**Work placement project: Exploratory analysis of Dublin Airbnb data and rental price prediction**

Author: **Andrius Dalisanskis**

Supervisor: **Martin O'Rourke**

*This project is submitted in partial fulfilment of the Higher Diploma in Data Analytics at Athlone Institute of Technology*

*in the*

*Department of Accounting and Business Computing*

30th March 2021

*I have read the Institute's code of practice on plagiarism. I hereby certify this material, which I now submit for assessment on the programme of study leading to the award of Higher Diploma in Data Analytics is entirely my own work and has not been taken from the work of others, only to the extent that such work has been cited within the text of my work.*

*Student ID Number:* **A00277506**

*Name of Candidate:* **Andrius Dalisanskis**

**

*Signature of Candidate:*

*Date:* **30th March 2021**

Abstract

Airbnb is a competitive environment, and hosts face the challenging task of setting the price for their accommodation. They must consider how their listings' characteristics compare to competing offers on the platform to generate good value. This project investigates how the features of the properties and the locations affect the prices. It contains an analysis of the Inside Airbnb listings and OpenStreetMap data. The research investigated the predictive power of three machine learning models: Linear Regression, Random Forest, and Neural Network. The machine learning models utilised the data with and without the points of interest features obtained from the OpenStreetMap service. The models' findings illustrated how the points of interest features do not improve the accuracy of the predictions.

Contents

[Chapter 1 Introduction 1](#_Toc67907217)

[1.1 Justification 1](#_Toc67907218)

[1.2 Aims and objectives 1](#_Toc67907219)

[1.3 Literature review 2](#_Toc67907220)

[Chapter 2 Methodology and implementation 4](#_Toc67907221)

[2.1 Data exploration 4](#_Toc67907222)

[2.1.1 Airbnb listings data 4](#_Toc67907223)

[2.1.2 OpenStreetMap data 6](#_Toc67907224)

[2.1.3 The analysis of the target variable 7](#_Toc67907225)

[2.1.4 The analysis of the feature variables 10](#_Toc67907226)

[2.1.5 Correlation 15](#_Toc67907227)

[2.2 Data preparation 16](#_Toc67907228)

[2.3 Modelling 18](#_Toc67907229)

[2.3.1 Neural Network 19](#_Toc67907230)

[2.3.2 Linear Regression 21](#_Toc67907231)

[2.3.3 Random Forest 22](#_Toc67907232)

[2.4 Evaluation 23](#_Toc67907233)

[Chapter 3 Conclusion 27](#_Toc67907234)

[3.1 Discussion 27](#_Toc67907235)

[3.2 Future work 28](#_Toc67907236)

[Reflection 29](#_Toc67907237)

[References 30](#_Toc67907238)

[Appendix 31](#_Toc67907239)

[Source codes 31](#_Toc67907240)

[OpenStreetMap scraper. Python script 31](#_Toc67907241)

[Feature selection. R script 32](#_Toc67907242)

[Data cleansing. R script 32](#_Toc67907243)

[Data analysis. R script 33](#_Toc67907244)

[Machine learning. R script 38](#_Toc67907245)

List of Tables

[Table 1 Selected features 4](#_Toc67878532)

[Table 2 - Summary statistics for review\_scores\_rating 6](#_Toc67878533)

[Table 3 - Summary statistics for OpenStreetMap features 7](#_Toc67878534)

[Table 4 - Basic price statistics before the cleaning 7](#_Toc67878535)

[Table 5 - Basic price statistics after the cleaning 8](#_Toc67878536)

[Table 6 - Neighbourhoods that have the best location ratings 12](#_Toc67878537)

[Table 7 - The parameters of the neural network 21](#_Toc67878538)

[Table 8 - The RMSE and MAE results of the machine learning models (OpenStreetMap data included) 25](#_Toc67878539)

[Table 9 - The RMSE and MAE results of the machine learning models (OpenStreetMap data excluded) 25](#_Toc67878540)

List of Figures

[Figure 1 - Map of the properties with missing neighbourhood values 5](#_Toc67878541)

[Figure 2 - Map of the properties with not missing neighbourhood values 5](#_Toc67878542)

[Figure 3 - The locations of the most and the least expensive areas 8](#_Toc67878543)

[Figure 4 - The price distribution of the ten most expensive areas 9](#_Toc67878544)

[Figure 5 - The price distribution of the ten least expensive areas 9](#_Toc67878545)

[Figure 7 - Distribution of the host status 10](#_Toc67878546)

[Figure 8 - Price distribution for each host status 10](#_Toc67878547)

[Figure 9 - The mean prices of the room types grouped by host status 10](#_Toc67878548)

[Figure 10 - Distribution of the neighbourhood value 11](#_Toc67878549)

[Figure 11 - Distribution of the property\_type variable 12](#_Toc67878550)

[Figure 13 - Distribution of the room\_type variable 13](#_Toc67878551)

[Figure 14 - Distribution of prices for each room type 13](#_Toc67878552)

[Figure 15 - Counts of each room type by the type of property 14](#_Toc67878553)

[Figure 16 - Distribution of the cancellation\_policy variable 15](#_Toc67878554)

[Figure 17 - Distribution of prices for each cancellation policy type 15](#_Toc67878555)

[Figure 19 - Correlation matrix 16](#_Toc67878556)

[Figure 20 - Artificial neural network 19](#_Toc67878557)

[Figure 21 - Histogram of the price feature with outliers 20](#_Toc67878558)

[Figure 22 - Histogram of the price feature without outliers 20](#_Toc67878559)

[Figure 23 - The distribution of the price variable before the Yeo-Johnson transformation 20](#_Toc67878560)

[Figure 24 - The distribution of the price variable after the Yeo-Johnson transformation 20](#_Toc67878561)

[Figure 25 - Variable importance plot 21](#_Toc67878562)

[Figure 26 - Random forest structure 23](#_Toc67878563)

[Figure 27 - Actual vs Predicted (Linear Regression) 26](#_Toc67878564)

[Figure 28 - Actual vs Predicted (Neural Network) 26](#_Toc67878565)

[Figure 29 - Actual vs Predicted (Random Forest) 26](#_Toc67878566)

[Figure 30 - Residuals vs Fit (Random Forest) 26](#_Toc67878567)

# Introduction

The sharing economy has become immensely popular (Niam Yaraghi, 2017) in many business sectors such as crowdfunding (e.g. Kickstarter), transportation (e.g. Uber, Lyft, Turo), accommodation (e.g. Airbnb, FlipKey, HouseTrip), and many others. Airbnb is probably the best example in the peer-to-peer accommodation industry. It enables homeowners to offer their free spaces for short-term rental. Airbnb has been growing fast since the fall of 2008, when the founders Brian Chesky and Joe Gebbia hosted their first guests. The company have since grown from two hosts to more than four million as of September 2020.

## 1.1 Justification

One of the keys to a successful hosting experience on the Airbnb platform is the price. Being a host on the platform, which currently has more than 5.5 million active listings, is challenging. A host needs to determine how much to ask for their property so that the price is both fair and competitive. Therefore, it is beneficial to know which features associated with the properties influence the price. This knowledge could help hosts to offer better value to the customers and increase the profits. Guests could benefit as well from a tool that permits them to compare offerings and to determine which offer represents better value.

## 1.2 Aims and objectives

This project investigates the factors that influence the price. It utilises data compiled by Inside Airbnb and Point-of-Interest data from OpenStreetMap. Inside Airbnb is an independent set of tools and data which allows members of the public to analyse freely available information with specific regard to the Airbnb listings. OpenStreetMap publishes open data about roads, trails, cafes and much more. The data about places of interest will be scraped for each listing in the dataset.

The exploratory analysis part of the project investigates the following topics:

* How does the rental price vary in Dublin by neighbourhood?
* Which neighbourhoods are rated highest by guests?
* Do regular and super hosts charge different rates?
* Does the number of amenities offered by the property affect the price?
* How does the number of places of interest in the vicinity of the property affect the price?

After analysis, the project will utilise the Linear Regression, the Random Forest, and the Neural Network algorithms as models to predict the Airbnb listings prices and then compare the predictions to the actual values. If successful, the resulting model could be used to create an application that predicts the listing's price based on the properties of the accommodation.

## 1.3 Literature review

The Airbnb phenomenon and the vast amount of data it generates have attracted many studies that explored various topics. Price and pricing strategy are vital for any business, and Airbnb has its own pricing algorithm as described by (Hill, 2015). Hill's research concludes that location, similarity, and recency are primary factors - "similarity" predicts the price of a new listing by comparing it with existing listings that are similar in several unique features, i.e. how many people it sleeps, the type of property (condo, house or even 'cave'), and the number of reviews. The "recency" element adjusts projected listing prices for seasonality and non-cyclical pricing changes. Finally, the location element predicts the impact of location on pricing, given that Airbnb listings are more broadly distributed than hotels, and given the importance of neighbourhood amenities which cannot be determined by a simple distance from the city centre (Robbin Deboosere, 2019).

(Y. Li, 2016) studied the Multi-Scale Affinity Propagation clustering method and used Linear Regression on the result to create a price prediction model for Airbnb. Distance from the listing to the city landmarks was selected as the clustering feature. Another study (MA, 2018) used Random Forests, Regression Tree, Linear Regression and Gradient Boosting Regression Trees to investigate rental prices in Beijing. Their study found that the Tree Regression model performed best.

This project will attempt to add value by investigating how the number of amenities in the area surrounding a property impacts the rental price and its prediction.

# Methodology and implementation

## 2.1 Data exploration

In this part, the researcher will discuss the general overview and observations of the data. The aim is to get a good understanding of the data.

### 2.1.1 Airbnb listings data

The Airbnb listings data is extracted from the Inside Airbnb website (<http://insideairbnb.com/get-the-data.html>). Inside Airbnb generates datasets from the publicly available information on the Airbnb website. This project utilises the dataset generated on the 25th of October 2019 to avoid using a dataset that may have been impacted by the current pandemic. The original dataset contains 9486 observations and 106 features. This research will not utilise all of the features. For example, personal details and text descriptions will be removed. The following table sums up selected features:

|  |  |  |
| --- | --- | --- |
| Feature | Description | Type |
| host\_is\_super\_host | It tells if the host is a super host | Boolean |
| neighbourhood | Neighbourhood of Dublin | Categorical |
| latitude | Latitude | Float |
| longitude | Longitude | Float |
| property\_type | Property type | Categorical |
| room\_type | Room type | Categorical |
| accommodates | How many guests property accommodates | Integer |
| beds | How many beds | Integer |
| amenities | List of amenities that are offered at the property | Integer |
| price | Price | Integer |
| number\_of\_reviews | Number of reviews | Integer |
| review\_scores\_rating | The total score | Integer |
| review\_scores\_location | Location review score | Integer |
| cancellation\_policy | Cancellation policy | Categorical |

Table Selected features

Our dataset has some missing values: 2 from host\_is\_super\_host, 1238 from neighbourhood, 14 from beds, 1557 from review\_scores\_rating, 1568 from review\_scores\_location.

There are many missing neighbourhood values. Let us see how it looks on the map:

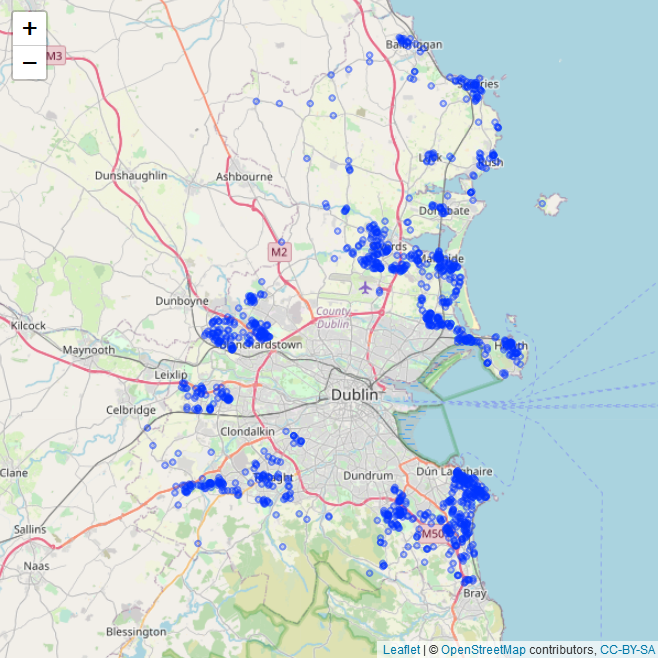


Figure - Map of the properties with missing neighbourhood values

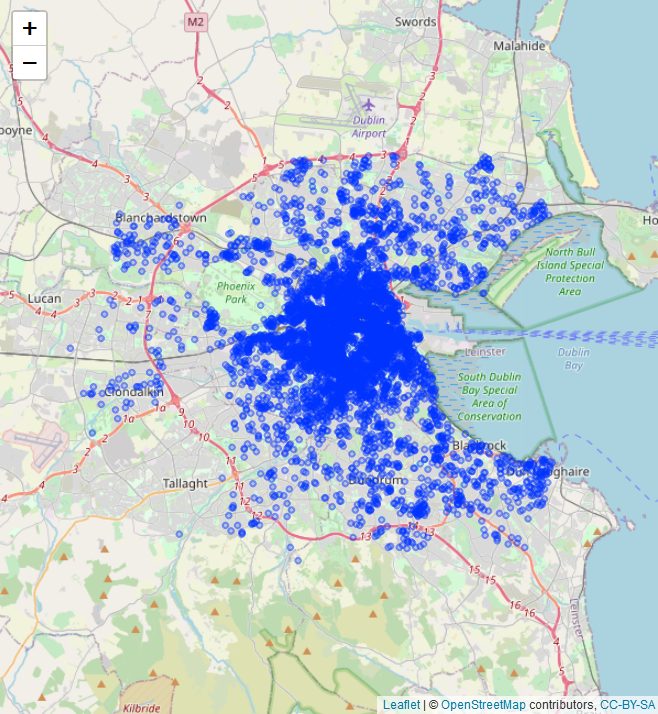


Figure - Map of the properties with not missing neighbourhood values

The maps illustrate how the neighbourhood values are missing not at random. The properties with missing neighbourhood values are primarily based outside the city.

Review\_scores\_rating feature has 1557 missing values. The researcher assumes that they are missing because of an insufficient number of reviews. Further analysis found that listings with no review\_scores\_rating do not have enough reviews.

|  |  |  |
| --- | --- | --- |
|  | Max of number\_of\_reviews | Median of number\_of\_reviews |
| Missing review\_scores\_rating | 4 | 0 |
| Not missing review\_scores\_rating | 655 | 17 |

Table - Summary statistics for review\_scores\_rating

Furthermore, every property with missing review\_scores\_rating also has a missing review\_scores\_location value.

### 2.1.2 OpenStreetMap data

This project will utilise points-of-interest data from OpenStreetMap in addition to Airbnb listings. The researcher chose to use OpenStreetMap because the OpenStreetMap license allows free access to all of the underlying map data. An alternative would be to use Google Places API, but this service comes at a cost. Python and Overpass API are used to write a script that queries OpenStreetMap and retrieves data for each property in the listings dataset. The challenging part of this step is determining how far away from the property should the points-of-interest be counted. One piece of research in the US found that 68% of the walking trips in the study were more than 400m (0.25 miles), and close to one-fifth of trips were longer than 1600m (1 mile) (Yang Y, 2012). Another study in Maastricht (Netherlands) found that the mean distance for the tourists on a shopping trip is 1994m (standard deviation 813.13) (Astrid D.A.M.Kemperman, 2009). The research team found that comfortable walking distance varies greatly depending on many things – on the subjects' socio-economic status, on the area, the purpose of the trip and many more. A distance of 0.25 miles (400 meters) is often used as an acceptable walking distance in US research studies (Yang Y, 2012). Therefore, this project scrapes the point of interest data in a 200m radius.

This step creates four additional features:

1. Bus\_stops – the number of bus stops around a given property
2. Tourism – the number of tourist locations around a given property
3. Amenities\_OSM – the number of pubs, restaurants, clubs, and cafes around a given property
4. Shops – the number of shops around a given property

Some summary statistics for the created features:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Feature | Min | 1st Qu. | Median | Mean | 3st Qu. | Max |
| Bus\_stops | 0 | 1 | 3 | 3.5 | 4 | 51 |
| Tourism | 0 | 0 | 1 | 2.8 | 3 | 32 |
| Amenities\_OSM | 0 | 0 | 2 | 6.8 | 8 | 118 |
| Shops | 0 | 0 | 4 | 10.6 | 13 | 105 |

Table - Summary statistics for OpenStreetMap features

### 2.1.3 The analysis of the target variable

The following table presents the basic statistics for the price variable:

|  |  |
| --- | --- |
| Mean | 136.5 |
| Median | 90 |
| Minimum | 9 |
| Maximum | 22337 |

Table - Basic price statistics before the cleaning

The maximum price is extraordinarily high. Therefore, it is vital to analyse if there is anything unusual about the most expensive listings, which makes them demonstrably not a part of the general population of listings. The Airbnb listing investigation shows that the most expensive property does not stand out from similar properties. The current listing price per night on Airbnb is a little over 500, and it can be booked only for the summer months because it is purpose-built student accommodation. The top four of the most expensive listings in our dataset are located in the same building.

Further investigation of the most expensive listings found similar issues. The researcher chose to match the project datasets' prices to the prices on the Airbnb website where possible. The most expensive properties with no current data on the Airbnb website were removed. The following table shows the basic statistics for the price after the cleansing process:

|  |  |
| --- | --- |
| Mean | 124.5 |
| Median | 90 |
| Minimum | 9 |
| Maximum | 1000 |

Table 5 - Basic price statistics after the cleaning

The following figure shows that the top ten of the most expensive neighbourhoods are located south of the River Liffey and are close to the city centre. The least expensive areas are located furthest from the city centre.

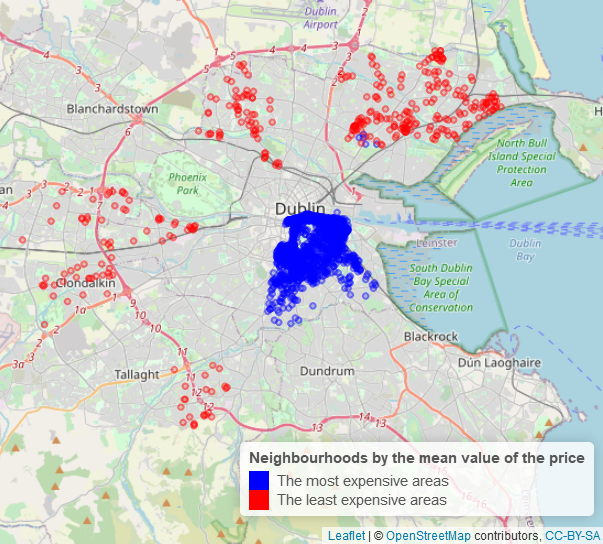


Figure - The locations of the most and the least expensive areas

The following figures show the price distributions for the ten most expensive and ten least expensive neighbourhoods:

|  |  |
| --- | --- |
| Figure - The price distribution of the ten most expensive areas | Figure - The price distribution of the ten least expensive areas |

The boxplots illustrate that both batches of data appear to be right-skewed. The overall ranges of the prices are similar for both datasets. The main difference is that the interquartile range is three times wider for the most expensive areas. Also, it is interesting to note that the first quantile of the most expansive areas is almost equal to the third quantile of the least expensive areas (values are 79 and 77, respectively). Mean values are 120 for the most expensive areas and 50 for the least expensive areas. Generally speaking, listings located closest to the city centre are more than twice more expensive than the listings which are located furthest away from the city centre.

### 2.1.4 The analysis of the feature variables

Variable *host\_is\_superhost* indicates if the host has a super host status. The following figures show the distribution of the *host\_is\_superhost* values and the price distributions for each host type.

|  |  |
| --- | --- |
| Figure - Distribution of the host status | Figure - Price distribution for each host status |

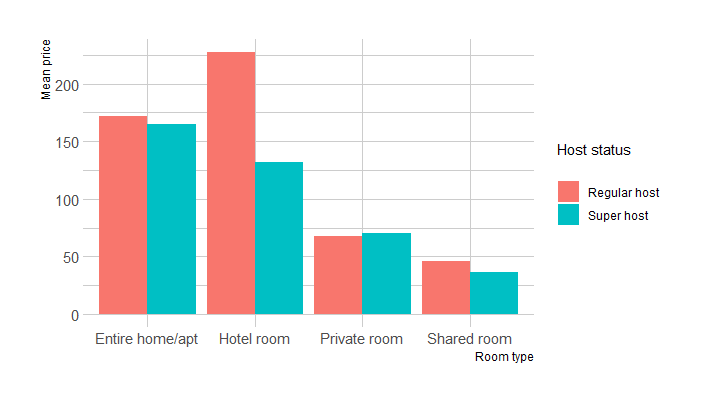
We can observe that the number of regular hosts is much higher than super hosts. It appears that, on average, super hosts and regular hosts charge similar prices. However, a closer examination of the figures determined that most super hosts charge slightly less for similar properties (Figure 9).

Figure - The mean prices of the room types grouped by host status

The following figure shows the distribution of the *neighbourhood* variable:

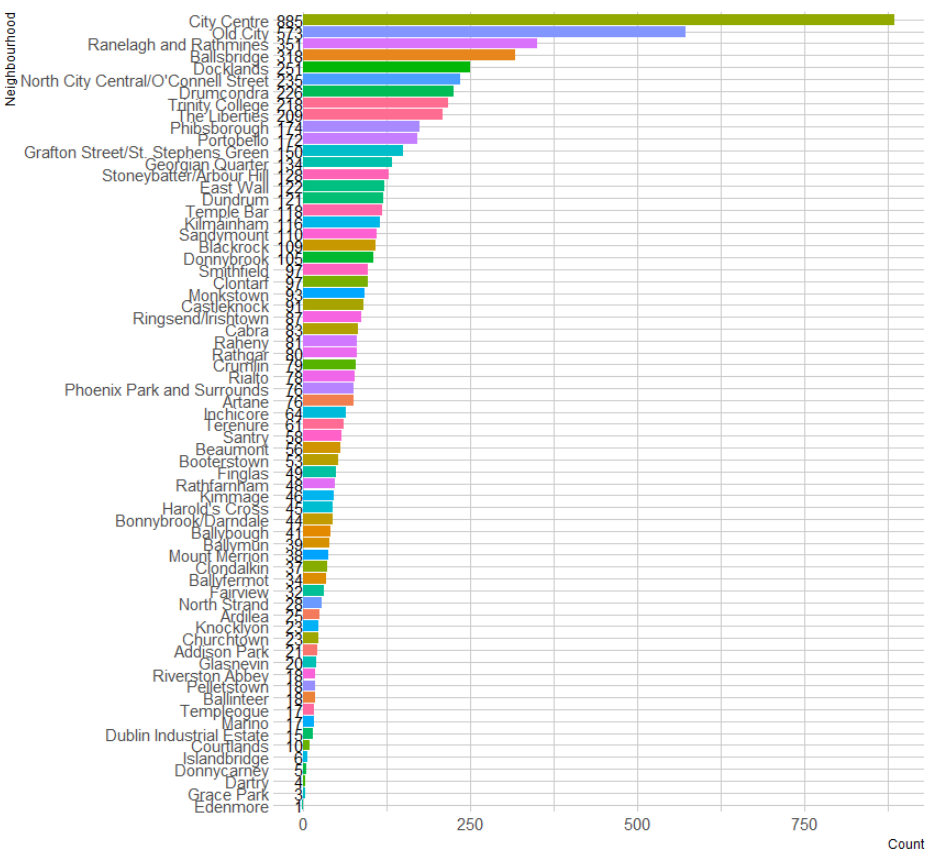


Figure - Distribution of the neighbourhood value

As we can see from the figure, the city centre and areas around it have the most listings in the project dataset. Table 6 shows the areas which have the best location ratings. Edenmore neighbourhood has only one review and should be disregarded. Please note that the top five neighbourhoods by location rating are city centre areas.

|  |  |
| --- | --- |
| Neighbourhood | Average location rating |
| Edenmore | 10 |
| Temple Bar | 9.96 |
| Grafton Street/St. Stephens Green | 9.84 |
| Smithfield | 9.80 |
| North City Central/O'Connell Street | 9.79 |
| Portobello | 9.76 |

Table - Neighbourhoods that have the best location ratings

The figure below emphasises how apartments and houses are the most popular property types. These properties represent 80% of the total listings.

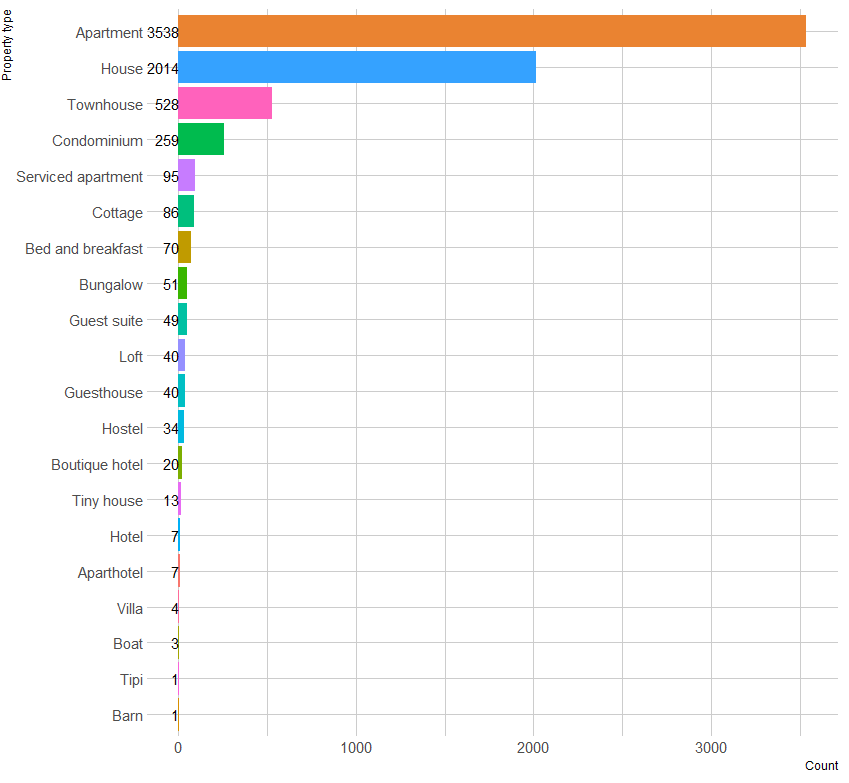


Figure - Distribution of the property\_type variable

The following figures show the number of occurrences for the room types and the price distributions for each type:

|  |  |
| --- | --- |
| Figure - Distribution of the room\_type variable | Figure - Distribution of prices for each room type |

Figure 13 illustrates that the entire home/apartment and private room are the most popular accommodation types. It is interesting to see that the price distributions for entire homes, apartments, and hotel rooms are reasonably similar (Figure 14) - both have similar mean and interquartile values. The most significant difference between the two is the outliers. Entire homes and apartments listings have a considerably larger amount of outliers. The researcher assumes that the imbalanced dataset is the root cause for this – there are only 96 hotel listings and 3640 entire homes.

The cheapest room type options are private rooms and shared rooms. The boxplot of private room prices shows a considerable amount of outliers, and some rooms are very expensive. This raises the concern that there might be listings that have inappropriate room type values.

The following figure illustrates the number of *private room* for each property type:

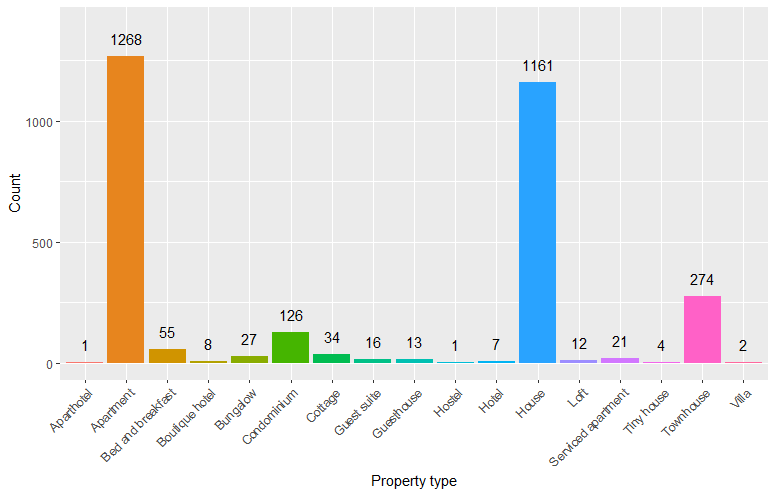


Figure - Counts of **private room** by the type of property

The plot confirms the suspicion. There is one aparthotel, eight boutique hotels, one hostel and seven hotels with the room type value set as a private room, i.e. the more appropriate value for the room type when the property is a hotel is a *hotel room*.

The following figures show the number of occurrences for the cancellation policy types and the price distributions for each type:

|  |  |
| --- | --- |
| Figure - Distribution of the cancellation\_policy variable | Figure - Distribution of prices for each cancellation policy type |

The strict cancellation policy is the most frequent and is followed by moderate and flexible types. Only 15 listings have super strict policies. An interesting relationship between the cancellation policy and price can be observed in Figure 17. Boxplots illustrate how the mean price is lowest for the properties with the most flexible policies and is highest for the properties with the strictest cancellation policies.

### 2.1.5 Correlation

It is important to investigate correlations between price and other features from the dataset to determine factors that influence the price. Another interesting question is to determine how feature variables are correlated with each other. Figure 19 shows the correlation matrix for all the numerical variables.

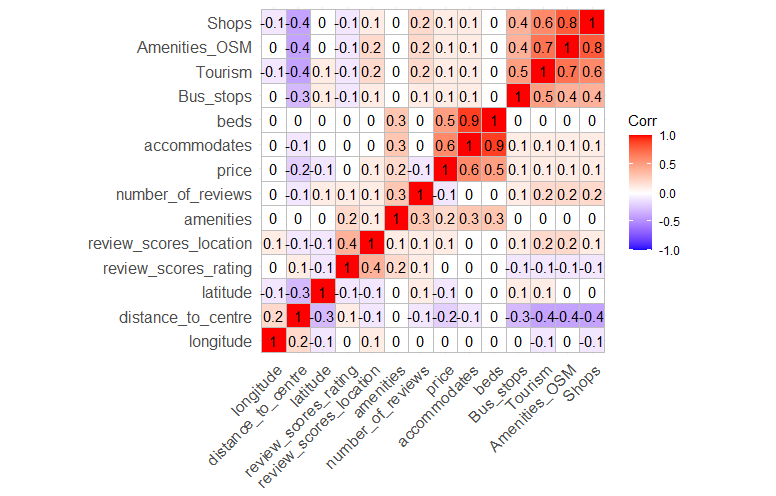


Figure - Correlation matrix

It can be noticed that features *beds* and *accommodates* have a correlation coefficient of 0.9, indicating a strong linear relationship between the two variables. This makes sense since the more beds the property has, the more guests it should accommodate. Furthermore, *price* somewhat correlates to *accommodates* (correlation coefficient 0.6). *Beds* will not be included in the machine learning models to reduce multicollinearity.

It comes as no surprise that the features which represent the amenities in the listings' vicinity have a reasonably strong correlation. The OpenStreetMap features also have a moderate negative correlation to the distance to the city centre.

## 2.2 Data preparation

As noted in the data exploration segment of the project, many missing values need to be cleaned, and some of the variables need to be transformed before the analysis can be done. The following section summarises the data cleansing effort.

*Host\_is\_superhost* is a Boolean variable, and the values are t (True) and f (False). There are 7460 regular hosts, 2023 super hosts, and two empty values. Rows with missing values will be removed.

*The neighbourhood* feature is a categorical variable and has 68 different values. 1238 rows have a missing *neighbourhood* value. The properties that have missing values are based outside the city boundaries, as shown in Figure 1. We have two options for how to deal with the missing neighbourhood values. One is to remove rows with missing values, and the other is to impute the missing values by employing reverse geocoding. Reverse geocoding can be carried out using the Nominatim service from OpenStreetMap. Further analysis will focus on the properties within the city limits, and the properties with a missing *neighbourhood* value will be removed.

*Property\_type* is a categorical variable and has 27 different values. One of the values is *other*. Airbnb asks users to select a property type during the listing creation process. Currently, the value *other* is not available on the listing creation form. Therefore it is impossible to investigate under which conditions such value could be selected. The *other* values were updated to the appropriate property type after analysing the descriptions of the related listings. The *property\_type* feature also features some unusual values. For example, there is one listing with *property\_type* value *Casa Particular (Cuba)*. A *casa particular* is a Spanish term that usually means "private home". The value was changed into a *house* after the analysis of the description field of the corresponding listing. Another listing has a property type value set as *Nature lodge,* while the description states that the property is a traditional Dublin townhouse. The value was changed accordingly. The remainder of the unusual values were investigated and updated in the same manner.

*Beds* feature is an integer variable and has 14 missing values. The properties with missing *beds* value will be removed.

*The amenities* feature is used to store all of the amenities offered by a property in a list format. This project will not analyse the impact of separate amenities but will look at the effect the total number of amenities has on the price. The example of *amenities* value:

|  |
| --- |
| **{Wifi,Washer,Dryer,"Smoke detector",Essentials}** |

The values will be split using string functions, and the values will be counted for each property.

*Review\_scores\_rating* and *review\_scores\_location* have 1557 and 1568 missing values. These are missing because there are not enough reviews. The lack of reviews means that the researcher cannot impute the missing values since the review is a subjective value and varies from person to person. The properties that have missing values will be removed from further analysis.

*Price* variables contain currency symbols, commas and '.00' at the end, as seen in an example below:

|  |
| --- |
| **$181.00** |

All of the extra symbols will be stripped using string functions. There are no missing values, but there is one property that is valued at $0.00. The property with this price will be removed from further analysis.

## 2.3 Modelling

The project's goal was to predict the price per night of Dublin Airbnb accommodations based on the Inside Airbnb and OpenStreetMap data. The researcher built three different machine learning models: Neural Network, Random Forest, and Linear Regression. The following sections describe the methods in detail.

### 2.3.1 Neural Network

An artificial neural network is a network that mimics the animal brain and utilises complex mathematical models for information processing. In any neural network, three or more layers are present: an input layer, one or more hidden layers and an output layer. Each layer consists of several neurons.

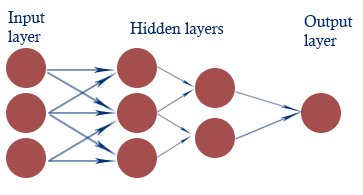


Figure - Artificial neural network

Each individual neuron acts as its own linear regression model, composed of input data, weights, threshold and an output. Once an input layer is determined, weights are assigned. These weights determine the importance of any given variable. All inputs are then multiplied by their respective weights and then summed. Afterwards, the output is passed through an activation function, which determines the output. If that output exceeds a given threshold, it activates the neuron, passing data to the next layer in the network. This results in the output of one node becoming the input of the next node (IBM, 2020).

This project uses the neuralnet package in R to create a neural network. Many different combinations of parameters were tested to improve the model's performance and accuracy with OpenStreetMap data and without. The best results were achieved with no OSM features by performing the following steps:

* The features *Shops* and *beds* have a high correlation to other features, and they were removed.
* The *price* outliers were removed. Figures 21 and 22 show the histograms of the price variable before and after the removal of the outliers.

|  |  |
| --- | --- |
| Figure - Histogram of the price feature with outliers | Figure - Histogram of the price feature without outliers |

* Yeo-Johnson transformation was applied to normalise the numerical features. The following figures show the distributions of the price variable before and after the transformation.

|  |  |
| --- | --- |
| Figure - The distribution of the price variable before the Yeo-Johnson transformation | Figure - The distribution of the price variable after the Yeo-Johnson transformation |

* Splitting the data to training (60%), validation (20%), and test (20%) sets. The validation set was used to tune the parameters.
* varImp package was used to calculate the importance of the variables.

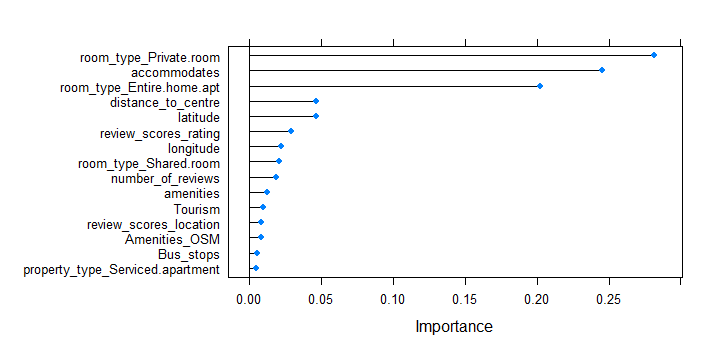


Figure - Variable importance plot

* The hyperparameters were selected by testing many different combinations. The following table shows the parameters for the best performing neural network.

|  |  |
| --- | --- |
| Parameter | Value |
| Number of top important features | 17 |
| Epochs | 1 |
| Threshold | 0.35 |
| Hidden layers | One hidden layer, five neurons |

Table - The parameters of the neural network

### 2.3.2 Linear Regression

Regression illustrates how one or more input variables together impact an output variable. Linear regression is one of the most common approaches used in predictive analytics. The primary uses for the regression analysis are (Statistics Solutions, 2013):

* Determining the strength of predictor variables
* Forecasting an effect
* Trend forecasting

A simple linear regression equation is defined by the formula *y = c + b∙x,* where *y* is the dependent variable, *c* *–* constant, *b –* regression coefficient, and *x* – explanatory variable.

Building a linear regression model is an iterative process with three stages: feature selection, assessment of the model (is it better than the previous iteration), and modification of the model by selecting a different set of features (if required).

For this project, the best Linear Regression result was achieved by performing the following steps:

* The features *Shops* and *beds* have a high correlation to other features, and they were removed.
* The *price* outliers were removed.
* Yeo-Johnson transformation was applied to normalise the numerical features.
* Splitting the data to training (60%), validation (20%), and test (20%) sets. The validation set is used to tune the parameters.
* varImp package was used to calculate the importance of the variables.
* Programmed the loop to calculate RMSE and MAE scores that adds one variable to the model at a time, with the most important coming first.

### 2.3.3 Random Forest

Random forest is a supervised learning algorithm. It uses an ensemble learning method and can be used for both regression and classification. It works by creating a specified number of decision trees. The output of the random forest depends on the type of task. It outputs the mode of the classes in classification tasks and the mean prediction of the individual decision trees in the regression tasks.

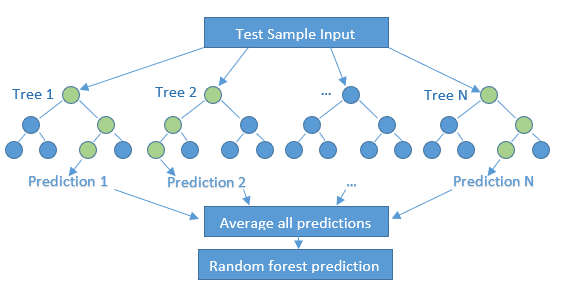


Figure - Random forest structure

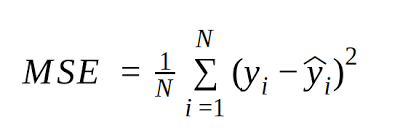
For this project, the best Random Forest result was achieved by performing the following steps:

* The features *Shops* and *beds* have a high correlation to other features, and they were removed.
* The *price* outliers were removed.
* Yeo-Johnson transformation was applied to normalise the numerical features.
* Splitting the data to training (60%), validation (20%), and test (20%) sets. The validation set is used to tune the parameters.
* varImp package was used to calculate the importance of the variables.
* Programmed the loop to calculate RMSE and MAE scores that adds one variable to the model at a time, with the most important coming first.
* Found the number of trees that produce the best RMSE and MAE scores.

## 2.4 Evaluation

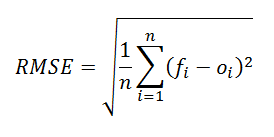
It is essential to quantify accuracy in order to compare the regression models. One of the most common methods for this task is to measure the error between the actual value and the predicted value. Researchers can derive many different metrics from the error value to measure the goodness of the model. This project will utilise Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) to provide insights and compare the models.

Mean Absolute Error is the average value of error in a set of predicted values without considering direction. It ranges from 0 to inf., and a lower value means a better model. It is mathematically described as:



MAE illustrates how big of an error a researcher can expect from the forecast on average. One problem with MAE is that the relative size of the error is not always obvious. When comparing only the MAE of two models, it can be difficult to determine which model has a significant error vs multiple minor errors.

Root Mean Squared Error is the square root of the average value of squared error in a set of predicted values. It does not consider the direction and ranges from 0 to inf. A lower value means a better model. It is mathematically described as:



RMSE's relationship to MAE can be described as MAE ≤ RMSE ≤ n1/2∙MAE, where n is the sample size. If the RMSE value is much higher than MAE, it means that the error variance is high and that the data might have outliers.

The models were trained with the OpenStreetMap features and without to investigate if these had an impact on the results. The following tables show the RMSE and MAE scores for each model after the parameter tuning stage and the final MAE score achieved with the unseen test data.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | RMSE (training) | MAE (training) | MAE (test) | Number of features |
| Linear Regression | 36.66187 | 23.81 | 24.22379 | 85 |
| Random Forest | 34.28093 | 23.05617 | 23.44632 | 72 |
| Neural Network | 34.80082 | 23.7327 | 24.00653 | 21 |

Table - The RMSE and MAE results of the machine learning models (OpenStreetMap data included)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | RMSE (training) | MAE (training) | MAE (test) | Number of features |
| Linear Regression | 35.26407 | 23.85232 | 24.21144 | 85 |
| Random Forest | 34.34483 | 23.02005 | 23.54127 | 57 |
| Neural Network | 34.63705 | 23.65134 | 24.05399 | 17 |

Table - The RMSE and MAE results of the machine learning models (OpenStreetMap data excluded)

The results illustrate that the OpenStreetMap features do not make any significant difference. Random Forest has the most significant RMSE and MAE value. With data similarly distributed to the data we have in the test set, the researcher can expect that, on average, the difference between the actual price and predicted price is 23.02.

Next, the researcher examined the actual vs predicted scatterplots. The points should be following a straight line closely at 45 degrees to represent a good fit.

|  |  |  |
| --- | --- | --- |
| Figure - Actual vs Predicted (Linear Regression) | | Figure - Actual vs Predicted (Neural Network) |
| Figure - Actual vs Predicted (Random Forest) | Figure - Residuals vs Fit (Random Forest) | |

All three of the actual vs fitted scatterplots have similar patterns. The majority of the data points are above the 450 diagonal lines at the lower end of the x-axis and below at the higher end. This indicates that all models tend to overestimate the cheaper properties and underestimate the expensive ones.

Figure 30 shows the residuals versus fit plot for the results of the Random Forest. From this, the researcher could observe that:

* Some points stand out in all the models, which means that there might be outliers in our data.
* The residuals do not appear to be random or symmetrical, which means that the error terms' variances are not equal.
* The residuals get more significant as the predictions move from small to large, which means that there is a heteroscedasticity problem.

Investigation of the plots showed that there is much room for improvement. The heteroscedasticity often can be solved by transforming variables and adding features. Outliers need to be examined in detail and then fixed or removed from the dataset.

# Conclusion

The project started with five objectives on page seven. In this chapter, the researcher presents the output of the project and conducts a discussion of the findings.

## 3.1 Discussion

The Airbnb data analysis illustrates that, on average, one can expect to pay more than twice for the properties which are close to the city centre compared to the properties that are further away. It comes as no surprise since the city centre areas offer better public transport links, many shops, restaurants and tourist attractions. These locations are also rated highest by the previous guests. When looking at the properties themselves, a host can expect to get a better return if they let private rooms or entire apartments and houses. Hosts can expect to attract a higher price if they also offer more amenities. The analysis also found that guests might expect to receive better value if they choose a host which has a super host status. A certain amount of bookings is one of the requirements to become a super host on the Airbnb platform. Hosts that charge slightly less for their properties can fulfil this requirement easier.

As for the machine learning models, the project found that the Random Forest algorithm achieved the best results when using the dataset without OpenStreetMap features. On average, the difference between the actual price and the predicted price is 23.02. All models tend to underestimate the price of accommodations that have a higher price.

## 3.2 Future work

The project revealed that there are many issues associated with the quality of the data. One of the main reasons for this issue is that hosts do not provide accurate details about their properties. A more detailed outlier analysis would help with the accuracy of the models. Other possible development opportunities:

* More data. The project did not include the properties that are outside the city boundaries. Also, the project utilised data of the listings that was compiled in October 2019. Researchers could combine the data from the whole year in a future project.
* More features. It would be interesting to investigate how the listings' price relates to the crime statistics, general property prices, and median income in the surrounding neighbourhoods. Also, the future project could include natural language processing and investigate the text descriptions and amenities of the listings.
* The model could be built into the application where a user provides the property's details and receives the predicted price.

# Reflection

The final Data Analytics project was an exciting learning experience. During the course, I learned many new skills and tools, and my main goal was to put into practice as many of them as possible. The first challenge that I encountered was the project idea which had to be unique at least to some extent. Many datasets are available online to choose from, but most of them would already be cleaned and prepared. Data cleaning and preparation is a big part of the data analysts' job, and I wanted my project to utilise messy real-world data. Also, I wanted to combine data from different sources. Travelling is my passion, and I decided to investigate Airbnb prices in Dublin with R using messy data from the Inside Airbnb website. I chose to scrape data using Python from the Openstreetmap service to add a second data source. The programming part of the project was easy because the course covered Python and R very well. I have some experience programming in other languages (e.g., C++, Java), and I was able to use that experience when working on this project.

The project involved building three machine learning models to predict Airbnb listing prices. I learned some basic machine learning methods during the course (e.g., Linear Regression, K-Means clustering, Decision Trees), but I wanted to research other algorithms. I chose to learn how to use the Neural Network and the Random Forest and successfully used said methods in my work. The project involved a very straightforward parameter tuning approach due to the scope of the research. My next step is to learn how to implement more sophisticated hyper-parameter tuning methods for Neural Networks and Random Forests.

Overall, I enjoyed the opportunity to apply the skills that I learned during the course. I used two programming languages, cleaned the data, conducted analysis, and successfully built three machine learning models. I also learned about new methods and techniques.

# References

Astrid D.A.M.Kemperman, A. W. (2009). Tourist shopping behavior in a historic downtown area. *Tourism Management (30:2)*, 208-218.

Hill, D. (2015). How much is your spare room worth? *IEEE Spectrum, vol. 52, no. 9*, 32-58.

IBM. (2020, August 17). *Neural Networks*. Retrieved from https://www.ibm.com/cloud/learn/neural-networks

MA, Y. e. (2018). Estimating Warehouse Rental Price using Machine Learning Techniques. *INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL, 13:2*, 235-250.

Murray, S. (2019, May 13). *The average cost of renting a property in Dublin is now over €2,000*. Retrieved from TheJournal.ie: https://www.thejournal.ie/daft-rent-report-4630834-May2019/

Niam Yaraghi, S. R. (2017). *The Current and Future State of the Sharing Economy.*

Robbin Deboosere, D. J.-G. (2019). Location, location and professionalization: a multilevel hedonic analysis of Airbnb listing prices and revenue. *Regional Studies, Regional Science, 6:1*, 143-156.

Statistics Solutions. (2013). *What is Linear Regression?* Retrieved from Statistics Solutions: https://www.statisticssolutions.com/what-is-linear-regression/

Y. Li, Q. P. (2016). Reasonable price recommendation on Airbnb using Multi-Scale clustering. *35th Chinese Control Conference (CCC)*, (pp. 7038-7041). Chengdu.

Yang Y, D.-R. A. (2012). Walking distance by trip purpose and population subgroups. *American Journal of Preventive Medicine, 43(1)*, 11--19.

# Appendix

## Source codes

### OpenStreetMap scraper. Python script

# -\*- coding: utf-8 -\*-

"""

Created on Wed Jan 20 14:41:56 2021

@author: Andrius Dalisanskis

Scrapes the data from the OpenStreetMap

"""

import overpy

import pandas as pd

from time import time

from time import sleep

# the radius in meters. How far away from the property we search for tags

RADIUS = 200.0

api = overpy.Overpass()

# Runs single query for the given coordinates. Function has a while loop which

# runs until we get result. The service throws an error sometimes, which needs

# to be caught. Then we wait 30 seconds and try same query again. Returns number

# of specified nodes found around the specified location.

def run\_query(lat, lon, q):

while True:

try:

r = api.query(q.format(RADIUS, lat, lon)) # send query to OSM

except overpy.exception.OverpassTooManyRequests:

sleep(30)

continue

except overpy.exception.OverpassGatewayTimeout:

sleep(30)

continue

break

return len(r.get\_nodes())

# This function takes in dataframe and query, then calls run\_query function for

# every row and saves results in a list. Returns the list of results for each

# row.

def scrape\_osm(df, q):

results = []

results = [run\_query(lat, lon, q) for

lat, lon in zip(df['latitude'], df['longitude'])]

return results

# Main function

def main():

file\_in = "data/listings\_selected\_features.csv"

file\_out = "data/listings\_plus\_osma.csv"

df = pd.read\_csv(file\_in, index\_col = 0)

queries = ["""node [amenity='bar'] (around: {0},{1},{2}); out;

node [amenity='pub'] (around: {0},{1},{2}); out;

node[amenity='cafe'] (around: {0},{1},{2}); out;

node [amenity='restaurant'] (around: {0},{1},{2}); out;

node [amenity='nightclub'] (around: {0},{1},{2}); out;""",

"""node [tourism] (around: {0},{1},{2}); out;""",

"""node [highway='bus\_stop'] (around: {0},{1},{2}); out;""",

"""node [shop] (around: {0},{1},{2}); out;"""]

results = scrape\_osm(df, queries[0])

df['Amenities\_OSM'] = results

results = scrape\_osm(df, queries[1])

df['Tourism'] = results

results = scrape\_osm(df, queries[2])

df['Bus\_stops'] = results

results = scrape\_osm(df, queries[3])

df['Shops'] = results

df.to\_csv(file\_out)

if \_\_name\_\_ == "\_\_main\_\_":

main()

### Feature selection. R script

# Loads original csv file, then selects features of interest and saves to new

# csv file

original\_file = '../data/listings\_original.csv'

new\_file = '../data/listings\_selected\_features.csv'

features <- c('host\_is\_superhost',

'neighbourhood',

'latitude',

'longitude',

'property\_type',

'room\_type',

'accommodates',

'beds',

'cancellation\_policy',

'amenities',

'price',

'number\_of\_reviews',

'review\_scores\_rating',

'review\_scores\_location'

)

listings <- read.csv(original\_file, header=TRUE, stringsAsFactors = FALSE)

listings <- listings[features]

summary(listings)

write.csv(x = listings, file = new\_file)

### Data cleansing. R script

# Loads original csv file, then selects features of interest and saves to library(stringr)

file\_in = '../data/listings\_plus\_osm.csv'

file\_out = '../data/clean\_data.csv'

df <- read.csv(file\_in, header = TRUE, stringsAsFactors = FALSE)

# draw the map of properties with missing neighbourhood data

map\_missing\_neighbourhood <-

leaflet(df[nchar(df$neighbourhood) == 0, ]) %>%

addTiles() %>%

addCircleMarkers(radius = 1,

lng = ~ longitude,

lat = ~ latitude)

map\_missing\_neighbourhood

#draw the man of properties with no missing neighbourhood data

map\_not\_missing\_neighbourhood <-

leaflet(df[nchar(df$neighbourhood) > 0, ]) %>%

addTiles() %>%

addCircleMarkers(radius = 1,

lng = ~ longitude,

lat = ~ latitude)

map\_not\_missing\_neighbourhood

summary(df$host\_is\_superhost)

# remove rows where host feature has no value

df <- df[-which(df$host\_is\_superhost == ''), ]

summary(df$neighbourhood)

df <-

df[-which(df$neighbourhood == ''), ] # remove rows with missing neighbourhood

# check the distribution of property types, then remove all rows that contain

# property types that have less than 10 occurrences

str(df$property\_type)

property\_types <- summary(df$property\_type)

property\_types <- sort(property\_types, decreasing = TRUE)

property\_types\_discard <- names(property\_types[property\_types < 10])

df <- df[-which(df$property\_type %in% property\_types\_discard), ]

# remove rows with missing beds value

df <- df[!is.na(df$beds), ]

# remove rows with missing reviews\_score\_rating and reviews\_score\_location value

df <- df[!is.na(df$review\_scores\_rating), ]

df <- df[!is.na(df$review\_scores\_location), ]

# will split the list of amenities, and then count how many amenities are

# offered by each property

df$amenities <- str\_split(df$amenities, ',')

df$amenities <- lapply(df$amenities, unlist)

df$amenities <- lapply(df$amenities, length)

df$amenities <- as.numeric(df$amenities)

# clean the price variable

df$price <- str\_replace(df$price, '[$]', '') # remove $ symbol

df$price <- str\_replace(df$price, ',', '') # remove ,

df$price <- str\_replace(df$price, '.00 ', '') # remove .00

df$price <- as.numeric(df$price)

df[df$price == 0, ]

write.csv(x = df,

file = file\_out,

row.names = FALSE)

### Data analysis. R script

library(ggplot2)

library(ggalt)

library(ggmap)

library(tidyverse)

library(hrbrthemes)

library(caret)

library(RColorBrewer)

library(viridis)

library(sf)

library(leaflet)

library(reshape2)

library(rnaturalearth)

library(rnaturalearthdata)

theme\_set(theme\_ipsum())

file\_in = '../data/clean\_data.csv'

df <- read.csv(file\_in)

summary(df)

##### Highest rated neighbourhoods #############################################

meanRatingOfNeighbourhoods <-

df %>%

group\_by(neighbourhood) %>%

summarise(mean\_rating = mean(review\_scores\_location)) %>%

arrange(desc(mean\_rating))

top5\_neighbourhoods\_br <- head(meanRatingOfNeighbourhoods, n = 5)

top5\_neighbourhoods\_br$neighbourhood <-

droplevels(top5\_neighbourhoods\_br$neighbourhood)

levels(top5\_neighbourhoods\_br$neighbourhood)

leaflet(df[df$neighbourhood %in% levels(top5\_neighbourhoods\_br$neighbourhood), ],

options = leafletOptions(zoomControl = FALSE)) %>%

addTiles() %>%

addCircleMarkers(

radius = 1,

lng = ~ longitude,

lat = ~ latitude,

color = 'blue',

label = ~ as.character(neighbourhood)

) %>%

addLegend(

position = "bottomright",

colors = c('blue', 'red'),

labels = c('The most expensive areas', 'The least expensive areas'),

opacity = 1,

title = "Neighbourhoods by the mean value of the price"

)

##### Price analysis ###########################################################

summary(df$price)

mean\_price\_by\_neighbourhood <-

df %>%

group\_by(neighbourhood) %>%

summarise(mean\_price = mean(price)) %>%

arrange(desc(mean\_price))

top10\_neighbourhoods <- head(mean\_price\_by\_neighbourhood, n = 10)

top10\_neighbourhoods$neighbourhood <-

droplevels(top10\_neighbourhoods$neighbourhood)

levels(top10\_neighbourhoods$neighbourhood)

bottom10\_neighbourhoods <- tail(mean\_price\_by\_neighbourhood, n = 10)

bottom10\_neighbourhoods$neighbourhood <-

droplevels(bottom10\_neighbourhoods$neighbourhood)

levels(bottom10\_neighbourhoods$neighbourhood)

leaflet(df[df$neighbourhood %in% levels(top10\_neighbourhoods$neighbourhood), ],

options = leafletOptions(zoomControl = FALSE)) %>%

addTiles() %>%

addCircleMarkers(

radius = 1,

lng = ~ longitude,

lat = ~ latitude,

color = 'blue',

label = ~ as.character(neighbourhood)

) %>%

addCircleMarkers(

data = df[df$neighbourhood %in%

levels(bottom10\_neighbourhoods$neighbourhood), ],

radius = 1,

lng = ~ longitude,

lat = ~ latitude,

color = 'red',

label = ~ as.character(neighbourhood)

) %>%

addLegend(

position = "bottomright",

colors = c('blue', 'red'),

labels = c('The most expensive areas', 'The least expensive areas'),

opacity = 1,

title = "Neighbourhoods by the mean value of the price"

)

ggplot(df[df$neighbourhood %in% levels(top10\_neighbourhoods$neighbourhood), ]) +

geom\_boxplot(aes(y = price), fill = 'lightblue')

ggplot(df[df$neighbourhood %in% levels(bottom10\_neighbourhoods$neighbourhood), ]) +

geom\_boxplot(aes(y = price), fill = 'lightblue')

#### Analysis of the feature variables #########################################

# host\_is\_superhost barplot

ggplot(df) +

geom\_histogram(aes(x = host\_is\_superhost,

fill = host\_is\_superhost),

stat = 'count') +

geom\_text(stat = 'count',

aes(

x = host\_is\_superhost,

label = after\_stat(count),

vjust = -0.2

)) +

theme(legend.position = 'none') +

xlab('Host status') +

ylab('Count') +

scale\_x\_discrete(labels = c('Regular host', 'Super host'))

ggplot(df) +

geom\_boxplot(aes(x = host\_is\_superhost, y = price),

fill = c('#f8766d', '#00bfc4')) +

xlab('Host status') +

ylab('Price') +

scale\_x\_discrete(labels = c('Regular host', 'Super host'))

df$host\_is\_superhost[df$host\_is\_superhost == 'f'] <- 'Regular host'

df$host\_is\_superhost[df$host\_is\_superhost == 't'] <- 'Super host'

ggplot(df) +

geom\_bar(

aes(room\_type, price, fill = as.factor(host\_is\_superhost)),

position = "dodge",

stat = "summary",

fun.y = "mean"

) +

labs(y = "Mean price", x = "Room type", fill = "Host status\n")

# analysis of the neighbourhood variable

neighbourhood\_counts <-

df %>% count(neighbourhood) %>% arrange(desc(n))

ggplot(neighbourhood\_counts) +

geom\_histogram(aes(

y = reorder(neighbourhood, n),

fill = neighbourhood,

x = n

),

stat = 'identity') +

geom\_text(stat = 'count',

aes(y = neighbourhood,

label = n,

hjust = 1.0)) +

theme(legend.position = 'none') +

ylab('Neighbourhood') +

xlab('Count')

meanRatingOfNeighbourhoods <-

df %>%

group\_by(neighbourhood) %>%

summarise(mean\_rating = mean(review\_scores\_location)) %>%

arrange(desc(mean\_rating))

# analysis of the property\_type variable

sort(summary(df$property\_type), decreasing = TRUE)

ggplot(df) +

geom\_boxplot(aes(y = property\_type, x = price), fill = 'lightblue') +

xlab('Price') +

ylab('Property type')

property\_type\_counts <-

df %>% count(property\_type) %>% arrange(desc(n))

ggplot(property\_type\_counts) +

geom\_histogram(aes(

y = reorder(property\_type, n),

fill = property\_type,

x = n

),

stat = 'identity') +

geom\_text(stat = 'count',

aes(y = property\_type,

label = n,

hjust = 1.0)) +

theme(legend.position = 'none') +

ylab('Property type') +

xlab('Count')

# analysis of the property\_type variable

sort(summary(df$property\_type), decreasing = TRUE)

ggplot(df) +

geom\_boxplot(aes(y = property\_type, x = price), fill = 'lightblue') +

xlab('Price') +

ylab('Property type')

property\_type\_counts <-

df %>% count(property\_type) %>% arrange(desc(n))

ggplot(property\_type\_counts) +

geom\_histogram(aes(

y = reorder(property\_type, n),

fill = property\_type,

x = n

),

stat = 'identity') +

geom\_text(stat = 'count',

aes(y = property\_type,

label = n,

hjust = 1.0)) +

theme(legend.position = 'none') +

ylab('Property type') +

xlab('Count')

# analysis of the room\_type variable

room\_type\_counts <- df %>% count(room\_type) %>% arrange(desc(n))

ggplot(df) +

geom\_boxplot(aes(x = room\_type, y = price, fill = room\_type)) +

xlab('Room type') +

ylab('Price') +

scale\_x\_discrete(guide = guide\_axis(angle = 45)) +

theme(legend.position = 'none')

ggplot(room\_type\_counts) +

geom\_histogram(aes(

x = reorder(room\_type, -n),

fill = room\_type,

y = n

),

stat = 'identity') +

geom\_text(stat = 'count',

aes(x = room\_type,

label = n,

vjust = -1)) +

theme(legend.position = 'none') +

ylab('Count') +

xlab('Room type') +

scale\_x\_discrete(guide = guide\_axis(angle = 45))

ggplot(df[df$room\_type == 'Private room', ]) +

geom\_bar(aes(x = as.factor(property\_type), fill = as.factor(property\_type))) +

scale\_x\_discrete(guide = guide\_axis(angle = 45)) +

geom\_text(stat = 'count',

aes(

x = as.factor(property\_type),

label = after\_stat(count),

vjust = -1

)) +

theme(legend.position = 'none') +

ylab('Count') +

xlab('Property type') +

ylim(0, 1400)

# analysis of the cancellation\_policy variable

cancellation\_policy\_counts <-

df %>% count(cancellation\_policy) %>% arrange(desc(n))

ggplot(df) +

geom\_boxplot(aes(x = cancellation\_policy, y = price, fill = cancellation\_policy)) +

xlab('Cancellation policy') +

ylab('Price') +

scale\_x\_discrete(guide = guide\_axis(angle = 45)) +

theme(legend.position = 'none')

ggplot(cancellation\_policy\_counts) +

geom\_histogram(aes(

x = reorder(cancellation\_policy, -n),

fill = cancellation\_policy,

y = n

),

stat = 'identity') +

geom\_text(stat = 'count',

aes(x = cancellation\_policy,

label = n,

vjust = -1)) +

theme(legend.position = 'none') +

ylab('Count') +

xlab('Cancellation policy') +

scale\_x\_discrete(guide = guide\_axis(angle = 45))

# continuous variables / boxplots

p1 <- geom\_boxplot(fill = 'lightblue')

ggplot(df, aes(x = accommodates)) + p1

ggplot(df, aes(x = amenities)) + p1

ggplot(df, aes(x = number\_of\_reviews)) + p1

ggplot(df, aes(x = review\_scores\_rating)) + p1

ggplot(df, aes(x = review\_scores\_location)) + p1

ggplot(df, aes(x = Bus\_stops)) + p1

ggplot(df, aes(x = Tourism)) + p1

ggplot(df, aes(x = Amenities\_OSM)) + p1

ggplot(df, aes(x = Shops)) + p1

# new feature - distance to city centre in km

library(geosphere)

dublin\_coords <- c(-6.2603, 53.3498)

temp <- distm(x = dublin\_coords,

y = df[, c('longitude', 'latitude')],

fun = distHaversine)

df$distance\_to\_centre <- temp[1, ]

df$distance\_to\_centre <- df$distance\_to\_centre / 1000

df$distance\_to\_centre <- round(df$distance\_to\_centre, 2)

# correlation matrix

library(ggcorrplot)

cont\_vars <- c(

'accommodates',

'amenities',

'number\_of\_reviews',

'review\_scores\_rating',

'review\_scores\_location',

'Bus\_stops',

'Tourism',

'Amenities\_OSM',

'Shops',

'price',

'latitude',

'longitude',

'beds',

'distance\_to\_centre'

)

corr <- round(cor(df[, cont\_vars]), 2)

ggcorrplot(corr, hc.order = TRUE, lab = TRUE)

ggplot(df[df$neighbourhood %in% levels(top10\_neighbourhoods$neighbourhood), ]) +

geom\_boxplot(aes(y = price))

ggplot(df[df$neighbourhood %in% levels(bottom10\_neighbourhoods$neighbourhood), ]) +

geom\_boxplot(aes(y = price))

### Machine learning. R script

library(randomForest)

######################## Load the dataset ######################################

file\_in = '../data/clean\_data.csv'

df <- read.csv(file\_in)

#remove highly correlated features because of covariance

drops <- c("beds", "Shops")

#remove comment to remove the Openstreetmap features as well

#drops <- c("beds","Shops","Bus\_stops","Tourism","Amenities\_OSM")

df <- df[,!(names(df) %in% drops)]

summary(df)

# remove the price outliers

Q <- quantile(df$price, probs = c(.25, .75), na.rm = T)

iqr <- IQR(df$price, na.rm = T)

df <-

df %>% filter(price > (Q[1] - 1.5 \* iqr) &

price < (Q[2] + 1.5 \* iqr))

######################## One hot encoding ######################################

features\_cat <-

c(

"host\_is\_superhost",

"neighbourhood",

"property\_type",

"room\_type",

"cancellation\_policy"

)

df <-

one\_hot(as.data.table(df),

cols = features\_cat,

dropUnusedLevels = TRUE)

df <- df %>% rename\_all(make.names) #fix columns names

######################## Scale data (Yeo-Johnson) ##############################

scaled\_yj\_obj <- lapply(df, as.numeric)

scaled\_yj\_obj <- lapply(scaled\_yj\_obj, yeojohnson)

df\_scaled <- predict(scaled\_yj\_obj)

df\_scaled <- as.data.frame(df\_scaled)

######################## Random sampling (split 60-20-20) ######################

split\_to\_sets <- function(x) {

set.seed(80)

spec = c(train = .6,

test = .2,

validate = .2)

g = sample(cut(seq(nrow(df)),

nrow(df) \* cumsum(c(0, spec)),

labels = names(spec)))

return(split(x, g))

}

dfo <-

split\_to\_sets(df) # split the original data to train/validate/test sets

dfs <-

split\_to\_sets(df\_scaled) # split the scaled data to train/validate/test sets

######################## Feature importance ####################################

control <- trainControl(method = "repeatedcv",

number = 10,

repeats = 3)

model <- NULL

model <-

train(price ~ .,

data = dfs$train,

method = "xgbTree",

trControl = control)

importance <- varImp(model, scale = FALSE)

rownames(importance$importance)[c(1:15)]

plot(importance, top = 15)

######################## Random forest #########################################

fRandomForest <- function(nFeatures, nTrees, d) {

set.seed(2)

trainFormula <-

as.formula(paste('price', paste(

rownames(importance$importance)[c(1:nFeatures)], collapse =

' + '

), sep = ' ~ '))

rfModel <-

randomForest(

trainFormula,

data = dfs$train,

na.action = na.omit,

ntree = nTrees

)

rfPredict <- predict(rfModel, d[, -which(names(d) %in% 'price')])

return(predict(

scaled\_yj\_obj$price,

newdata = rfPredict,

inverse = TRUE

))

}

######################## Linear Regression #####################################

fLinearRegression <- function(nFeatures, d) {

set.seed(2)

trainFormula <-

as.formula(paste('price', paste(

rownames(importance$importance)[c(1:nFeatures)], collapse =

' + '

), sep = ' ~ '))

lrModel <- lm(trainFormula, data = dfs$train)

lrPredict <- predict(lrModel, d[, -which(names(d) %in% 'price')])

return(predict(

scaled\_yj\_obj$price,

newdata = lrPredict,

inverse = TRUE

))

}

######################## Neural Network ########################################

fNeuralNetwork <- function(nFeatures, h, e, t, d) {

set.seed(2)

trainFormula <-

as.formula(paste('price', paste(

rownames(importance$importance)[c(1:nFeatures)], collapse =

' + '

), sep = ' ~ '))

nnModel <-

neuralnet(

trainFormula,

data = dfs$train,

hidden = h,

rep = e,

threshold = t

)

nnPredict <- NULL

nnPredict <-

tryCatch(

neuralnet::compute(nnModel, d[, -which(names(d) %in% 'price')]),

error = function(e)

NULL)

if (is.null(nnPredict)) {

cat('FAIL-')

return(NULL)

}

else{

return(predict(

scaled\_yj\_obj$price,

newdata = nnPredict$net.result,

inverse = TRUE

))

}

}

######################## Helper functions ######################################

fRMSE <- function(obs, pred) {

return((sum((obs - pred) ^ 2) / length(obs)) ^ 0.5)

}

fMAE <- function(obs, pred) {

return(mean(abs(obs - pred)))

}

fActualVsPredictedPlot <- function(pred) {

ggplot() +

geom\_point(aes(x = dfo$test$price, y = pred),

color = 'blue',

alpha = 0.5) +

geom\_abline(slope = 1) +

xlab('Real price') +

ylab('Predicted price') +

xlim(c(0, 300)) +

ylim(c(0, 300))

}

######################## Random Forest Training ###############################

minMAE <- 99999

minRMSE <- 99999

minFeats <- 0

lMAE <- NULL

lRMSE <- NULL

lFeats <- NULL

for (i in c(1:length(rownames(importance$importance)))) {

rfRes <- fRandomForest(i, 64, dfs$validate)

tempRMSE <- fRMSE(dfo$validate$price, rfRes)

tempMAE <- fMAE(dfo$validate$price, rfRes)

cat('No of features:', i, 'RMSE: ', tempRMSE, 'MAE:', tempMAE, '\n')

if (tempMAE < minMAE) {

minMAE <- tempMAE

minRMSE <- tempRMSE

minFeats <- i

}

lMAE <- c(lMAE, tempMAE)

lRMSE <- c(lRMSE, tempRMSE)

lFeats <- c(lFeats, i)

}

cat('Min RMSE:',

minRMSE,

'Min MAE:',

minMAE,

'No of features:',

minFeats)

ggplot() +

geom\_line(aes(y = lMAE, x = lFeats, color = 'red')) +

geom\_line(aes(y = lRMSE, x = lFeats, color = 'blue')) +

geom\_point(aes(y = minMAE, x = minFeats, color = 'red'), size = 5) +

geom\_point(aes(y = minRMSE, x = minFeats, color = 'blue'), size = 5) +

xlab('Number of features') +

ylab('Value') +

scale\_color\_identity(

name = "Metric",

breaks = c("red", "blue"),

labels = c("MAE", "MRSE"),

guide = "legend"

)

minTrees <- 9999

bestPrediction <- NULL

minMAE <- 99999

minRMSE <- 99999

lMAE <- NULL

lRMSE <- NULL

lTrees <- NULL

for (i in seq(32, 128, by = 1)) {

rfRes <- fRandomForest(minFeats, i, dfs$validate)

tempRMSE <- fRMSE(dfo$validate$price, rfRes)

tempMAE <- fMAE(dfo$validate$price, rfRes)

cat('Features:',

minFeats,

'Trees:',

i,

'RMSE: ',

tempRMSE,

'MAE:',

tempMAE,

'\n')

if (tempMAE < minMAE) {

minMAE <- tempMAE

minRMSE <- tempRMSE

minTrees <- i

bestPrediction <- rfRes

}

lMAE <- c(lMAE, tempMAE)

lRMSE <- c(lRMSE, tempRMSE)

lTrees <- c(lTrees, i)

}

cat('No of features:',

minFeats,

'Trees:',

minTrees,

'RMSE: ',

tempRMSE,

'MAE:',

minMAE,

'\n')

rfRes <- fRandomForest(57, 88, dfs$test)

(tempRMSE <- fRMSE(dfo$test$price, rfRes))

(tempMAE <- fMAE(dfo$test$price, rfRes))

fActualVsPredictedPlot(rfRes)

ggplot() +

geom\_point(aes(x = rfRes, y = (dfo$test$price - rfRes)),

color = 'blue',

alpha = 0.5,

size = 1.5) +

geom\_abline(slope = 0) +

xlab('Fitted value') +

ylab('Residuals') +

xlim(c(0, 225))

######################## Neural Network Training ##############################

minMAE <- 99999

minRMSE <- 99999

bestPrediction <- NULL

minFeats <- 0

lMAE <- NULL

lRMSE <- NULL

lFeats <- NULL

for (i in c(1:length(rownames(importance$importance)))) {

nnRes <- fNeuralNetwork(i, c(5), 1, 0.35, dfs$validate)

if (!is.null(nnRes)) {

tempRMSE <- fRMSE(dfo$validate$price, nnRes)

tempMAE <- fMAE(dfo$validate$price, nnRes)

} else{

tempRMSE <- 999

tempMAE <- 999

}

cat('No of features:', i, 'RMSE: ', tempRMSE, 'MAE:', tempMAE, '\n')

if (tempMAE < minMAE) {

minMAE <- tempMAE

minRMSE <- tempRMSE

minFeats <- i

bestPrediction <- rfRes

}

lMAE <- c(lMAE, tempMAE)

lRMSE <- c(lRMSE, tempRMSE)

lFeats <- c(lFeats, i)

}

cat('Min RMSE:',

minRMSE,

'Min MAE:',

minMAE,

'No of features:',

minFeats)

nnRes <- fNeuralNetwork(17, c(5), 1, 0.35, dfs$test)

(tempRMSE <- fRMSE(dfo$test$price, nnRes))

(tempMAE <- fMAE(dfo$test$price, nnRes))

fActualVsPredictedPlot(nnRes)

######################## Linear Regression Training ###########################

minMAE <- 99999

minRMSE <- 99999

minFeats <- 0

lMAE <- NULL

lRMSE <- NULL

lFeats <- NULL

for (i in c(1:length(rownames(importance$importance)))) {

lrRes <- fLinearRegression(i, dfs$validate)

tempRMSE <- fRMSE(dfo$validate$price, lrRes)

tempMAE <- fMAE(dfo$validate$price, lrRes)

cat('No of features:', i, 'RMSE: ', tempRMSE, 'MAE:', tempMAE, '\n')

if (tempMAE < minMAE) {

minMAE <- tempMAE

minRMSE <- tempRMSE

minFeats <- i

}

lMAE <- c(lMAE, tempMAE)

lRMSE <- c(lRMSE, tempRMSE)

lFeats <- c(lFeats, i)

}

cat('Min RMSE:',

minRMSE,

'Min MAE:',

minMAE,

'No of features:',

minFeats)

lrRes <- fLinearRegression(85, dfs$test)

(tempRMSE <- fRMSE(dfo$test$price, lrRes))

(tempMAE <- fMAE(dfo$test$price, lrRes))

fActualVsPredictedPlot(lrRes)