introduced
19.Altitude

Fig: - Final set of data features obtained

Data Exploration

Data Exploration involves using visual exploration to understand what is in a dataset and the characteristics of data such as size of data, completeness of data, correctness of data and possible relations between data elements [13].

Step 1 – This step involves visualising the volume and structure of data. The following snippet of code shows how descriptive statistics are obtained.

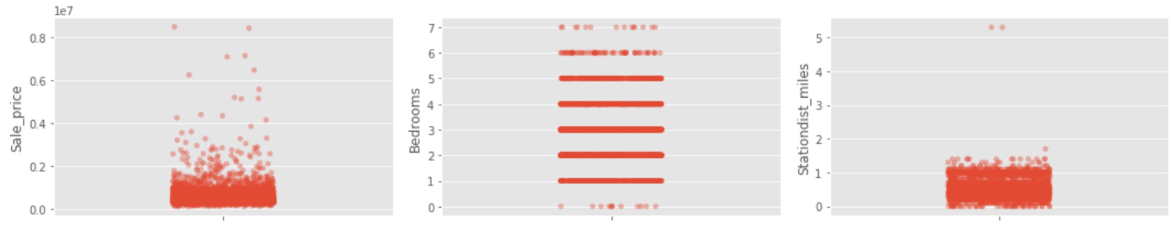
```
import pandas as pd
import numpy as np
#importing plotting libraries
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
plt.style.use('ggplot')

#Reading the csv file and showing its shape along with first 5 rows
mer = pd.read_csv('mer.csv')

#outputs the no of rows and columns in the dataset,
print('=> The number of rows and columns in dataset is (rows, columns):')
print(mer.shape,'\n')

#outputs the columns header,
print('=> A sample of the column headers in the dataset are:')
print(mer.iloc[:,2:8].columns.values)

#outputs the summary statistics and info,
print('\n=> These are the summary statistics for a sample of the columns:\n')
print(mer.iloc[:,10:].describe())
print('\n=> These are the info for a sample of the columns:\n')
print(mer.iloc[:,11:18].info())
```

```
=> The number of rows and columns in dataset is (rows, columns):
(3225, 18)

=> A sample of the column headers in the dataset are:
['KT1_KT2' 'Sale_price' 'Bedrooms' 'House_type' 'Freehold_leasehold'
 'Res_new_old']

=> These are the summary statistics for a sample of the columns:

         Sale_year   Sale_month     Latitude    Longitude        Easting  \
count  3225.000000  3225.000000  3225.000000  3225.000000    3225.000000
mean   2015.868837     6.813953    51.414063    -0.293917  518742.682171
std       1.396059     3.310839     0.007426     0.010581     733.160051
min    2014.000000     1.000000    51.398340    -0.312499  517411.000000
25%    2015.000000     4.000000    51.408764    -0.302632  518145.000000
50%    2016.000000     7.000000    51.413937    -0.295138  518667.000000
75%    2017.000000    10.000000    51.418660    -0.287458  519199.000000
max    2018.000000    12.000000    51.431053    -0.252341  521622.000000

            Northing     Altitude
count    3225.000000  3225.000000
mean   169718.792558    15.845271
std       828.475230    10.084349
min    167951.000000     7.000000
25%    169114.000000    11.000000
50%    169707.000000    12.000000
75%    170251.000000    16.000000
max    171617.000000    57.000000

=> These are the info for a sample of the columns:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3225 entries, 0 to 3224
Data columns (total 7 columns):
Sale_month    3225 non-null int64
Latitude      3225 non-null float64
Longitude     3225 non-null float64
Easting       3225 non-null int64
Northing      3225 non-null int64
Ward          3225 non-null object
Altitude      3225 non-null int64
dtypes: float64(2), int64(4), object(1)
memory usage: 176.4+ KB
None
```

From the statistics we deduce that there are 3225 rows and 18 columns. The statistics show all the basic feature of the data with complete information on spatial co-ordinates.

Step 2: - This step involved plotting the data and exploring it visually. This is done in the code snippet as follows: -

```
sample_df = mer[['Sale_price', 'Bedrooms','Stationdist_miles']]
plt.figure(figsize=(15, 3))

#function for plotting stripplots given a dataframe
def stripplot_these(df):
    for idx, name in enumerate(df.columns):
        n = idx + 1
        plt.subplot(1,3,n)
        sns.stripplot(x=name, data=df, jitter=0.15, orient= 'v', alpha=.4)
    plt.tight_layout()
    plt.show()

stripplot_these(sample_df)
print("\nA strip-plot shows how the data is spread out in relation to it’s own group.\n")
```




```
A strip-plot shows how the data is spread out in relation to it’s own group.
```

From the above, it is concluded that the sale price seems to have good distribution although there may be some skewness to it. The ‘distance to station’ plot indicates that this data has few outliers.

Step 3: - In this final step we establish correlation between variables. Correlation is one of the most widely used statistical concepts. Correlation can help in predicting one quantity from other, it can also indicate the presence of a causal relationship [14].

The following snippet of code selects the numerical variables present in the dataset.

```
#all numerical columns
mer_notobject=mer.select_dtypes( exclude=['object']).copy()
mer_notobject.head(5)
```

	Sale_price	Bedrooms	Stationdist_miles	Sale_year	Sale_month	Latitude	Longitude	Easting	Northing	Altitude
0	435000	2	0.4	2015	10	51.412605	-0.307355	517812	169535	9
1	1295000	4	0.4	2016	2	51.412605	-0.307355	517812	169535	9
2	550000	2	0.4	2017	10	51.412605	-0.307355	517812	169535	9
3	550000	5	0.4	2018	4	51.412605	-0.307355	517812	169535	9
4	143750	2	0.4	2016	10	51.412605	-0.307355	517812	169535	9

In this early stage we are checking the correlation between variables in the dataset. Correlation between target sale price and other predictor variables is obtained in the following code snippet.

```
corr = data_numerical.corr()
print (corr['Sale_price'].sort_values(ascending=False)[:7], '\n')
```

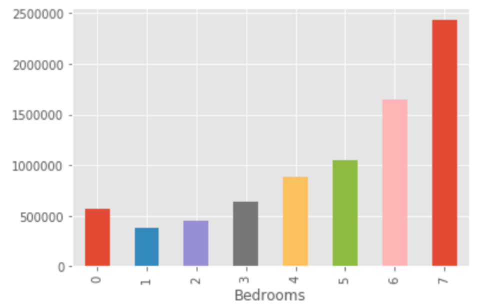
```
Sale_price           1.000000
Bedrooms             0.486910
Altitude             0.316755
Longitude            0.299323
Easting              0.293919
Northing             0.250086
Stationdist_miles    0.244641
Name: Sale_price, dtype: float64
```

The correlation statistics show that there are few variables that are correlated to each other, however we do not observe that they are highly correlated that could lead to multicollinearity problem.

Following graphical plot verifies the correlation observed

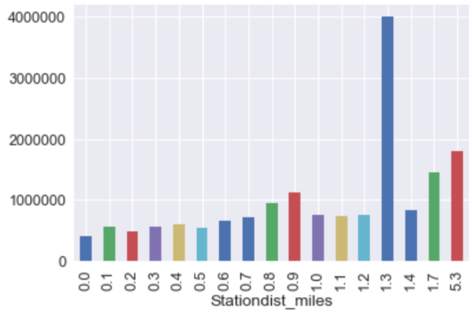
```
mer.groupby('Bedrooms')['Sale_price'].mean().plot.bar()
```

<matplotlib.axes._subplots.AxesSubplot at 0x27c0f8bd4a8>



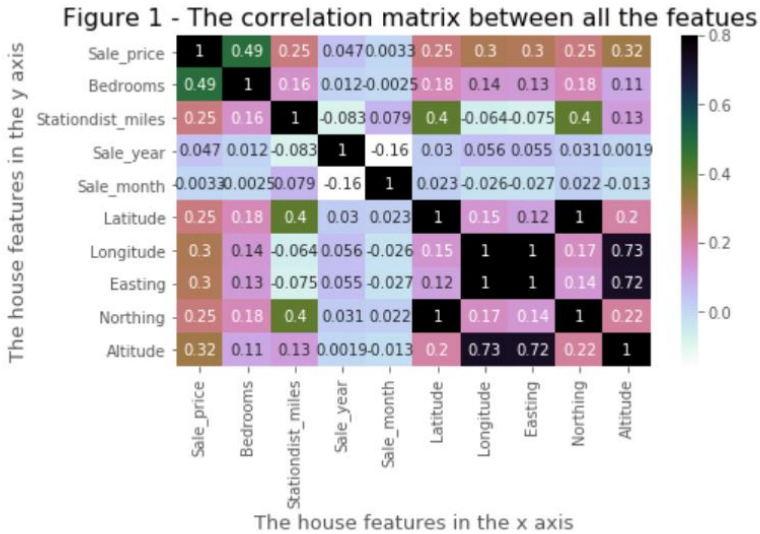

```
mer.groupby('Stationdist_miles')['Sale_price'].mean().plot.bar()
```

<matplotlib.axes._subplots.AxesSubplot at 0x27c33943f60>



Finally, a correlation matrix was constructed to conclude the data exploration. This is shown in code snippet as follows

```
# check for any correlations between variables,CORRELATION MATRIX
# corr = mer.corr()
# sns.heatmap(corr,vmax=.8,square=True)


plt.figure(figsize=(7,4))
columns = ['Sale_price','Bedrooms','Stationdist_miles','Sale_year','Sale_month',
           'Latitude','Longitude','Easting','Northing','Altitude']
sns.heatmap(mer[columns].corr(),annot=True,cmap='cubehelix_r',vmax=0.8)
plt.xlabel('The house features in the x axis',fontsize= 13)
plt.ylabel('The house features in the y axis',fontsize= 13)
plt.title('Figure 1 - The correlation matrix between all the featues ', fontsize= 16);
#draws  heatmap with input as the correlation matrix calculted by(iris.corr())
plt.show()
```

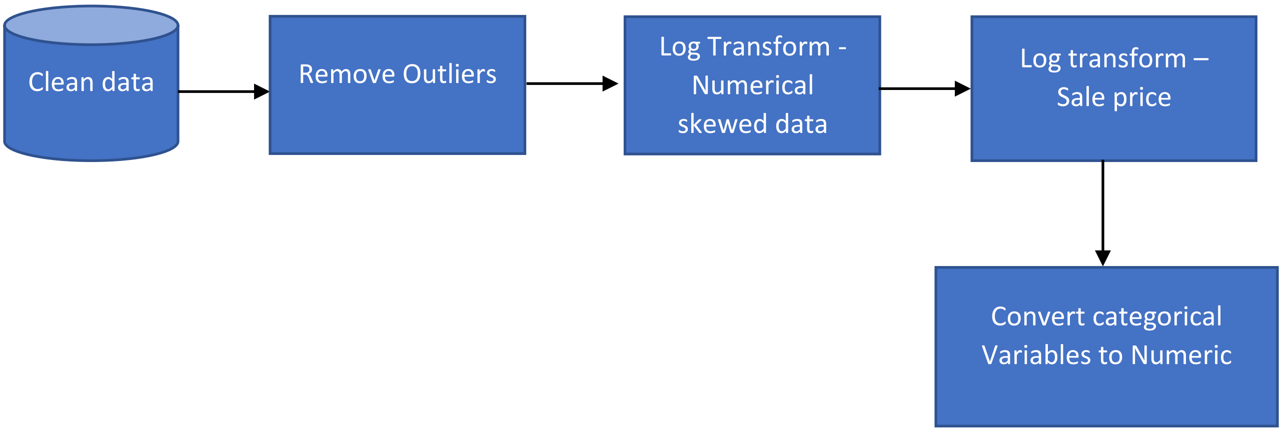


Above figure shows correlation matrix heatmap with Python Seaborn

Thus, after an overview on the data by visualisation and statistical analysis, we have a better insight of data.

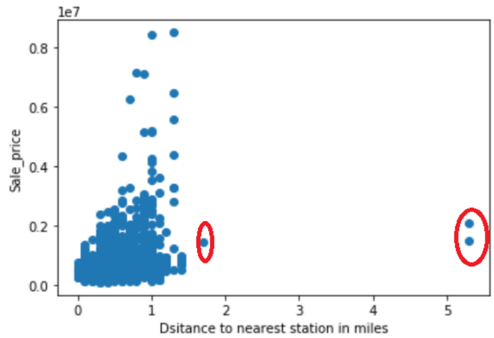
Data Pre-Processing

This stage involves dealing with the outlier values, encode variables and take initiative to remove any unreliability in the data set. The flowchart gives an overview of the data pre-processing stages.



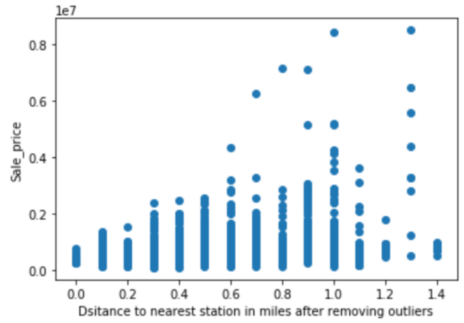
As a first step, we aim to find the outlier in the dataset. Outliers are observations that are far from most of the remainder of data values. Following code plots the Sale_Price verses 'Distance to Nearest Station'

```
#Looking for outliers in station distance
plt.scatter(x=data['Stationdist_miles'], y=data['Sale_price'])
plt.ylabel('Sale_price')
plt.xlabel('Dsitance to nearest station in miles')
plt.show()
print("\nThere are larger values(outliers) found for 'Distance to nearest station in miles'.\n")
```



The red circle on the data points are considered outliers. The logical assumption is that, a 'distance to nearest station' when greater than 1.5 miles is unlikely to influence a purchasing decision. These are not incorrect values but are far outside the range of data we want to consider in our analysis. Thus, they are deleted by the following code, also shown is the plot with outlier removed.

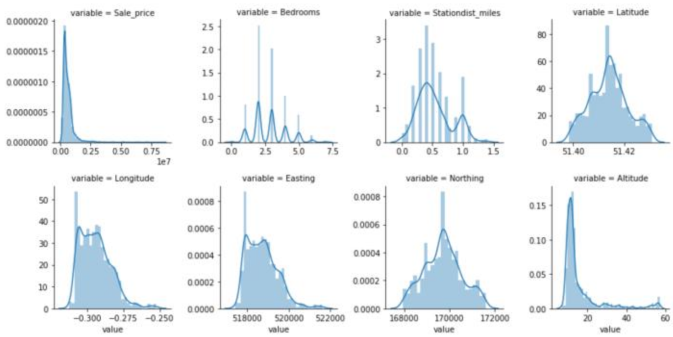
```
#Deleting outliers
data= data[data['Stationdist_miles'] < 1.5]

#check the graph again
plt.scatter(x=data['Stationdist_miles'], y=data['Sale_price'])
plt.ylabel('Sale_price')
plt.xlabel('Dsitance to nearest station in miles after removing outliers')
plt.show()
```



Predictor variables are divided into two categories: - numerical and categorical.

Since the data can have skewed distribution, as a next step skewness in the data is checked. This is done in the following code snippet.

```
import warnings
warnings.filterwarnings("ignore")
##ignore warning (from sklearn and seaborn)

#create numeric plots
num = [f for f in data.columns if data.dtypes[f] != 'object']
nd = pd.melt(data, value_vars = num)
n1 = sns.FacetGrid (nd, col='variable', col_wrap=4, sharex=False, sharey = False)
n1 = n1.map(sns.distplot, 'value')
n1
print("\nSince some of the variables are right skewed, need to transform them later.\n")
```



The above graphs show that the data is rightly skewed.

The table in the following code snippet shows that Altitude, Longitude and Easting shows skewness greater than 0.75 and they need to be log transformed.

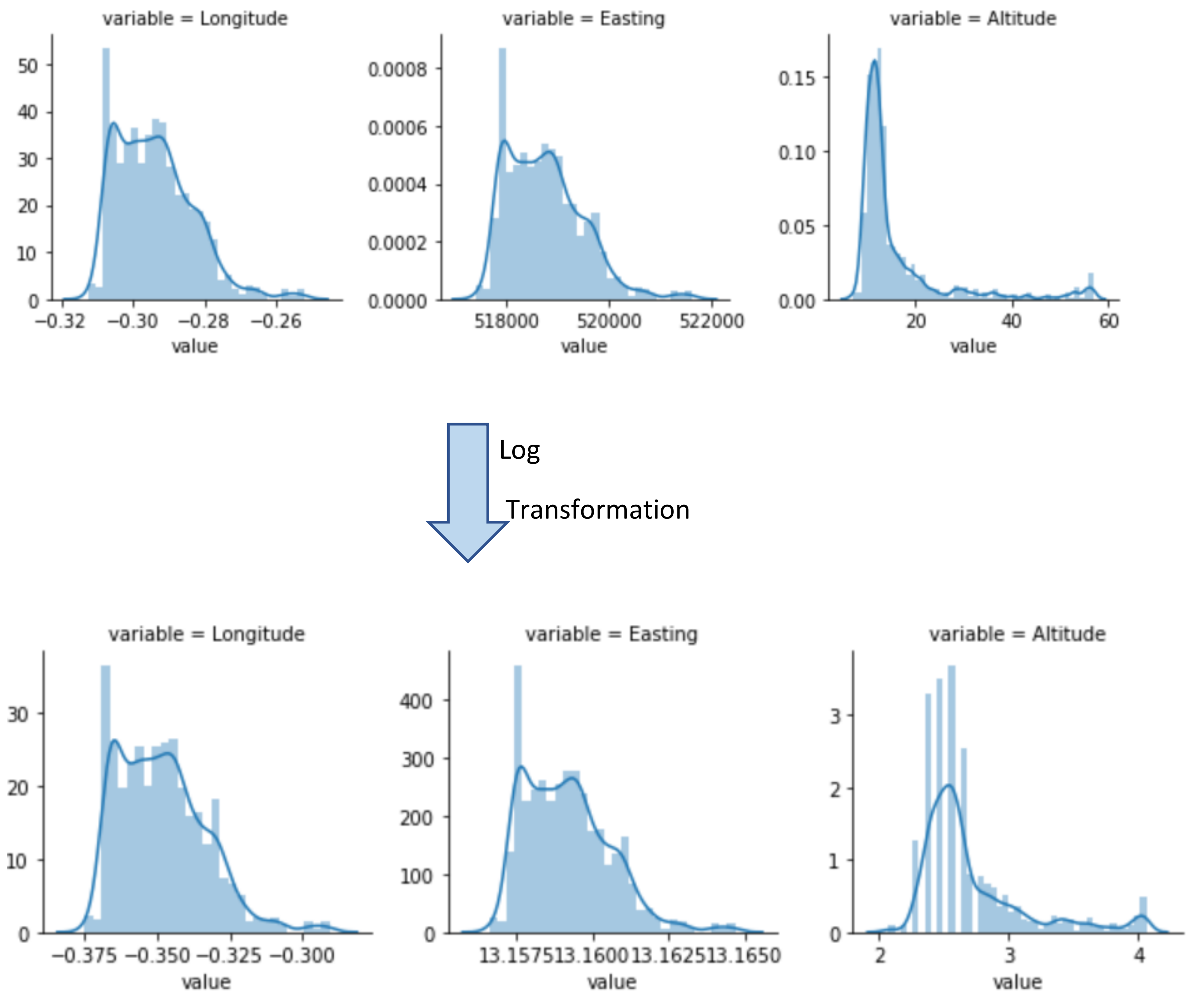
```python
#get numeric features
numeric_feats = [f for f in data.columns if data.dtypes[f] != object]
numeric_feats.remove('Sale_price')


# Check the skew of all numerical features
skewed_feats = data[numeric_feats].apply(lambda x: skew(x.dropna())).sort_values(ascending=False)
print("\nSkew in numerical features: \n")
skewness = pd.DataFrame({'Skew' :skewed_feats})
skewness.head(10)
```

```
Skew in numerical features:
```

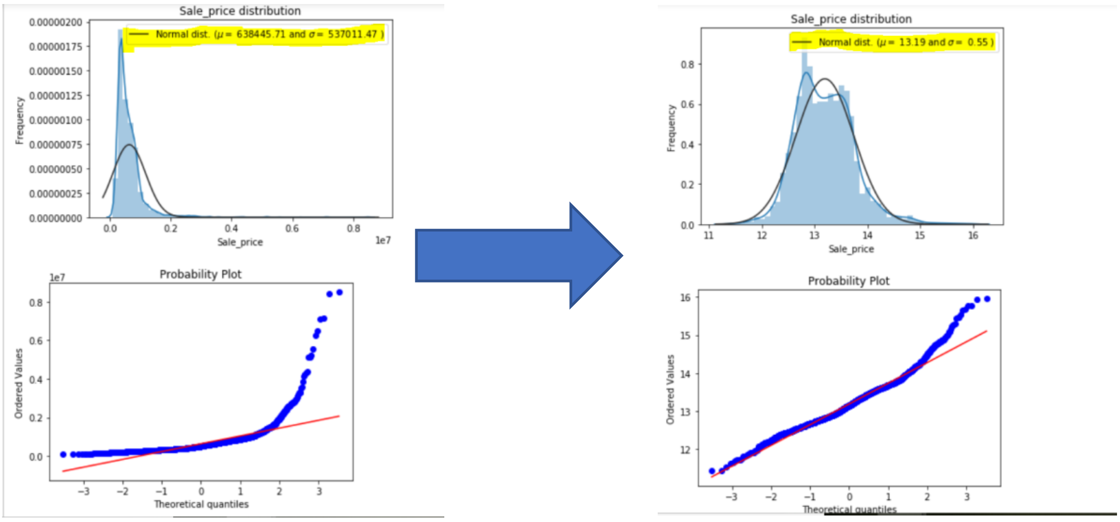
	Skew
Altitude	2.698353
Longitude	0.837923
Easting	0.829234
Bedrooms	0.649918
Stationdist_miles	0.635153
Latitude	0.121230
Northing	0.100379

The code snippet shows the log transformation data. The pre – transformation plots are displayed on top and post – transformation on the to aid us to compare the distribution before and after.

```python
#Log transform skewed numeric features:
data['Longitude'] = np.log1p(data['Longitude'])
data['Easting'] = np.log1p(data['Easting'])
data['Altitude'] = np.log1p(data['Altitude'])
```

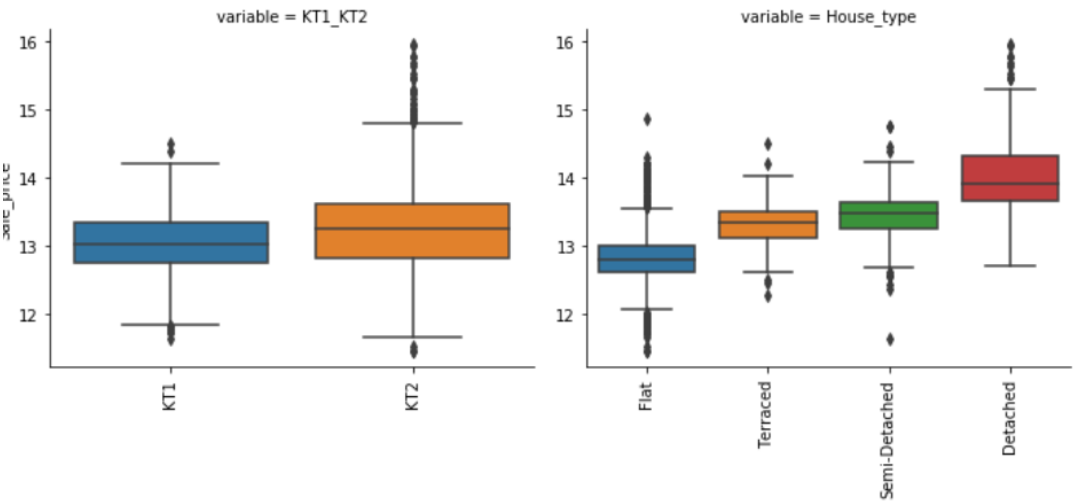


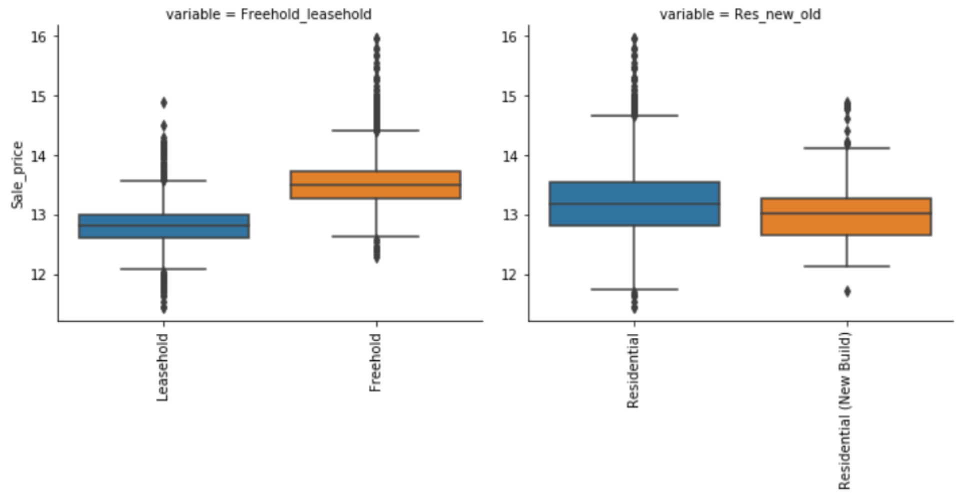
Similar steps are taken for the Sale Price distribution and QQ plot is obtained as well.

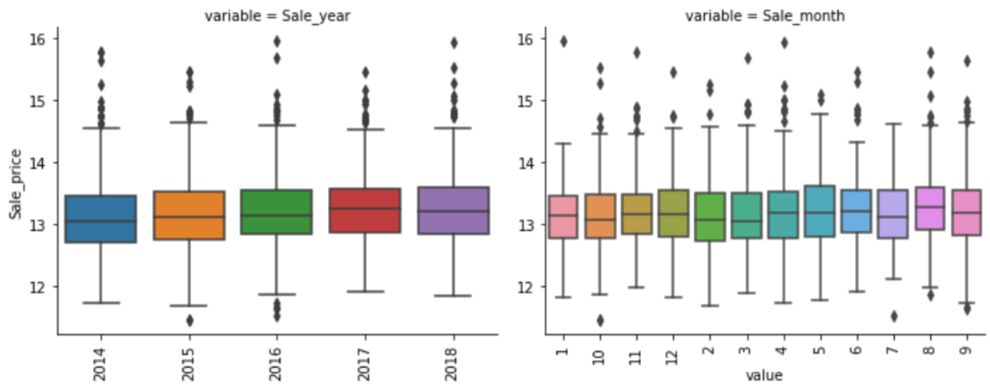


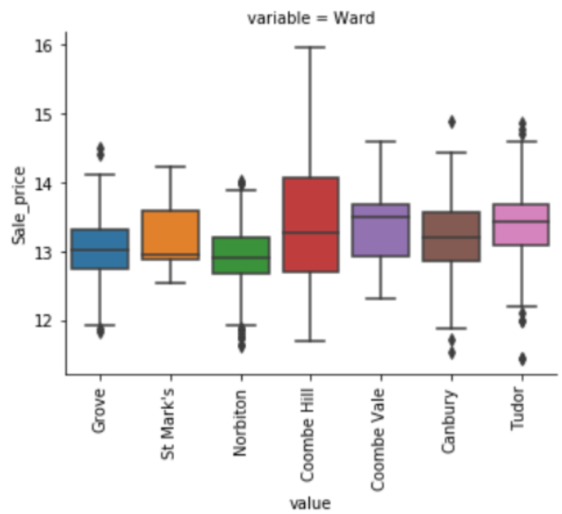
As a next step, the box plots are generated for categorical variables against the sale price. The code and the graphs are plotted here as under: -

```
def boxplot(x,y,**kwargs):
            sns.boxplot(x=x,y=y)
            x = plt.xticks(rotation=90)

cat = [f for f in data.columns if data.dtypes[f] == 'object']

p = pd.melt(data, id_vars='Sale_price', value_vars=cat)
g = sns.FacetGrid (p, col='variable', col_wrap=2, sharex=False, sharey=False, size=5)
g = g.map(boxplot, 'value','Sale_price')
g
```




variable = Freehold_leasehold
variable = Res_new_old
Sale_price
16
15
14
13
12
Leasehold
Freehold
Residential
Residential (New Build)


variable = Sale_year
variable = Sale_month
Sale_price
16
15
14
13
12
2014
2015
2016
2017
2018
1
10
11
12
2
3
4
5
6
7
8
9
value


variable = Ward
Sale_price
16
15
14
13
12
Grove
St Mark's
Norbiton
Coombe Hill
Coombe Vale
Canbury
Tudor
value

We finally apply one hot encoding to convert the process variables in a form that makes it easier for ML algorithms to do better job in prediction.

```
# Get one hot encoding of columns KT1_KT2
one_hot = pd.get_dummies(data['KT1_KT2'])
# Drop column KT1_KT2 as it is now encoded
data = data.drop('KT1_KT2',axis = 1)
# Join the encoded data
data = data.join(one_hot)

# Get one hot encoding of columns House_type
one_hot = pd.get_dummies(data['House_type'])
# Drop column House_type as it is now encoded
data = data.drop('House_type',axis = 1)
# Join the encoded data
data = data.join(one_hot)


# Get one hot encoding of columns Freehold_leasehold
one_hot = pd.get_dummies(data['Freehold_leasehold'])
# Drop column Freehold_leasehold as it is now encoded
data = data.drop('Freehold_leasehold',axis = 1)
# Join the encoded data
data = data.join(one_hot)

# Get one hot encoding of columns Res_new_old
one_hot = pd.get_dummies(data['Res_new_old'])
# Drop column Res_new_old as it is now encoded
data = data.drop('Res_new_old',axis = 1)
# Join the encoded data
data = data.join(one_hot)

# Get one hot encoding of columns Ward
one_hot = pd.get_dummies(data['Ward'],prefix="Ward" + "_")
# Drop column Ward as it is now encoded
data = data.drop('Ward',axis = 1)
# Join the encoded data
data = data.join(one_hot)


# Get one hot encoding of columns Sale_year
one_hot = pd.get_dummies(data['Sale_year'],prefix="Sale_year" + "_")
# Drop column Sale_year as it is now encoded
data = data.drop('Sale_year',axis = 1)
# Join the encoded data
data = data.join(one_hot)


# Get one hot encoding of columns Sale_month
one_hot = pd.get_dummies(data['Sale_month'],prefix="Sale_month" + "_")
# Drop column Sale_month as it is now encoded
data = data.drop('Sale_month',axis = 1)
# Join the encoded data
data = data.join(one_hot)
```

The following snippet shows the effect of creating dummy categorical variables

```
#all numerical columns
data_numerical=data.select_dtypes( include=[np.number])
data_numerical.dtypes
```

```
Sale_price                   float64
Bedrooms                       int64
Stationdist_miles            float64
Latitude                     float64
Longitude                    float64
Easting                      float64
Northing                       int64
Altitude                     float64
KT1                            uint8
KT2                            uint8
Detached                       uint8
Flat                           uint8
Semi-Detached                  uint8
Terraced                       uint8
Freehold                       uint8
Leasehold                      uint8
Residential                    uint8
Residential (New Build)        uint8
Ward__Canbury                  uint8
Ward__Coombe Hill              uint8
Ward__Coombe Vale              uint8
Ward__Grove                    uint8
Ward__Norbiton                 uint8
Ward__St Mark's                uint8
Ward__Tudor                    uint8
Sale_year__2014                uint8
Sale_year__2015                uint8
Sale_year__2016                uint8
Sale_year__2017                uint8
Sale_year__2018                uint8
Sale_month__1                  uint8
Sale_month__10                 uint8
Sale_month__11                 uint8
Sale_month__12                 uint8
Sale_month__2                  uint8
Sale_month__3                  uint8
Sale_month__4                  uint8
Sale_month__5                  uint8
Sale_month__6                  uint8
Sale_month__7                  uint8
Sale_month__8                  uint8
Sale_month__9                  uint8
dtype: object
```

Inspecting the data correlation now reveals better values

```
corr = data_numerical.corr()

print (corr['Sale_price'].sort_values(ascending=False)[:15], '\n')
print (corr['Sale_price'].sort_values(ascending=False)[-5:])
```

```
Sale_price           1.000000
Freehold             0.658947
Bedrooms             0.575888
Detached             0.547382
Northing             0.292371
Latitude             0.289425
Semi-Detached        0.269526
Stationdist_miles    0.250547
KT2                  0.195176
Longitude            0.182910
Easting              0.178581
Ward__Tudor          0.162183
Ward__Coombe Hill    0.145512
Altitude             0.124639
Terraced             0.101030
Name: Sale_price, dtype: float64

Ward__Grove       -0.134623
Ward__Norbiton    -0.193454
KT1               -0.195176
Flat              -0.652769
Leasehold         -0.658947
Name: Sale_price, dtype: float64
```

Data Modelling

Machine learning is the study of algorithms that can learn from data and make predictions, by building a model from example inputs rather than following static instructions [24]. In supervised learning, the system is presented with example inputs and outputs, with the aim of producing a function that maps the inputs to outputs [25].

Regression is a subset of supervised learning, where the outputs are continuous. The problem of predicting future housing prices is considered a regression problem, since we are concerned with predicting values that fall within a continuous range of outputs. Through regression, we will be able to explore the relationship between the independent variables we have selected (Bedrooms, distance to nearest station, post code location coordinates) and the property price

A Machine Learning algorithm needs to be trained on a set of data to learn the relationships between different features and how these features affect the target variable. For this we need to divide the entire data set into two sets. One is the training set on which we are going to train our algorithm to build a model. The other is the testing set on which we will test our model to see how accurate its predictions are [26].

Our dataset should be as large as possible to train the model and removing considerable part of it for validation poses a problem of losing valuable portion of data that we would prefer to be able to train [28]. In order to address this issue, we used the K-fold Cross validation technique. In this technique, the dataset is divided into k subsets and model is trained on k-1 subsets and the last subset is held for test. This process is repeated k times, such that each time, one of the k subsets is used as the test set/ validation set and the other k-1 subsets are put together to form a training set [28].

Following code snippet show this: -

```
y = data_numerical.Sale_price#transforms the y variable for the model
X = data_numerical.drop(['Sale_price'], axis=1)
# X denotes the set of predictor data, and y is the target variable.
```

```
#partitioning the data and to start modeling

X_train, X_test, y_train, y_test = train_test_split(
                                    X, y, random_state=42, test_size=.2)

#view number of training and testing data
print('Training prediction variable contains :',len(y_train) ,'rows')
print('Training independent variable contains :',len(X_train) ,'rows')
print('Testing prediction variable contains :',len(y_test) ,'rows')
print('Testing independent variable contains :',len(X_test) ,'rows')
```

```
Training prediction variable contains : 2577 rows
Training independent variable contains : 2577 rows
Testing prediction variable contains : 645 rows
Testing independent variable contains : 645 rows
```

As shown in the code snippet, the test data size is chosen to be 20%, leaving a sufficiently large i.e. 80% of data as the training data. An investigation on the length of each variable set reveals total number of data rows for both training and test data. The Sale Price is dependant variable and is assigned Y-axis. The remaining features form X-axis.

Following prediction model building steps are applicable to all the methods: -

- Once the model is fitted on the training data, we calculate the R^2 score and RMSE is obtained.
- Cross-validation is done to acquire the cross-validation score.
- Finally, for the benchmark model we obtain the ten most important features in house price prediction along with the residual plot.

Linear Regression

In the initial stage we use linear regression which is basic predictive analysis. Linear regression applies a linear approach to model the relationship between scalar response (or dependent variable) with one or more explanatory variables (or independent variables) [16].

```
#linear model
lr = linear_model.LinearRegression()
model = lr.fit(X_train, y_train)
```

```
print ("R^2 for test is: \n", model.score(X_test, y_test))
```

```
R^2 for test is:
 0.6561406455275824
```

The r-squared value is a measure of how close the data are to the fitted regression line. It takes a value between 0 and 1, 1 meaning that all of the variance in the target is explained by the data. In general, a higher r-squared value means a better fit.

```
predictions = model.predict(X_test)
```

In the next step, the results are cross validated, the following snippet shows this

```
from sklearn.model_selection import KFold
from sklearn import model_selection
kfold = KFold(n_splits=5, random_state=42)
modelcv = linear_model.LinearRegression()
results = model_selection.cross_val_score(modelcv, X, y, cv=kfold)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100.0))
```

```
Accuracy: 52.341% (11.720%)
```

Advanced regression techniques

While the linear regression model is too simple to accurately build a prediction model, there are many fundamental concepts in linear regression that many other regression techniques build upon.

Further to improve the accuracy and reduce error, advance regression techniques are used to form the prediction model.

1. Ridge Regression

Ridge Regression is a technique used when the data suffers from multicollinearity (independent variables are highly correlated) [17]. In our data set the independent variables are correlated. The number of bedrooms, the location of property and the distance to station are correlated with each other. In such a scenario if the least square estimates are unbiased, their variances are large that results in greater difference between observed values and true values. By adding a degree if bias to the regression estimates, ridge regression reduces the standard errors.

Following code snippet shows the application of Ridge regressor.

```
# Append classifier to preprocessing pipeline.
# Now we have a full prediction pipeline.
pipeline = Pipeline(steps=[('regressor', Ridge())])


pipeline.fit(X_train, y_train)
print("model R2 score: %.3f" % pipeline.score(X_test, y_test))
print("RMSE: %.3f" % np.sqrt(mean_squared_error(y_test,pipeline.predict(X_test))))

kfold =10
scoring = ['r2']
results = cross_validate(pipeline, X, y, cv=kfold, return_train_score=True,scoring=scoring)
#outputs the scores
print('test scores: {}'.format(results['test_r2']))
print("\nAverage test Score: {}".format(np.mean(results['test_r2'])))
```

```
model R2 score: 0.636
RMSE: 0.328
test scores: [-0.00371055  0.57666232  0.38334012  0.60623163  0.25952517  0.73609695
  0.5208146   0.33115052  0.77015545  0.68027336]

Average test Score: 0.4860539570696779
```

```
kfold =10
model_ridge=Ridge()
results = model_selection.cross_val_score(model_ridge, X_test, y_test, cv=kfold)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100.0))
```

```
Accuracy: 61.110% (9.361%)
```

2.Kernel Ridge

Kernel ridge regression combines Ridge Regression (linear least squares with l2-norm regularization) with the kernel trick. It thus learns a linear function in the space induced by the respective kernel and the data. For non-linear kernels, this corresponds to a non-linear function in the original space. The form of the model learned by Kernel Ridge is identical to support vector regression (SVR). Kernel Ridge can be done in closed form and is typically faster for medium sized dataset [27].

The following code snippet applies this: -

```
pipeline = Pipeline(steps=[('regressor', KernelRidge())])


pipeline.fit(X_train, y_train)
print("model R2 score: %.3f" % pipeline.score(X_test, y_test))
print("RMSE: %.3f" % np.sqrt(mean_squared_error(y_test,pipeline.predict(X_test))))

kfold = 5
scoring = ['r2','neg_mean_squared_error']
results = cross_validate(pipeline, X, y, cv=kfold, return_train_score=True,scoring=scoring)
#outputs the scores
print('Test scores: {}'.format(results['test_r2']))
print("\nAverage test Score: {}".format(np.mean(results['test_r2'])))
```

```
model R2 score: 0.636
RMSE: 0.328
Test scores: [0.20958947 0.46291217 0.45180098 0.44041692 0.54135682]

Average test Score: 0.42121527355353117
```

```
kfold =10
model_kr=KernelRidge()
results = model_selection.cross_val_score(model_kr, X, y, cv=kfold)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100.0))
```

```
Accuracy: 48.624% (22.989%)
```

3.Lasso

Lasso (Least Absolute Shrinkage and Selection Operator) is similar to Ridge Regression. In addition, it can reduce the variability and improving the accuracy of linear regression models [17].

The following code snippet shows the application of Lasso regression.

```
pipeline = Pipeline(steps=[('regressor', Lasso(alpha=0.1,max_iter=1000,tol=0.0001))])


pipeline.fit(X_train, y_train)
print("model R2 score: %.3f" % pipeline.score(X_test, y_test))
print("RMSE: %.3f" % np.sqrt(mean_squared_error(y_test,pipeline.predict(X_test))))

kfold = 10
scoring = ['r2','neg_mean_squared_error']
results = cross_validate(pipeline, X, y, cv=kfold, return_train_score=True,scoring=scoring)
#outputs the scores
print('Test scores: {}'.format(results['test_r2']))
print("\nAverage test Score: {}".format(np.mean(results['test_r2'])))
```

```
model R2 score: 0.348
RMSE: 0.440
Test scores: [0.0723399  0.32611092 0.06093502 0.39443225 0.30943712 0.46925032
 0.41912268 0.20361388 0.39204855 0.32989634]

Average test Score: 0.2977186964674094
```

```
kfold =10
model_lasso=Lasso(alpha=0.1,max_iter=1000,tol=0.0001)
results = model_selection.cross_val_score(model_lasso, X, y, cv=kfold)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100.0))
```

```
Accuracy: 29.772% (13.421%)
```

When the group of predictors are highly correlated, lasso picks only one of them and shrinks the other to zero. This is likely be the reason of low scores of as shown in the snippet above.

4. Random Forest Regressor

Random Forest is a trademark term for an ensemble of decision trees. To classify a new object based on attributes, each tree gives a classification as if the tree votes for that class. The forest chooses the classification having the most votes (over all the trees in the forest) [18].

The advantage of Random Forest Regressor is that there is no risk of overfitting. The main limitation of Random Forests is that due to large number of trees, the algorithms can be slower [19].

The following code snippet show this

```
pipeline = Pipeline(steps=[('regressor', RandomForestRegressor())])


pipeline.fit(X_train, y_train)
print("model R2 score: %.3f" % pipeline.score(X_test, y_test))
print("RMSE: %.3f" % np.sqrt(mean_squared_error(y_test,pipeline.predict(X_test))))

kfold = 10
scoring = ['r2','neg_mean_squared_error']
results = cross_validate(pipeline, X, y, cv=kfold, return_train_score=True,scoring=scoring)
#outputs the scores
print('Test scores: {}'.format(results['test_r2']))
print("\nAverage Test Score: {}".format(np.mean(results['test_r2'])))
```

```
model R2 score: 0.783
RMSE: 0.253
Test scores: [0.30098181 0.76585823 0.38609035 0.46728297 0.42429653 0.73747012
 0.62332003 0.66726649 0.79818451 0.72943212]

Average Test Score: 0.5900183153862759
```

```
kfold =10
model_rfr=RandomForestRegressor()
results = model_selection.cross_val_score(model_rfr, X, y, cv=kfold)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100.0))
```

```
Accuracy: 60.756% (17.813%)
```

5.XGB Regressor

XGBoost is an advanced implementation of the gradient boosting algorithm [20]. Gradient boosting is a machine learning technique for regression and classification problems that produces a prediction model in the form of an ensemble of weak prediction models [15].

XGBoost is highly effective Machine Learning algorithm. It includes variety of regularisation what reduces overfitting and improves overall performance. It implements parallel processing and hence is faster. It is also possible to define custom optimization objectives and evaluation criteria.

The following code snippet applies the XGBoost Regressor: -

```
pipeline = Pipeline(steps=[('regressor', xgb.XGBRegressor(colsample_bytree=0.4603, gamma=0.0468,
                             learning_rate=0.05, max_depth=3,
                             min_child_weight=1.7817, n_estimators=2200,
                             reg_alpha=0.4640, reg_lambda=0.8571,
                             subsample=0.5213, silent=1,
                             random_state =7, nthread = -1))])


pipeline.fit(X_train, y_train)
print("model R2 score: %.3f" % pipeline.score(X_test, y_test))
print("RMSE: %.3f" % np.sqrt(mean_squared_error(y_test,pipeline.predict(X_test))))

kfold = 10
scoring = ['r2','neg_mean_squared_error']
results = cross_validate(pipeline, X, y, cv=kfold, return_train_score=True,scoring=scoring)
#outputs the scores
print('Test scores: {}'.format(results['test_r2']))
print("\nAverage test Score: {}".format(np.mean(results['test_r2'])))
print("\n Average test RMSE:{}".format(np.mean(results['test_neg_mean_squared_error'])))
```

```
model R2 score: 0.808
RMSE: 0.239
Test scores: [0.49476971 0.68406531 0.5736964  0.63619833 0.4539017  0.77348114
 0.63035865 0.69240832 0.81497186 0.68376239]

Average test Score: 0.6437613804471642

 Average test RMSE:-0.09179322013152225
```

```
kfold =10
model_xgb=xgb.XGBRegressor(colsample_bytree=0.4603, gamma=0.0468,
                             learning_rate=0.05, max_depth=3,
                             min_child_weight=1.7817, n_estimators=2200,
                             reg_alpha=0.4640, reg_lambda=0.8571,
                             subsample=0.5213, silent=1,
                             random_state =7, nthread = -1)
results = model_selection.cross_val_score(model_xgb, X, y, cv=kfold)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100.0))
```

```
Accuracy: 64.376% (10.730%)
```

Although the missing values in the dataset had been taken care of in data preparation stage. One of the advantages with XGB Boosting is that it has inbuilt routines to take care of the missing values.

6. Gradient Boosting Regressor

Boosting is a method of converting weak learners into strong learners. This is the basis of Gradient Boosting Regressor. Gradient Boosting produces a prediction model in the form of an ensemble of weak prediction model typically decision trees [15].

In Gradient Boosting a weak learner is taken and at each step of iteration another weak learner is added to increase the performance and build up a strong learner. This reduces the loss of the loss function. The loss represents the error residuals (the difference between actual value and predicted value) and using this loss value the predictions are updated to minimise the residual [22].

At the end, models at each iteration contribute with weights and the set is combined into some overall predictors; thus, boosting converges a sequence of weak learners into a very complex predictor [23].

The following code snippet shows using a pipeline to apply the Gradient Boosting Regressor.

```
pipeline = Pipeline(steps=[('regressor',GradientBoostingRegressor(n_estimators=3000, learning_rate=0.05,
                                   max_depth=4, max_features='sqrt',
                                   min_samples_leaf=15, min_samples_split=10,
                                   loss='huber', random_state =5))])


pipeline.fit(X_train, y_train)
print("model R2 score: %.3f" % pipeline.score(X_test, y_test))
print("RMSE: %.3f" % np.sqrt(mean_squared_error(y_test,pipeline.predict(X_test))))

kfold = 10
scoring = ['r2','neg_mean_squared_error']
results = cross_validate(pipeline, X, y, cv=kfold, return_train_score=True,scoring=scoring)
#outputs the scores
print('Test scores: {}'.format(results['test_r2']))
print("\nAverage test Score: {}".format(np.mean(results['test_r2'])))
print("\n Average test RMSE:{}".format(np.mean(results['test_neg_mean_squared_error'])))
```

```
model R2 score: 0.814
RMSE: 0.235
Test scores: [0.54680755 0.66120592 0.50955376 0.70475865 0.42815561 0.74516543
 0.60321941 0.64517037 0.79758721 0.67273321]

Average test Score: 0.6314357115777384

 Average test RMSE:-0.09453051305267698
```

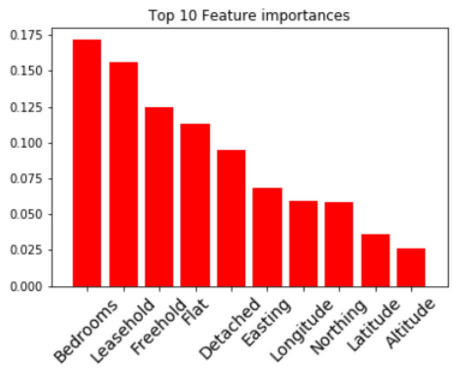
```
kfold =10
model_gbr=GradientBoostingRegressor(n_estimators=3000, learning_rate=0.05,
                                   max_depth=4, max_features='sqrt',
                                   min_samples_leaf=15, min_samples_split=10,
                                   loss='huber', random_state =5)
results = model_selection.cross_val_score(model_gbr, X, y, cv=kfold)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100.0))
```

```
Accuracy: 63.144% (10.622%)
```

The result shows a high R^2 score. Out of the various regression models tried, the model attained by Gradient Boosting Regressor is the best prediction model. This regressor thus fits our requirements and this model is considered as the bench mark model. As such further metrics are evaluated for the same.

```
GBRtuned=GradientBoostingRegressor(n_estimators=3000, learning_rate=0.05,
                                   max_depth=4, max_features='sqrt',
                                   min_samples_leaf=15, min_samples_split=10,
                                   loss='huber', random_state =5)
model=GBRtuned.fit(X_train,y_train)


# Calculate the feature ranking - Top 10
importances = GBRtuned.feature_importances_
indices = np.argsort(importances)[::-1]

print("Top 10 Important Features\n")

for f in range(10):
    print("%d. %s (%f)" % (f + 1, X.columns[indices[f]], importances[indices[f]]))

#Plot the feature importances of the forest
indices=indices[:10]
plt.figure()
plt.title("Top 10 Feature importances")
plt.bar(range(10), importances[indices], color="r", align="center")
plt.xticks(range(10), X.columns[indices], fontsize=14, rotation=45)
plt.xlim([-1, 10])
plt.show()
```

```
Top 10 Important Features

1. Bedrooms (0.171439)
2. Leasehold (0.156086)
3. Freehold (0.124581)
4. Flat (0.113284)
5. Detached (0.094395)
6. Easting (0.068529)
7. Longitude (0.059507)
8. Northing (0.058182)
9. Latitude (0.036275)
10. Altitude (0.026016)
```



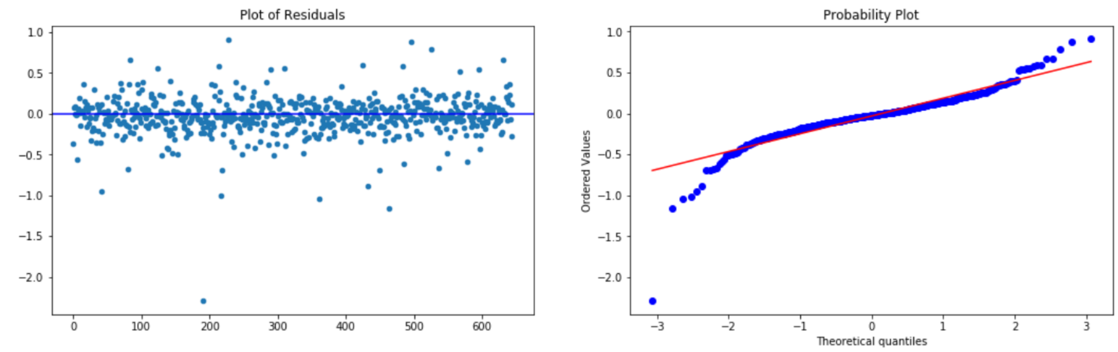
In the above graph we obtain the top 10 important features. It ranks the importance of individual variable based on their relative influence that is a measure indicating relative importance of each variable in training the model.

Residual plotting

As stated earlier the residuals is the difference between actual value and predicted value. With the final model obtained the residuals are plotted to show their effective behaviour. The code snippet and graphs are as under.

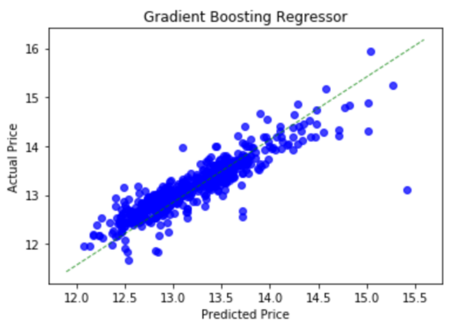
The residuals roughly formed a “horizontal band” around the 0 line, this suggest that the variances of the error terms are equal, furthermore no one residual “stands out” from the basic random pattern of residuals which involve that there are no outliers.

```
#calculate the residuals
gbr_preds=pipeline.predict(X_test)
gbr_preds = pd.DataFrame(gbr_preds)
y_test = y_test.reset_index(drop=True)
residuals = y_test - gbr_preds[0]

#plotting Residual and Probabililty graph
plt.figure(figsize=(18, 5))
plt.subplot(1,2,1)
plt.axhline(0, color="blue")
plt.title('Plot of Residuals')
plt.scatter(residuals.index,residuals, s=20)

plt.subplot(1,2,2)
plt.title('Probability Plot')
stats.probplot(residuals, dist='norm',plot=plt)
plt.show()
```



The following code snippet plots the actual v/s predicted price.

```
actual_values = y_test
plt.scatter(gbr_preds, actual_values, alpha=.75,
            color='b') #alpha helps to show overlapping data
xmin, xmax = plt.xlim()
ymin, ymax = plt.ylim()
plt.plot([xmin, xmax], [ymin, ymax], "g--", lw=1, alpha=0.75)
plt.xlabel('Predicted Price')
plt.ylabel('Actual Price')
plt.title('Predicted prices (GBP) vs. True prices (GBP)')
plt.title('Gradient Boosting Regressor')
plt.show()
```



The graph of predicted price versus actual price show good relation between actual house prices and predicted house prices.

Evaluation metrics

This project uses two different evaluation metrics to test the hypotheses: R square score,

and RMSE.

R square is the goodness of fit of the predictions to actual values (Coefficient of determination). It is the explained variation divided by the total variation.

$$R^2 \equiv 1 - \frac{SS_{\rm res}}{SS_{\rm tot}}.$$

MSE (Mean Square Error is the average of the error. It is the average of the sum of the squares of the difference between the predicted value and true value (Metrics).

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y_i} - Y_i)^2$$

RMSE (Root Mean Square Error) is the square root of the average of all the error. It is simply the square root of the mean square error (Metrics).

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

So, after evaluating various regression models to fit our data set, we draw a Benchmarking table which summarises all the results we have obtained.

Regression methods	**RMSE**	**R^2 score**	**Cross validation score**
Linear Regression	0.101	0.656	52.341%
Ridge	0.328	0.636	48.605%
Kernel Ridge	0.328	0.636	48.624%
Lasso	0.440	0.348	29.772%
Random Forest Regressor	0.254	0.782	60.756%
XGBoosting Regressor	0.238	0.808	64.376%
Gradient Boosting Regressor	0.235	0.814	63.144%

Conclusion

This project helped us to understand the complete process of machine learning/data science modelling. We hope that it helps other people understand machine learning concepts. We explored and compared various regression models before arriving on Gradient Boosting as our benchmark regression. In addition, our models also helped identify which characteristics of housing were most strongly associated with price and could explain most of the price variation. Another finding in this project work is that, our relatively smaller data set led to low score during cross validation.

There was one interesting finding in our work and in that contrary to popular belief, 'the distance to station' did not feature in top 10 important features indicating that it does not influence the price of house.

From our finding, Number of bedroom and lease type of house were most influencing factors in determining price of the house.

We hope that our findings are helpful to potential home buyers and real estate investors. The results from our study can potentially help provide answers to home owners and investors when making decisions such as what housing attributes to consider in generating the highest value of a home.

Future Work

- The effective dataset used in this coursework is around 3200 rows that stretches for last five years. Towards the end part of the project we felt that the dataset must have been bigger. Probably we lookout for data for last 10 years. An option could be to include surrounding area into the 'Area of Study'.

- The analysis can be applied to different regions as well. This coursework focuses on Kingston Upon Thames as our area of study. Some interesting case studies can be constructed. At our early stages of project, we had considered compiling a prediction model on 'Best Top 10 places to live in UK' or 'Best Top 10 happy places in UK'

- The cleaning method, specially 'distance to station' uses an approximation method to compute the distance. The method is only approximate and there is a scope to use an accurate method to establish that.

- The pipeline methods used offers usage of more parameters to improve models. More parameters can be supplied to the pipeline method to improve the model prediction.

- The dataset collected had moderate number of features. Ideally, we needed more features. Apart from the generalised features as considered in this project, there are other factors that affect the house price as well. These features include, proximity to the school, whether there is parking available, if yes than the type of parking, the condition of the house is important as well. A newly refurbished house is likely to attract higher price. Other features include proximity to local transport, pets in the home, clever storage, infestation and untoward incidents, kerb appeal etc.

- We have not checked the dependency of our features. A future work on this topic could check and evaluate dependencies.

- There are quite different metrics that can be derived with the current model. This can indicate a better result.

- Different technique of standardization of features and PCA transformations can be used to improvise the model.

References

[1] 10 reasons to live in Kingston upon Thames available at https://www.mumsnet.com/mnlocal/reasons-to-live-in-kingston-upon-thames

[2] https://www.parsehub.com/intro

[3] Python Data Analysis Library found at https://pandas.pydata.org/. The site was accessed on 18/03/19, during this time the latest release was 0.24.2 dated March 2019.

[4] UK area guides-> Moving to Kingston Upon Thames by property news team dated 6th June 2016, found at https://www.zoopla.co.uk/discover/buying-area-guides/living-in-kingston-upon-thames/#6cUtUiyLs86lzBu6.97

[5] https://www.onedome.com/locations/uk/england/london/moving-to-kingston-upon-thames

[6] Kingston Upon Thames dated 02/03/19 found at https://en.wikipedia.org/wiki/Kingston_upon_Thames

[7] Abdi H & Williams L.J (2010) "Principal components analysis" found at the link

https://onlinelibrary.wiley.com/doi/full/10.1002/wics.101

[8] The home page of matplotlib found at https://matplotlib.org version 3.03 last updated on March 18, 2019.

[9] Michael Waskom – Seaborn: statistical data visualization version 0.9.0 found at https://seaborn.pydata.org/index.html

[10] Description as found on http://www.numpy.org/, accessed on 28/03/19.

[11] Scikit-learn from Wikipedia, accessed at https://en.wikipedia.org/wiki/Scikit-learn dated 19/03/2019.

[12] HM Land Registry, public data sets available at http://landregistry.data.gov.uk/

[13] Data Exploration from Wikipedia accessed at https://en.wikipedia.org/wiki/Data_exploration dated 9/01/2019

[14] Ruslana Dalinina "Introduction to Correlation " dated 31 Jan 2017 found at link https://www.datascience.com/blog/introduction-to-correlation-learn-data-science-tutorials

[15] Gradient boosting from Wikipedia, accessed at https://en.wikipedia.org/wiki/Gradient_boosting dated 22/03/2019

[16] Linear Regression from Wikipedia, accessed at https://en.wikipedia.org/wiki/Linear_regression dated 26/03/2019

[17] Sunil Ray "7 types of Regression Techniques you should know" available at https://www.analyticsvidhya.com/blog/2015/08/comprehensive-guide-regression/ dated 14/08/2015

[18] Sunil Ray "Essentials of Machine learning algorithms dated 09/09/2017 available at https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/

[19] Niklas Donge "The Random Forest Algorithm" dated Feb 22 2018 available at https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd

[20] Aishwarya Singh, "A comprehensive guide to ensemble Learning" dated 18/06/2018 available at https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/

[21] 'Kernel' Ridge Regression' from scikit-learn v0.20.3 at https://scikit-learn.org/stable/modules/kernel_ridge.html

[22] Rohith Gandhi, "Gradient Boosting and XGBoost" dated 06/05/2018 available at https://hackernoon.com/gradient-boosting-and-xgboost-90862daa6c77

[23] Ayoub RMIDI, "Build, develop and Deploy a Machine Learning Model to predict cars price using Gradient Boosting" https://towardsdatascience.com/build-develop-and-deploy-a-machine-learning-model-to-predict-cars-price-using-gradient-boosting-2d4d78fddf09

[24] C. M. Bishop, Pattern Recognition and Machine Learning. Springer, 2006.

[25] K. P. Murphy, Machine Learning: A Probabilistic Perspective. MIT Press, 2012

[26] Satyabrata Pal, " Scikit-learn Tutorial: Machine Learning in Python" dated Nov 15, 2018 available at https://www.dataquest.io/blog/sci-kit-learn-tutorial/.

[27] 1.3 Kernel ridge regression, scikit-learn v0.20.3 available https://scikit-learn.org/stable/modules/kernel_ridge.html

[28] Saranya Mandva, "Cross Validation and HyperParameter Tuning in Python" dated Sep 18 2018 at link. https://medium.com/@mandava807/cross-validation-and-hyperparameter-tuning-in-python-65cfb80ee485