Regression analysis of Auckland property capital valuations

Dataset Building

1.1 Adding Population column

Using koordinates API to create a function that returns population in the area, given arguments latitude and longitude [1.1]. Then, using lambda function, creating a new population column in the main data-frame.

1.2 Adding Deprivation index

First, the deprivation index data was downloaded in xlsx format, and then the necessary columns were taken out of the deprivation index data and merged together with the main data-frame, joining left on "SA1" and right on "SA12018_code" [1.2], "SA12018_code" column is later dropped due to redundancy.

1.3 Adding travel and distance times to CBD

We know that proximity of property to business districts can have impact on its price. Due to unusual geography of Auckland, we cannot rely on "as the crow flies" calculations from property to CBD, since in between there may be a harbour or another terrain feature that vehicles cannot travel through. Therefore, we use Google Maps API to calculate the distance and time it takes to travel to CBD. For the purposes of simplicity, Albert park is taken as the CBD address for our model.
First, we define a function that takes address as an argument and returns travel time and distance to CBD [1.3]. If an error occurs and API does not return necessary values, travel time and distance values are set to infinity. Then the function is applied to the main data-frame.

Dataset cleaning

2.1 Dealing with NaN values

First we check if there are any NaN values present in the dataset. It appears that there are 2 entires with missing values in "Bathrooms" and one with missing value in "Suburbs" columns [2.1].

2.1.1 NaN in "Suburbs"

The empty "Suburbs" column appeared due to property being in the Great Barrier Island. Thus, we check if any other properties are located on an island by looking for "Island" keyword in the "Suburbs" column. We then find that properties on Great Barrier Island have "'Great Barrier Island (Aotea Island)" assigned as their suburb [2.1.1].

2.1.2 NaN in "Bathrooms"

After looking at correlation plot (will be introduced later), we know that number of bathrooms and bedrooms in the house is highly correlated ($r = 0.71$). Thus, if we know number of bedrooms, we can replace missing bathroom values with mean number of bathrooms for a house with the same number of bedrooms [2.1.2].
Since both houses with NaN values have 4 bedroom, we find that average number of bathrooms in houses with 4 bedrooms is 2, thus we replace NaN with 2.

2.2 Checking column types

After using `dtypes` function on the data-frame, we find that all columns have appropriate type except "Land area", which appeared to be an `object` instead of `float64` or `int64`. Thus we inspect unique values of land area and find that some are recorded with units m^2.
Therefore, we define a function which deletes m^2 from its argument. The function is then applied to the data-frame. Afterwards, "Land area" column type is changed to float [2.2].

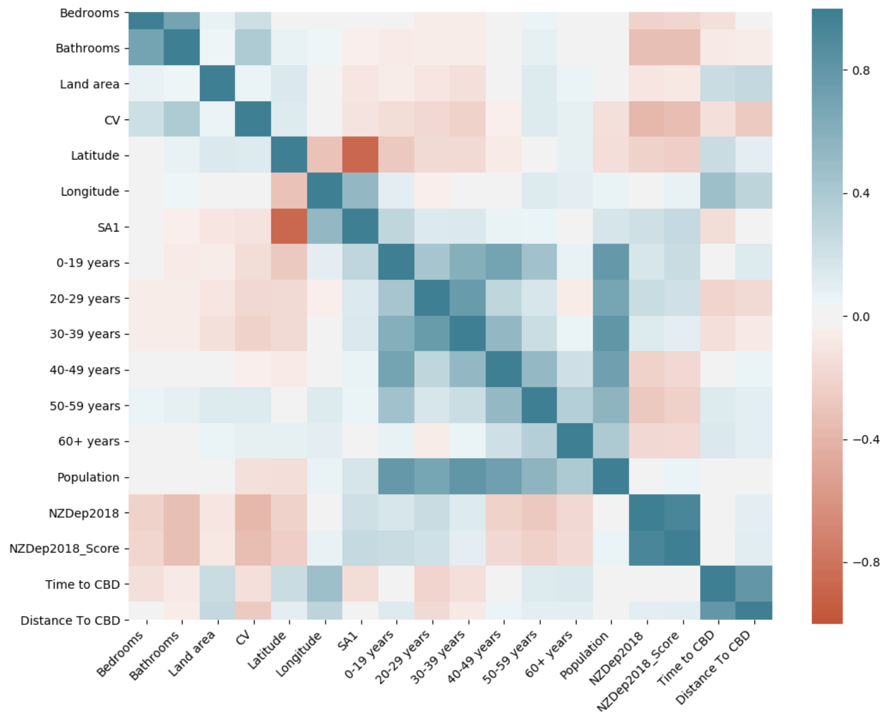
2.3 Dealing with Infinite distances and travel times

We check if any values have infinite travel time/distances to CBD. We find that Google Maps API could not compute travel times for 5 properties located on Rakino and Great Barrier Islands. Given that travel times/distances for these properties will be significantly higher than for other houses in the dataset, we can substitute infinities with rough estimates of travel time/distance to those islands.
From a google search we estimate that travel time to Great barrier is approximately 2h46 (93km) and to (21.7 km) Rakino is 1h02. These estimates are then changed into seconds and meters and added to the data-frame [2.3].

Data Analysis

Some ideas regarding the analysis were taken from this Kaggle Notebook

3.1 Correlation matrix

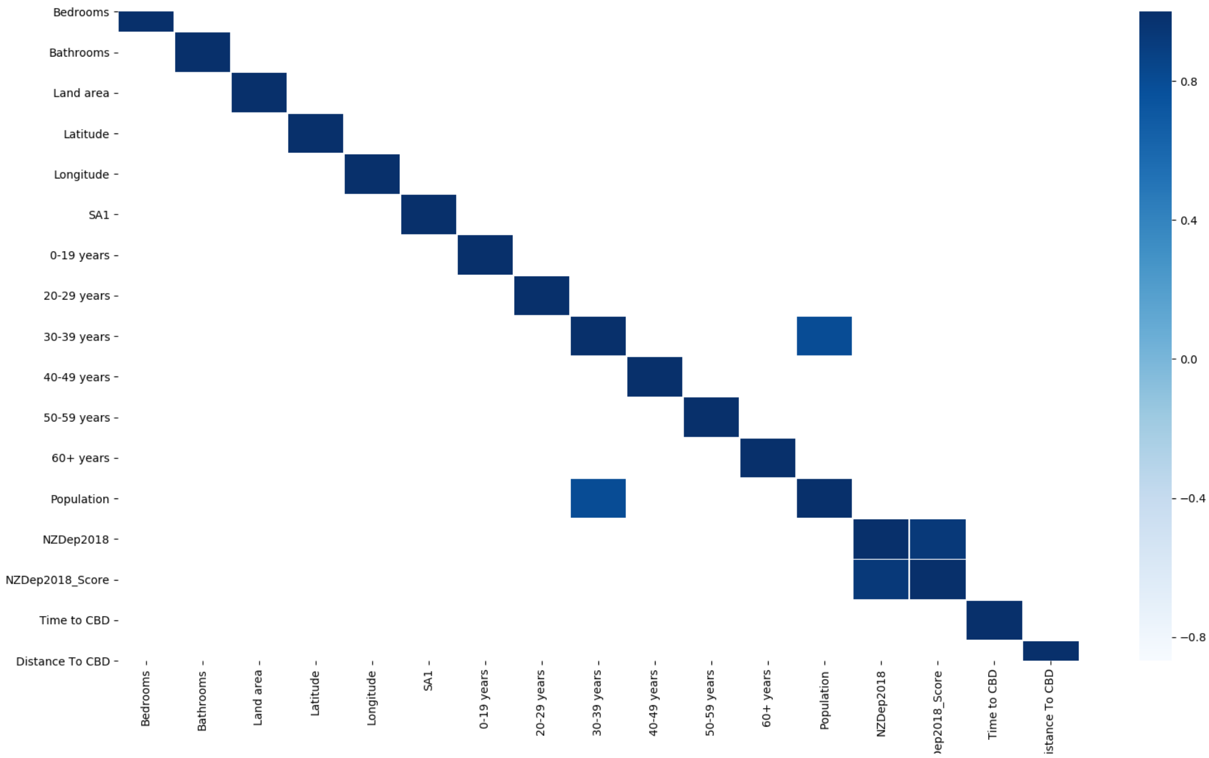


We create correlation matrix [3.1] to see if there are any hidden or valuable correlations in the data. From the matrix it can be seen that CV is not strongly correlated with any of the factors, the highest correlation that it has is about ± 0.4 with positive highest correlation with number of bathrooms and negative highest correlation being deprivation index.
Unsurprisingly, we find strong correlation between population and age groups, deprivation index and deprivation score, and between time and distance to CBD.
We find that SA1 index is correlated with Latitude, which is likely due to SA1 index being assigned with respect to geographical location (e.g. from North to South).

3.2 Addressing Multicollinearity

In order to avoid issues with multicollinearity, which could in turn cause overfitting we remove one of the two variables with $r \geq 0.8$.

From the matrix below [3.2], it can be seen that columns “Population” and “NZ2018_Score”, should be removed.



3.3 Removing redundant features

By finding the correlation between each of the factors and the target (CV), we can identify redundant factors, as they would have no correlation with the target.
Therefore, it can be seen that longitude has no correlation with CV, thus it will be removed [3.3].

	CV
CV	1.000000
Bathrooms	0.385580
Bedrooms	0.246067
50-59 years	0.130483
Latitude	0.120665
60+ years	0.083165
Land area	0.059562
Longitude	0.018066
40-49 years	-0.044717
SA1	-0.110210
Population	-0.125603
Time to CBD	-0.135068
0-19 years	-0.156661
20-29 years	-0.182687
30-39 years	-0.214621
Distance To CBD	-0.263259
NZDep2018_Score	-0.344429
NZDep2018	-0.378027

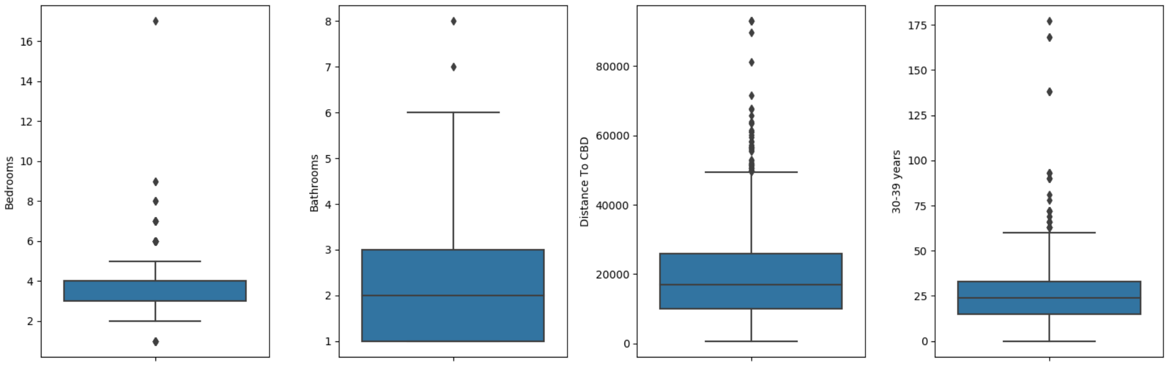
From the table we can observe an interesting pattern, that suburbs with ageing population (50-60+) are positively correlated with CV, however suburbs with younger generations have negative correlation (0-39).

Additionally, we remove “Address” feature since after ‘one-hot encoding’ each address will only contain one datapoint, thus it will have no use for the model.

3.4 Analysing Individual distributions

3.4.1 Removing significant outliers

We now to consider the outliers of the distributions which are most correlated with CV, since they will have the most impact on the model.

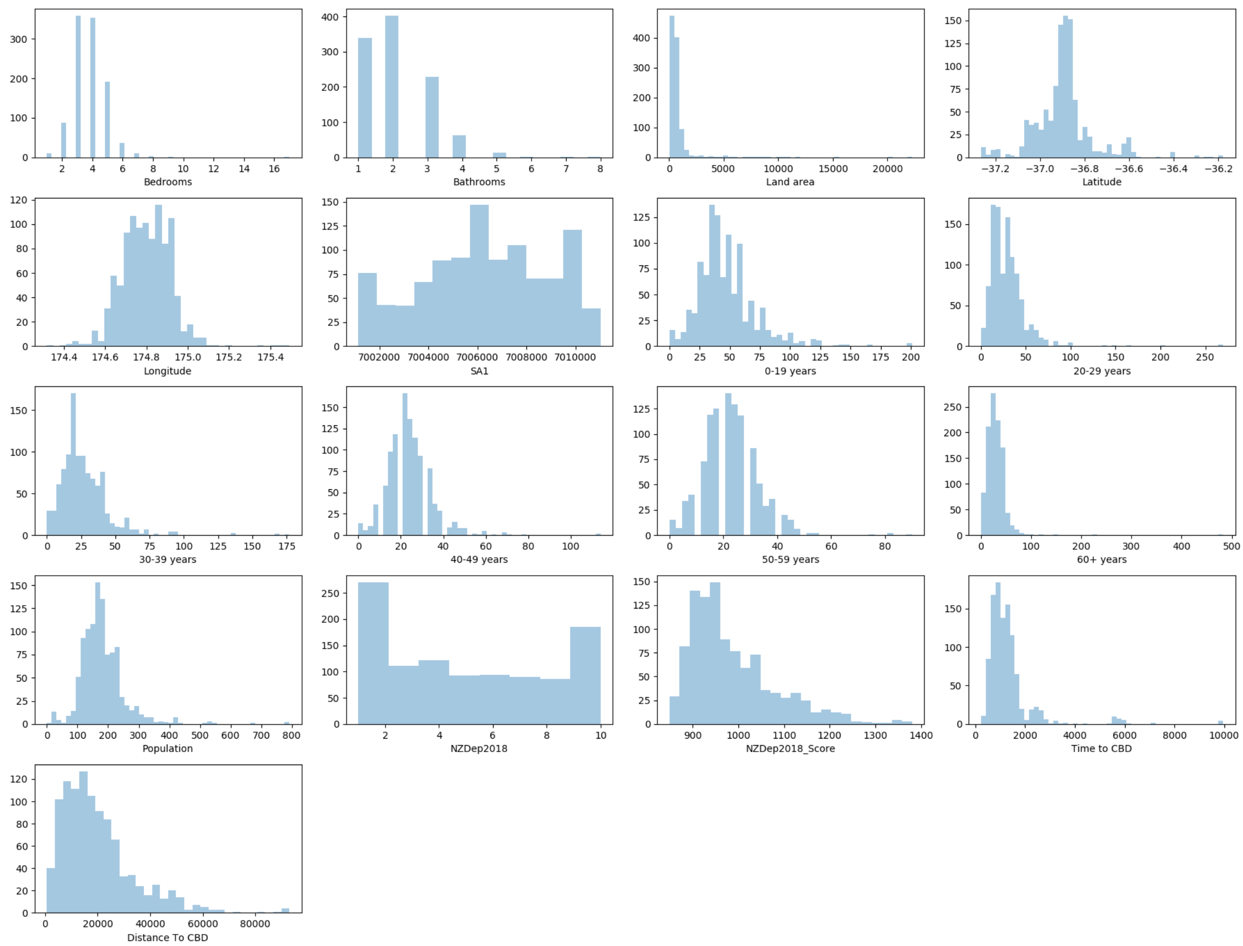


From 3.3 we know that the most significant skewed distributions are: 'Bedrooms','Bathrooms', 'Distance To CBD', '30-39 years'.

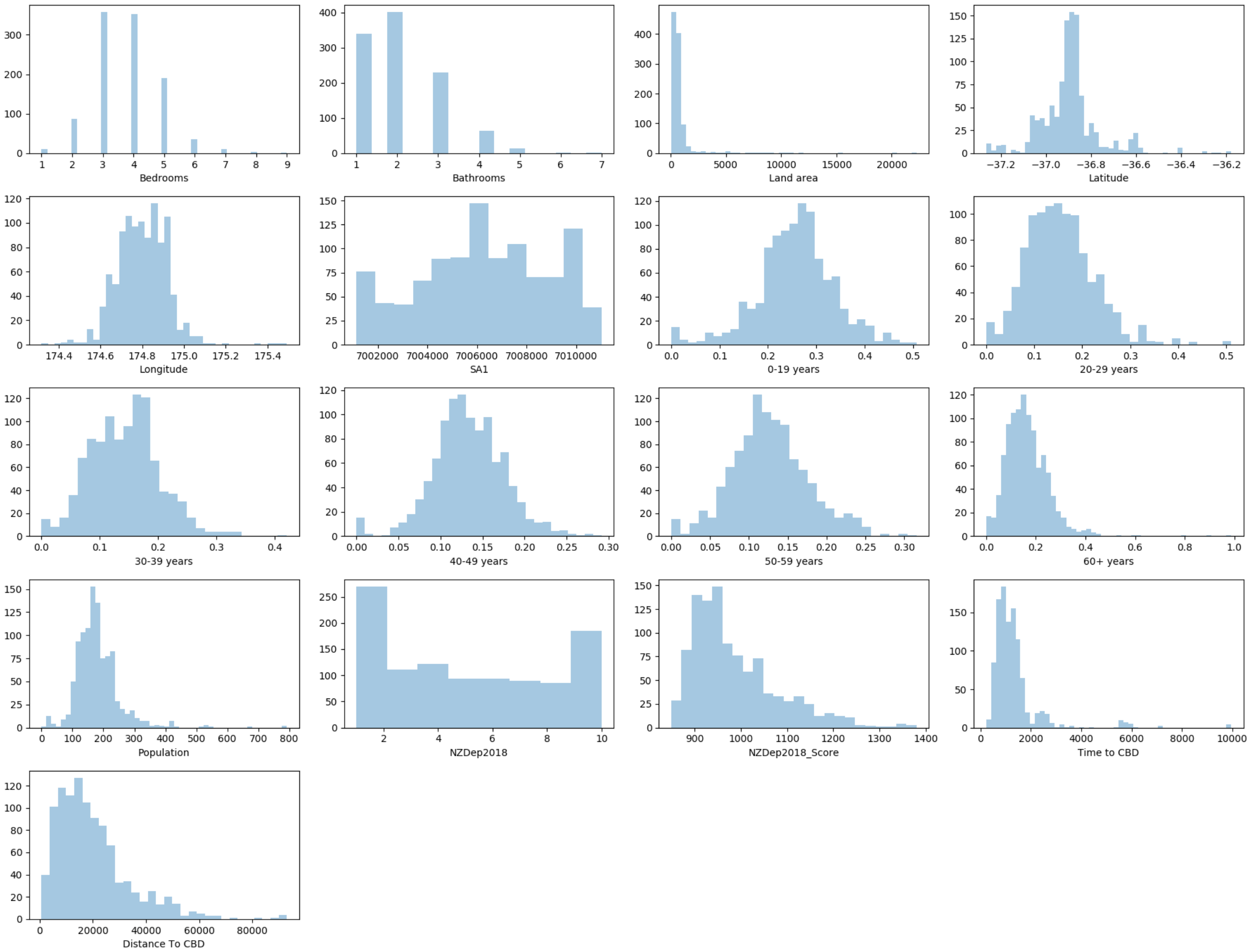
From the plots above, we can see that the clear outliers are in distributions “Bathrooms” (7, 8) and “Bedrooms” (17), thus they will be removed from the data-frame [3.4.1].

3.4.2 Distributions of numerical features

From the plots above, it can be seen that most distributions are right-skewed and unimodal. In particular, most of age distributions are heavily right skewed, which is likely due to differences in population size in census areas (SA1).



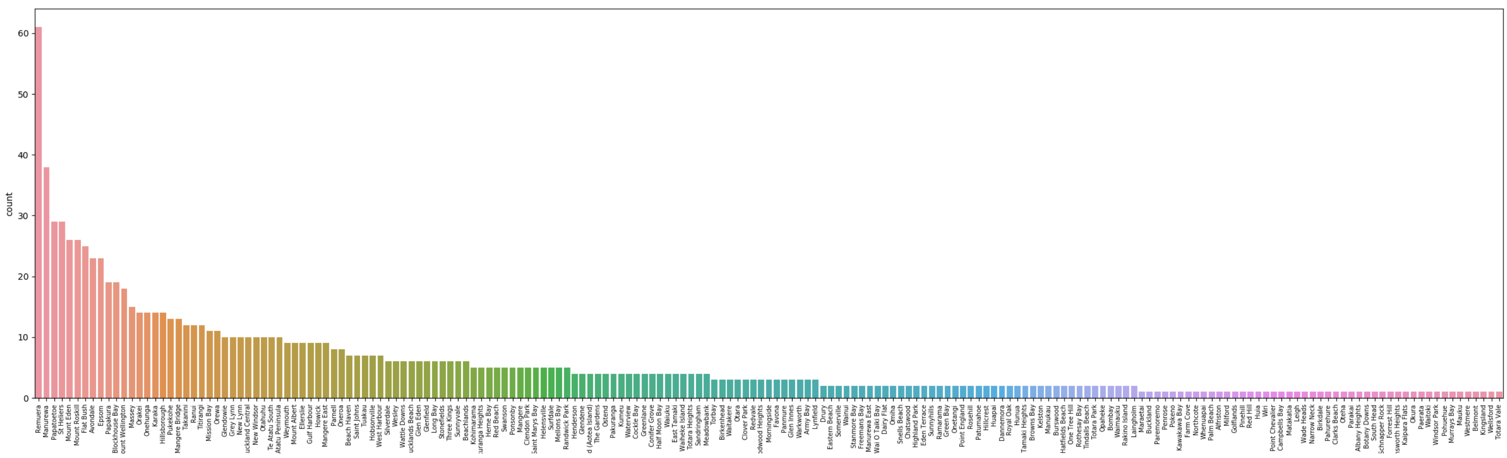
Therefore, age distributions can be normalised by taking the proportion of each age gap, in each census area. The distributions with normalised ages are shown below.



Now it can be seen that, disregarding the outliers, most distributions appear to be symmetrical. Therefore, we can use **Robust Scaler**, which transforms the data in relation to interquartile range and is robust to outliers, which are abundant in this data [3.4.2].

3.4.3 Distribution of categorical features

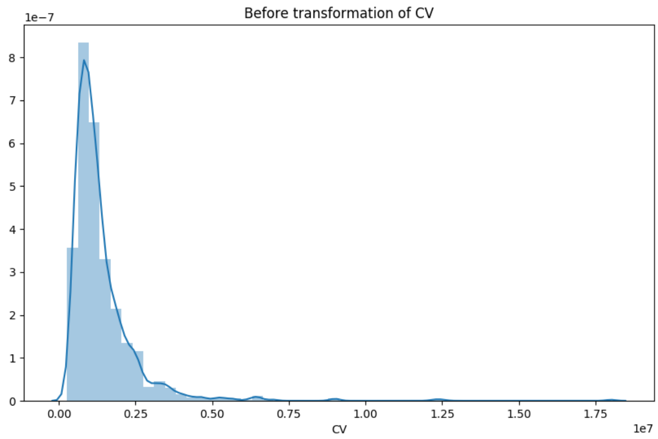
The only categorical variable is “Suburbs”, the frequency plot below shows that most suburbs have more than one property and the number of unique suburb names is not too big, thus we can use ‘one-hot encoding’ to use this data in the model [3.4.3].

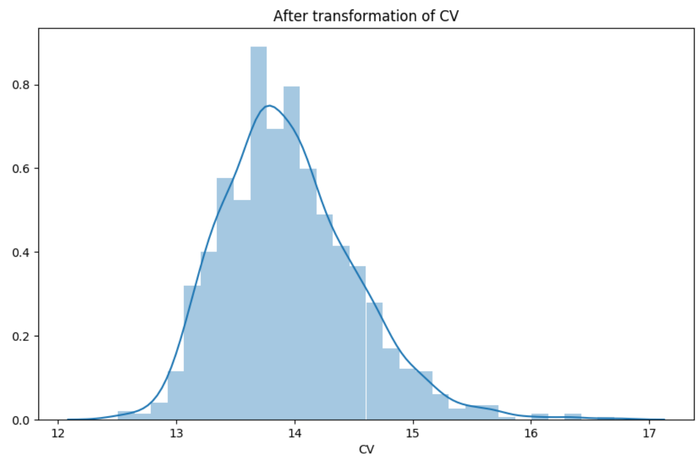


The frequency plot shows that suburbs is not the best feature to be used for modelling, since the number of houses in each suburb greatly varies, for example Remuera is much greater in size than most other suburbs this it appears to be an outlier. It is possible that postcodes could have been better for modelling as they are more consistent in size.

3.4.4 Distribution of target variable (CV)

The CV distribution is heavily right-skewed (left), showing that underlying distribution of CV is likely to be exponentially distributed.
Therefore, we apply log to make it normally distributed. After transformation the distribution appears to be symmetrical [3.4.4].





Modelling

4.1 Training and test sets

First we split data into training and test sets to be used for model fitting and model evaluation [4.1].

4.2 Model Selection

To gain practical experience we will be training and tuning ensemble models for predictions. Then linear models will be used to gain valuable information for executive summary.
We will be using the following ensemble models: XGBoostRegressor, CatBoostRegressor, Random forest regressor.

4.3 Ensemble models hyper-parameter tuning

To tune the hyper-parameters for the ensemble models we will define a parameter list for each of the models and then use **RandomizedSearchCV** to identify the best parameters for the model. RandomizedSearchCV will sample settings for each model 100 and will use default 5 cross-validation splitting strategy.

4.3.1 XGBoost Regressor

We first define a list of hyper-parameters for XGBoost Regressor and then sample 100 of them using RandomizedSearchCV. After the best parameters are found we fit the model using train_x and train_y data sets. The model weights are then saved for later metric evaluation [4.3.1].

4.3.2 CatBoost Regressor

Similarly as with XGBoost, we define a different list of hyper-parameters for CatBoost Regressor and then sample them using RandomizedSearchCV. We then fit the model using the best parameters and save the weights for later metric evaluation [4.3.2].

4.3.3 Random Forest Regressor (RFR)

During hyper-parameter tuning for RFR we find that it is much quicker to train, thus we can take a two step process in determining the best parameters. We first define a parameter list with wide range for each of the parameters and then use RandomizedSearchCV to select the best settings.
Now we define a new list of parameters with much smaller range, with values centred around the best settings that we got from RandomizedSearchCV. We then apply GridSearchCV to the new parameter list to test all the combinations of the parameters and return the best settings for the RFR [4.3.3].

4.4 Model evaluation and selection

We will be using the following metrics for model evaluation: MAE, RMSE, Score (R^2)[1].
After running the metrics on the models we get the following table [4.4]:

```
Sorted by Score:
      Model  CV(5)    MAE   RMSE  Score
1   XGBoost  0.585  0.234  0.354  0.617
0  CatBoost  0.601  0.238  0.360  0.605
2       RFG  0.575  0.248  0.376  0.570
```

From the table above it can be seen that XGBoost appears to be the best model for prediction, as it has the smallest MAE and RMSE.
XGBoost explains 61.7% of the variability in the data, and is closely followed by CatBoost which explains 60.5% of variability in the data.

It appears that both Gradient boosting machines outperform Random forest on this data.
The better performance of Gradient boosting over Random forest is generally recognised in Machine Learning, however Gradient boosting is more susceptible to overfitting on noisy data, which could explain the difference in performance in our case.
However, we can assume that we avoided overfitting by removing features with multicollinearity, outliers and by scaling the data. Additionally, the difference in scores between the models is not large enough to be attributed to overfitting.

Therefore, we can use XGBoost regressor for future predictions.

4.5 Linear model in R

We will create a linear model in R to quantify the relationships between the variables. After applying Occam's razor method, the final model is [4.5]:

$$log(CV_i) = \beta_0 + \beta_1 \times Suburb_i + \beta_2 \times LandArea_i \times TimeToCBD_i + \beta_3 \times Bedrooms_i + \beta_4 \times Bathrooms_i +$$

$$\beta_5 \times LandArea_i + \beta_6 \times NZDep2018 + \beta_7 \times DistanceToCBD + \beta_8 \times 30 - 39years + \beta_9 \times 40 - 49years + \beta_{10} \times 50 - 59years$$

Multiple R^2 of the model, was 0.71. Therefore, model explained 71% of the variability in the data.

---

[1] According to my research scikit learn score is R squared value.

Executive summary

We have fitted 3 ensemble models and 1 linear model. The best performing ensemble model was XGBoost which explained 61% of variability in the data. The linear multiplicative model explained 71% of variability.
The much better performance of the linear model could possibly be attributed to the relatively small dataset and limited amount of features, which would limit the performance of the gradient boosting models.

Using the linear model we estimate:

- For every additional bedroom in the house, the median CV increases by somewhere between 3.8% and 10.4%
- For every additional bathroom in the house, the median CV increases by somewhere between 6.4% and 14%.

The impact of bathrooms and bedrooms on the CV is likely to be indirect and is reflecting the relationship between CV and floor area, since larger floor area will allow for more bedrooms and bathrooms.

Additionally, we estimate:

- For every additional square meter in land area, the median CV increases by somewhere between 0.008% and 0.014%
- For every additional kilometre it takes to get to CBD, the median CV decreases by somewhere between 0.45% and 3.7%
- For every additional point in NZ deprivation index of the house's census area, the median CV decreases by somewhere between 1.6% and 4.1%
- For every additional percent of people in age group (30-39) in population of the house's census area, the median CV decreases by somewhere between 19% and 69%
- For every additional percent of people in age group (40-49) in the population of the house's census area, the median CV decreases by somewhere between 9.9% and 76%
- For every additional percent of people in age group (50-59) in the population of the house's census area, the median CV increases by somewhere between 39% and 365%

From the age group impacts we can conclude that older people can afford more expensive housing, than younger people. This directly follows from older people being being generally wealthier, as they have been accumulating the wealth for longer period of time.

Appendix

Dataset Building

1.1 Adding Population column

```
import json
import sys
sys.path.append("/home/nbuser/library")
import time
import pandas as pd
import requests
```

```
'''Defining function to get population from koordinates API'''
def get_population(lat, lgt):
    url = 'https://koordinates.com/services/query/v1/vector.json'
    api_key = '[redacted]'
    layer_id = 104612
    params = {
        'key' : api_key,
        'layer' : layer_id,
        'y' : lat,
        'x' : lgt,
        'format' : 'json'
    }
    time.sleep(0.1)
    response = requests.get(url, params=params)
    if response.status_code != 200:
        return pd.Series({'Population' : None})

    population = response.json()['vectorQuery']['layers']['104612']['features'][0]['properties']['C18_CNPop']
    return pd.Series({'Population' : population})
```

```
df = pd.read_csv("property.csv")
```

Adding population column

```
df['Population'] = df.apply(lambda row: get_population(row['Latitude'], row['Longitude']), axis=1)
df.head()
```

	Bedrooms	Bathrooms	Address	Land area	CV	Latitude	Longitude	SA1	0-19 years	20-29 years	30-39 years	40-49 years	50-59 years	60+ years	Suburbs	Population
0	5	3.0	106 Lawrence Crescent Hill Park, Auckland	714	960000	-37.012920	174.904069	7009770	48	27	24	21	24	21	Manurewa	177
1	5	3.0	8 Corsica Way Karaka, Auckland	564	1250000	-37.063672	174.922912	7009991	42	18	12	21	15	30	Karaka	123
2	6	4.0	243 Harbourside Drive Karaka, Auckland	626	1250000	-37.063580	174.924044	7009991	42	18	12	21	15	30	Karaka	123
3	2	1.0	2/30 Hardington Street Onehunga, Auckland	65	740000	-36.912996	174.787425	7007871	42	6	21	21	12	15	Onehunga	120
4	3	1.0	59 Israel Avenue Clover Park, Auckland	601	630000	-36.979037	174.892612	7008902	93	27	33	30	21	33	Clover Park	228

1.2 Adding Deprivation index

```
otago_df = pd.read_excel("otago730395.xlsx")
otago_df.head()
```

	SA12018_code	NZDep2018	NZDep2018_Score	URPopnSA1_2018	SA22018_code	SA22018_name
0	7000000	10.0	1245.0	141	100100	North Cape
1	7000001	10.0	1245.0	114	100100	North Cape
2	7000002	NaN	NaN	0	100300	Inlets Far North District
3	7000003	10.0	1207.0	225	100100	North Cape
4	7000004	9.0	1093.0	138	100100	North Cape

```
index = otago_df[['NZDep2018','SA12018_code', 'NZDep2018_Score']]
df = df.merge(right=index, left_on='SA1', right_on='SA12018_code')
df.head()
```

and area	CV	Latitude	Longitude	SA1	0-19 years	20-29 years	30-39 years	40-49 years	50-59 years	60+ years	Suburbs	Population	NZDep2018	SA12018_code	NZDep2018_Score
714	960000	-37.012920	174.904069	7009770	48	27	24	21	24	21	Manurewa	177	6.0	7009770	997.0
564	1250000	-37.063672	174.922912	7009991	42	18	12	21	15	30	Karaka	123	1.0	7009991	881.0
626	1250000	-37.063580	174.924044	7009991	42	18	12	21	15	30	Karaka	123	1.0	7009991	881.0
65	740000	-36.912996	174.787425	7007871	42	6	21	21	12	15	Onehunga	120	2.0	7007871	908.0
601	630000	-36.979037	174.892612	7008902	93	27	33	30	21	33	Clover Park	228	9.0	7008902	1091.0

1.3 Adding Distance and Travel times

```
import googlemaps
import math
# Requires API key
gmaps = googlemaps.Client(key='[redacted]')

# Printing the result
print(my_dist)
```

```
{'distance': {'text': '21.0 km', 'value': 21003}, 'duration': {'text': '21 mins', 'value': 1257}, 'status': 'OK'}
```

```
def distanceToCBD(address):
    destination = "33-43 Princes Street, Auckland CBD, Auckland 1010" #taking Albert park as Auckland CBD
    dist = gmaps.distance_matrix(address,destination)['rows'][0]['elements'][0]
    try:
        timeToCBD = dist['duration']['value']
        distanceToCBD = dist['distance']['value']
    except (KeyError):
        timeToCBD = math.inf
        distanceToCBD = math.inf
    return pd.Series({'Time to CBD' : timeToCBD, 'Distance to CBD' : distanceToCBD})
```

```
df[['Time to CBD', 'Distance To CBD']] =  df.Address.apply(distanceToCBD)
```

Dataset Cleaning

2.1 Dealing with NaN values

```
df.isnull().values.any()
```

```
True
```

```
df.isna().sum()
```

```
Bedrooms           0
Bathrooms          2
Address            0
Land area          0
CV                 0
Latitude           0
Longitude          0
SA1                0
0-19 years         0
20-29 years        0
30-39 years        0
40-49 years        0
50-59 years        0
60+ years          0
Suburbs            1
Population         0
NZDep2018          0
NZDep2018_Score    0
dtype: int64
```

```
df1 = df.loc[df.isna().any(axis=1)]
df1
```

	Bedrooms	Bathrooms	Address	Land area	CV	Latitude	Longitude	SA1	0-19 years	20-29 years	30-39 years	40-49 years	50-59 years	60+ years	Suburbs	Population	N
309	4	NaN	14 Hea Road Hobsonville, Auckland	214	1250000	-36.798371	174.647430	7002267	60	66	60	24	24	18	Hobsonville	246	
311	4	NaN	16 Hea Road Hobsonville, Auckland	245	1100000	-36.798371	174.647430	7002267	60	66	60	24	24	18	Hobsonville	246	
568	1	1.0	14 Te Rangitawhiri Road Great Barrier Island, ...	2141	740000	-36.197282	175.416921	7001131	27	6	6	18	39	60	NaN	156	

2.1.1 NAN IN "SUBURBS"

The empty suburb is due to property being on Great Barrier Island Thus we can check if there are any other Great Barrier Properties

```
islands =[]
for i in df.Suburbs.unique():
    try:
        if "Island" in i:
            islands.append(i)
    except (TypeError):
        print('Value was NaN')
islands
```

```
Value was NaN
```

```
['Great Barrier Island (Aotea Island)', 'Waiheke Island', 'Rakino Island']
```

We can see that datapoint 568 may have suburb 'Great Barrier Island (Aotea Island)'

```
df.set_value(568, 'Suburbs', 'Great Barrier Island (Aotea Island)')
df1 = df.loc[df.isna().any(axis=1)]
df1
```

```
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/ipykernel/__main__.py:1: FutureWarning: set_value is deprecate
d and will be removed in a future release. Please use .at[] or .iat[] accessors instead
  if __name__ == '__main__':
```

	Bedrooms	Bathrooms	Address	Land area	CV	Latitude	Longitude	SA1	0-19 years	20-29 years	30-39 years	40-49 years	50-59 years	60+ years	Suburbs	Population	NZ
309	4	NaN	14 Hea Road Hobsonville, Auckland	214	1250000	-36.798371	174.64743	7002267	60	66	60	24	24	18	Hobsonville	246	
311	4	NaN	16 Hea Road Hobsonville, Auckland	245	1100000	-36.798371	174.64743	7002267	60	66	60	24	24	18	Hobsonville	246	

2.1.2 NAN IN "BATHROOMS"

Since other two NaN are bathrooms we can set them as mean number among the houses with 4 rooms

```
dfOf4Bedrooms = df.loc[df.Bedrooms == 4]
meanBathrooms = int(dfOf4Bedrooms.Bathrooms.mean())
meanBathrooms
```

```
2
```

```
#changing the num of bathrooms of other NaNs to 2
df.set_value(309, 'Bathrooms', 2)
df.set_value(311, 'Bathrooms', 2)
df.loc[df.isna().any(axis=1)].shape
```

```
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/ipykernel/__main__.py:2: FutureWarning: set_value is deprecate
d and will be removed in a future release. Please use .at[] or .iat[] accessors instead
  from ipykernel import kernelapp as app
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/ipykernel/__main__.py:3: FutureWarning: set_value is deprecate
d and will be removed in a future release. Please use .at[] or .iat[] accessors instead
  app.launch_new_instance()
```

```
(0, 18)
```

2.2 Checking column types

```
df.dtypes
```

```
Bedrooms              int64
Bathrooms           float64
Address              object
Land area            object
CV                    int64
Latitude            float64
Longitude           float64
SA1                   int64
0-19 years            int64
20-29 years           int64
30-39 years           int64
40-49 years           int64
50-59 years           int64
60+ years             int64
Suburbs              object
Population            int64
NZDep2018           float64
NZDep2018_Score     float64
dtype: object
```

```
# Land area should be an integer thus we check it
df['Land area'].unique()
```

```
array(['714', '564', '626', '65', '601', '100', '531', '1024', '80',
       '204', '170', '637', '640', '650', '138', '75', '724', '429',
       '520', '1381', '732', '799', '1105', '463', '681', '4068', '106',
       '713', '211', '402', '883', '883 m²', '675', '388', '1034', '1295',
       '1102', '551', '809', '1108', '745', '613', '758', '727', '59',
       '260 m²', '126', '615', '756', '3609', '431', '3648', '3177',
       '545', '420 m²', '481', '279', '120', '1037', '202', '1031', '602',
       '810', '475', '736', '110', '99', '153', '245', '2567 m²', '1500',
```

```
# some rows in land area contain m^2, thus we remove it
def remove_m2(landArea):
    msquared = 'm²'
    if msquared in landArea:
        landArea = landArea[:3]
    return landArea
df['Land area'] = df['Land area'].apply(remove_m2)
df['Land area'] = df['Land area'].astype(float)
df.dtypes
```

```
Bedrooms              int64
Bathrooms           float64
Address              object
Land area           float64
CV                    int64
Latitude            float64
Longitude           float64
SA1                   int64
0-19 years            int64
20-29 years           int64
30-39 years           int64
40-49 years           int64
50-59 years           int64
60+ years             int64
Suburbs              object
Population            int64
NZDep2018           float64
NZDep2018_Score     float64
dtype: object
```

2.3 Dealing with Infinite distances and travel times

```
df1 = df.loc[df['Time to CBD'] == math.inf]
df1
```

Land area	CV	Latitude	Longitude	SA1	0-19 years	20-29 years	30-39 years	40-49 years	50-59 years	60+ years	Suburbs	Population	NZDep2018	NZDep2018_Score	Time to CBD	Distance To CBD
538.0	580000	-36.177655	175.359070	7001130	39	9	18	24	24	42	Great Barrier Island (Aotea Island)	207	9.0	1093.0	inf	inf
141.0	740000	-36.197282	175.416921	7001131	27	6	6	18	39	60	Great Barrier Island (Aotea Island)	156	9.0	1122.0	inf	inf

953.0	920000	-36.257895	175.436448	7001131	27	6	6	18	39	60	Great Barrier Island (Aotea Island)	156	9.0	1122.0	inf	inf
366.0	270000	-36.719592	174.949563	7001354	0	0	0	0	0	6	Rakino Island	24	6.0	991.0	inf	inf
40.0	3250000	-36.719672	174.951524	7001354	0	0	0	0	0	6	Rakino Island	24	6.0	991.0	inf	inf
538.0	650000	-36.305955	175.492424	7001135	30	21	21	21	39	69	Great Barrier Island (Aotea Island)	198	9.0	1107.0	inf	inf

We can see that 3 entries are from great barrier island and 2 are from Rakino island. From a google search we estimate that travel time to Great barrier is approximately 2h46 (93km) and to (21.7 km) Rakino is 1h02 by ferry

```
timetoGBI = (2*60+46)*60 #to sec
distanceToGBI = 93000 #to meters
timetoRI = 62*60 # to sec
distanceToRI = 21700
def change_distance(row):
    suburb = row['Suburbs']
    distance = row['Distance To CBD']
    time = row['Time to CBD']
    if suburb == 'Great Barrier Island (Aotea Island)':
        distance = distanceToGBI
        time = timetoGBI
    elif suburb == 'Rakino Island':
        distance = distanceToRI
        time = timetoRI
    return pd.Series({'Time to CBD': time, 'Distance To CBD': distance})
```

Data Analysis

3.1 Correlation matrix

```
corr = data.corr()
ax = sns.heatmap(
    corr,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(20, 220, n=200),
    square=True
)
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=45,
    horizontalalignment='right'
);
plt.show()
```

3.2 Addressing Multicollinearity

```
correlation = data.corr()
print(correlation[['CV']].sort_values(['CV'], ascending=False))

plt.figure(figsize=(14,12))
correlation = numeric_col.corr()
sns.heatmap(correlation, mask = correlation <0.8, linewidth=0.5, cmap='Blues')
```

3.3 Removing redundant features

```
correlation = data.corr()
print(correlation[['CV']].sort_values(['CV'], ascending=False))
```

```
X.drop(['NZDep2018_Score', 'Longitude', 'Address', 'Population'], axis=1, inplace=True)
```

3.4 Analysing individual distributions

3.4.1 REMOVING SIGNIFICANT OUTLIERS

```
out_col = ['Bedrooms','Bathrooms', 'Distance To CBD', '30-39 years']
fig = plt.figure(figsize=(20,5))
for index,col in enumerate(out_col):
    plt.subplot(1,5,index+1)
    sns.boxplot(y=col, data=data)
fig.tight_layout(pad=1.5)

data = data.drop(data[(data['Bedrooms'] > 10) & (data['Bathrooms'] > 6)].index)
```

3.4.2 DISTRIBUTIONS OF NUMERICAL FACTORS

```
numeric_col = X.select_dtypes(exclude=['object']).copy()

fig = plt.figure(figsize=(18,16))
for index,col in enumerate(numeric_col.columns):
    plt.subplot(6,4,index+1)
    sns.distplot(numeric_col.loc[:,col].dropna(), kde=False)
fig.tight_layout(pad=1.0)
```

```
# converting population to percentages
def population_to_percentage(age_gap, population):
    try:
        age_gap = age_gap/population
    except (ZeroDivisionError):
        pass
    return age_gap

#converting population to percentage
X['0-19 years'] = X.apply(lambda row: population_to_percentage(row['0-19 years'],
                row['Population']), axis=1)
X['20-29 years'] = X.apply(lambda row: population_to_percentage(row['20-29 years'],
                row['Population']), axis=1)
X['30-39 years'] = X.apply(lambda row: population_to_percentage(row['30-39 years'],
                row['Population']), axis=1)
X['40-49 years'] = X.apply(lambda row: population_to_percentage(row['40-49 years'],
                row['Population']), axis=1)
X['50-59 years'] = X.apply(lambda row: population_to_percentage(row['50-59 years'],
                row['Population']), axis=1)
X['60+ years'] = X.apply(lambda row: population_to_percentage(row['60+ years'],
                row['Population']), axis=1)
```

```
from sklearn.preprocessing import RobustScaler

cols = X.select_dtypes(np.number).columns
transformer = RobustScaler().fit(X[cols])
X[cols] = transformer.transform(X[cols])
```

3.4.3 DISTRIBUTION OF CATEGORICAL FEATURES

```
cat_train = X.select_dtypes(include=['object']).copy()

plt.figure(figsize=(30,8))
ax = sns.countplot(x=cat_train.iloc[:,1], data=cat_train.dropna(),
                    order = cat_train['Suburbs'].value_counts().index)
plt.xticks(rotation=90, fontsize=7)
```

```
X = pd.get_dummies(X)
```

3.4.4 DISTRIBUTION OF TARGET VARIABLE (CV)

```
plt.figure(figsize=(10,6))
plt.title("Before transformation of CV")
dist = sns.distplot(data['CV'],norm_hist=False)

plt.figure(figsize=(10,6))
plt.title("After transformation of CV")
dist = sns.distplot(np.log(data['CV']),norm_hist=False)

y = np.log(y)
```

Modelling

4.1 Training and test sets

```
train_x, test_x, train_y, test_y = train_test_split(X,y,test_size=0.3,random_state=42)
```

4.3 Ensemble models

4.3.1 XGBOOST REGRESSOR

```
from sklearn.metrics import mean_squared_error, mean_absolute_error
from xgboost import XGBRegressor
from sklearn import ensemble
from sklearn.model_selection import cross_val_score
from catboost import CatBoostRegressor
from joblib import dump
from joblib import load

from sklearn.model_selection import RandomizedSearchCV

param_lst = {
    'learning_rate' : [0.01, 0.1, 0.15, 0.3, 0.5],
    'n_estimators' : [100, 500, 1000, 2000, 3000],
    'max_depth' : [3, 6, 9],
    'min_child_weight' : [1, 5, 10, 20],
    'reg_alpha' : [0.001, 0.01, 0.1],
    'reg_lambda' : [0.001, 0.01, 0.1]
}
```

```
xgb = XGBRegressor(booster='gbtree', objective='reg:squarederror')

xgb_reg = RandomizedSearchCV(estimator = xgb, param_distributions = param_lst,
                             n_iter = 100,  cv = 5)

xgb_search = xgb_reg.fit(train_x, train_y)
best_param = xgb_search.best_params_
xgb = XGBRegressor(**best_param)
xgb.fit(train_x, train_y)
dump(xgb, "pima_xgb.joblib.dat")
print('Saved model to: pima_xgb.joblib.dat')
```

4.3.2 CATBOOST REGRESSOR

```
cb = CatBoostRegressor(loss_function='RMSE', logging_level='Silent')

param_lst = {
    'n_estimators' : [100, 300, 500, 1000, 1300, 1600],
    'learning_rate' : [0.0001, 0.001, 0.01, 0.1],
    'l2_leaf_reg' : [0.001, 0.01, 0.1],
    'random_strength' : [0.25, 0.5 ,1],
    'max_depth' : [3, 6, 9],
    'min_child_samples' : [2, 5, 10, 15, 20],
    'rsm' : [0.5, 0.7, 0.9],
}

catboost = RandomizedSearchCV(estimator = cb, param_distributions = param_lst,
                              n_iter = 100,
                              cv = 5)

catboost_search = catboost.fit(train_x, train_y)

best_param = catboost_search.best_params_
cb = CatBoostRegressor(logging_level='Silent', **best_param)
cb.fit(train_x, train_y)
dump(cb, "pima.joblib.dat")
print("Saved model to: pima.joblib.dat")
```

4.3.3 RANDOM FOREST REGRESSOR (RFR)

```
from sklearn.ensemble import RandomForestRegressor

n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
bootstrap = [True, False]

# Create the parameter list
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

rf = RandomForestRegressor()
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 100,
                               cv = 3, verbose=2, random_state=42, n_jobs = -1)
rf_random.fit(train_x, train_y)

print(rf_random.best_params_)
```

Using grid search on best random search parameters

```
from sklearn.model_selection import GridSearchCV

# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [False],
    'max_depth': [60, 70, 80, 90],
    'max_features': [2, 3],
    'min_samples_leaf': [1, 2, 3],
    'min_samples_split': [3, 5, 7],
    'n_estimators': [400, 600, 800, 1000]
}
rf = RandomForestRegressor()
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                          cv = 3, n_jobs = -1, verbose = 2)

grid_search.fit(train_x, train_y)
print(grid_search.best_params_)
```

4.4 Model evaluation and selection

```
def mean_cross_val(model, X, y):
    score = cross_val_score(model, X, y, cv=5)
    mean = score.mean()
    return mean


rf = RandomForestRegressor(bootstrap = False, max_depth = 90,
                           max_features=3, min_samples_leaf=1,
                           min_samples_split=5, n_estimators=1000)
rf.fit(train_x, train_y)
preds = rf.predict(test_x)
preds_test_rf = rf.predict(test_x)
mae_rf = mean_absolute_error(test_y, preds)
rmse_rf = np.sqrt(mean_squared_error(test_y, preds))
score_rf = rf.score(test_x, test_y)
cv_rf = mean_cross_val(rf, X, y)

cb = load("pima.joblib.dat")
preds = cb.predict(test_x)
preds_test_cb = cb.predict(test_x)
mae_cb = mean_absolute_error(test_y, preds)
rmse_cb = np.sqrt(mean_squared_error(test_y, preds))
score_cb = cb.score(test_x, test_y)
cv_cb = mean_cross_val(cb, X, y)

xgb = load("pima_xgb.joblib.dat")
preds = xgb.predict(test_x)
preds_test_xgb = xgb.predict(test_x)
mae_xgb = mean_absolute_error(test_y, preds)
rmse_xgb = np.sqrt(mean_squared_error(test_y, preds))
score_xgb = xgb.score(test_x, test_y)
cv_xgb = mean_cross_val(xgb, X, y)
```

```python
model_performances = pd.DataFrame({
    "Model" : ["CatBoost", "XGBoost", "RFG"],
    "CV(5)" : [str(cv_cb)[0:5], str(cv_xgb)[0:5], str(cv_rf)[0:5]],
    "MAE" : [str(mae_cb)[0:5], str(mae_xgb)[0:5], str(mae_rf)[0:5]],
    "RMSE" : [str(rmse_cb)[0:5], str(rmse_xgb)[0:5], str(rmse_rf)[0:5]],
    "Score" : [str(score_cb)[0:5], str(score_xgb)[0:5], str(score_rf)[0:5]]
})

print("Sorted by Score:")
print(model_performances.sort_values(by="Score", ascending=False))
```

4.5 Linear Model in R

```r
property.fit = lm(log(CV)~  Suburbs + Land.area*Time.to.CBD + Bedrooms+Bathrooms+Land.area +   NZDep
2018+  Distance.To.CBD  + X30.39.years + X40.49.years+  X50.59.years , data=property.df)
summary(property.fit)
```

```
## Land.area                              2.26e-10 ***
## Time.to.CBD                            0.121476
## Bedrooms                               1.14e-05 ***
## Bathrooms                              4.75e-08 ***
## NZDep2018                              7.86e-06 ***
## Distance.To.CBD                        0.012354 *
## X30.39.years                           0.004898 **
## X40.49.years                           0.023381 *
## X50.59.years                           0.003617 **
## Land.area:Time.to.CBD                  0.001588 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3374 on 851 degrees of freedom
## Multiple R-squared:  0.7123, Adjusted R-squared:  0.6454
## F-statistic: 10.64 on 198 and 851 DF,  p-value: < 2.2e-16
```

```r
100 * (exp(confint(property.fit)) - 1)
```

```
## Land.area                               7.751763e-03  1.457876e-02
## Time.to.CBD                            -6.037355e-03  5.141702e-02
## Bedrooms                                3.863094e+00  1.035880e+01
## Bathrooms                               6.420545e+00  1.400720e+01
## NZDep2018                              -4.111638e+00 -1.633538e+00
## Distance.To.CBD                        -3.680651e-03 -4.483301e-04
## X30.39.years                           -6.963511e+01 -1.925284e+01
## X40.49.years                           -7.605762e+01 -9.891513e+00
## X50.59.years                            3.511251e+01  3.654121e+02
## Land.area:Time.to.CBD                  -3.735575e-06 -8.776925e-07
```

The model contains an individual coefficient for every suburb, however since there are 200+ suburbs I will show only the coefficients and CI of the first 5.

```
##                                                 2.5 %        97.5 %
## (Intercept)                             2.291614e+07  1.087951e+08
## SuburbsAlfriston                       -2.592961e+01  3.916897e+02
## SuburbsArmy Bay                         9.365625e-01  4.584169e+02
## SuburbsAuckland Central                -4.737811e+01  1.408002e+02
## SuburbsAvondale                        -7.334828e+00  2.983721e+02
## SuburbsBeach Haven                     -3.125907e+01  2.104385e+02
```