

**School of InfoComm Technology**

**Machine Learning**

Diploma in Data Science (DS)

Diploma in Information Technology (IT)

October 2022 Semester

**INDIVIDUAL ASSIGNMENT 1**

(30% of Machine Learning Module)

**Deadline for Submission:**

**17th Dec 2022 (Saturday), 2359 Hours**

|  |  |  |
| --- | --- | --- |
| Student Name | : | Lim Wee Liang Kelven |
| Student Number | : | S10221788K |
| Video Presentation Link | : | <https://youtu.be/pLwhdRoc0TU> |

**Penalty for late submission:**

10% of the marks will be deducted every day after the deadline.

**NO** submission will be accepted after 24th Dec 2022, 23:59.

Contents

[Overview 3](#_Toc122088317)

[HR Analytics 3](#_Toc122088318)

[Problem Understanding 3](#_Toc122088319)

[Data Exploration 3](#_Toc122088320)

[Data Cleansing and Transformation 4](#_Toc122088321)

[Dropping Redundant Columns 4](#_Toc122088322)

[Removing Null Values 4](#_Toc122088323)

[One-hot Encoding (OHE) and Mapping 5](#_Toc122088324)

[Stratified Sampling 10](#_Toc122088325)

[Train-test Split 11](#_Toc122088326)

[Check for Outliers 11](#_Toc122088327)

[Standardising data 13](#_Toc122088328)

[Correlation Analysis 14](#_Toc122088329)

[Airbnb 16](#_Toc122088330)

[Problem Understanding 16](#_Toc122088331)

[Data Exploration 17](#_Toc122088332)

[Data Cleansing and Transformation 18](#_Toc122088333)

[Dropping Redundant Columns 18](#_Toc122088334)

[Removing Null Values 18](#_Toc122088335)

[One-hot Encoding (OHE) and Mapping 18](#_Toc122088336)

[Train-test Split 22](#_Toc122088337)

[Check for Outliers 22](#_Toc122088338)

[Standardising data 25](#_Toc122088339)

[Correlation Analysis 26](#_Toc122088340)

[Summary and Further Improvements 27](#_Toc122088341)

[Summary 27](#_Toc122088342)

[HR Analytics 27](#_Toc122088343)

[Airbnb 28](#_Toc122088344)

[Further Improvements 28](#_Toc122088345)

# Overview

This report contains the documentation of the analysis and findings from two datasets: hr\_data.csv from HR Analytics, and listings.csv from Airbnb. The objectives of this report are:

* To conduct data preparation, exploration, and analysis through visualisation and statistical methods
* To prepare the data ready for machine learning

This report will also explain problems for each dataset, the steps taken, and solutions used to modify the datasets.

Some of the approaches and explanations used in HR Analytics are also used in Airbnb. Hence, some explanations will be omitted when going through the Airbnb section to avoid repetition.

# HR Analytics

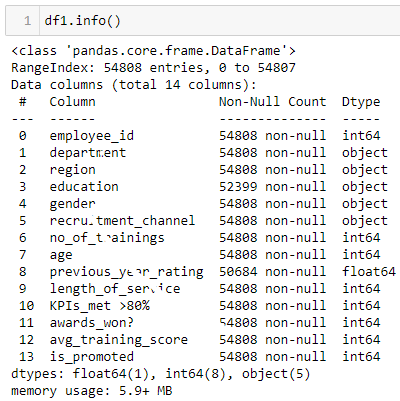
## Problem Understanding

The company, HR Analytics, wants to identify employees who are most likely to get promoted efficiently.

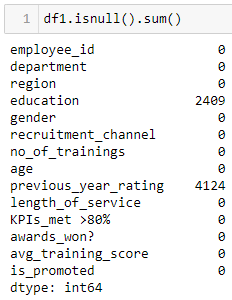
## Data Exploration

The data dictionary provided shows the names and descriptions of the columns. An important feature to observe is that the target column, “is\_promoted”, has boolean values, with 0 representing no promotion and 1 representing a promotion. Hence, a logistic regressor is needed.

Upon loading the data, the *.info()* function shows how many rows and columns are present and the data types for each column. For now, there are 54808 rows and 14 columns.



The *.isnull()* and *.sum()* functions identify which columns contain nulls and calculate how many nulls are present in the respective column. From here, it shows that the columns “education” and “previous\_year\_rating” have 2409 and 4124 nulls respectively.



## Data Cleansing and Transformation

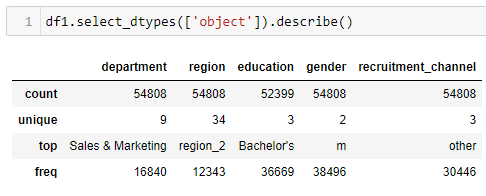
### Dropping Redundant Columns

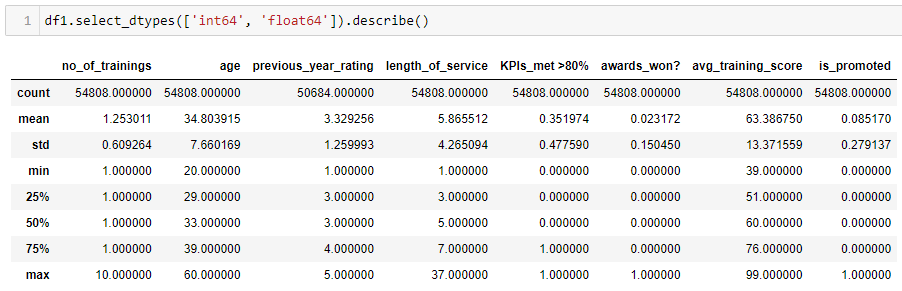
“employee\_id” is first dropped as it is unnecessary for analysis. The code used is shown below.

# Drop employee\_id column as it is unnecessary for analysis  
df1 = df1.drop('employee\_id', axis = 1)

### Removing Null Values

Before removing the null values from “education” and “previous\_year\_rating”, the statistical breakdown of their columns is needed. The codes and statistics for “education” and “previous\_year\_rating” are shown below.

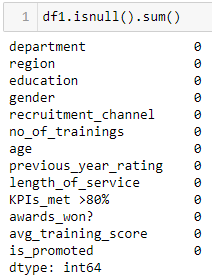




It shows that the most frequent value for “education” is “Bachelor’s” and the mean of “previous\_year\_rating” is 3.32. For “education”, the null values will be replaced with “Bachelor’s” while those in “previous\_year\_rating” will be replaced with the median value instead of the mean because the median is unaffected by extreme values. The codes used for the replacement are shown below.

# Replace nulls in education with the most frequent  
df1['education'] = df1['education'].fillna(“Bachelor's”)  
  
# Replace nulls in previous\_year\_rating with median  
df1['previous\_year\_rating'] = df1['previous\_year\_rating'].fillna(df1['previous\_year\_rating'].median()).astype(int)

The *.isnull()* and *.sum()* functions are used again to check if the removals were successful.

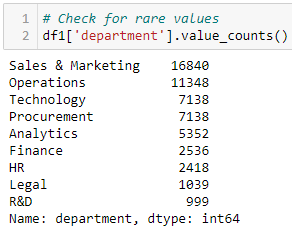


### One-hot Encoding (OHE) and Mapping

There are five categorical columns: “department”, “region”, “education”, “gender”, and “recruitment\_channel”. Their string values will be replaced by integers for the machine-learning model.

#### department

The *.value\_counts()* function checks for rare values by calculating the number of unique values present and how often they appear.

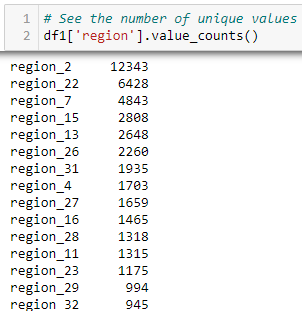


It shows that there are no rare values. Since there are more than five unique values, normal mapping will be used instead of OHE. The code is shown below.

Graphical user interface, text, application

Description automatically generated

#### region



Since there are values that appear less than 500 times, they will be considered rare. The code used to replace the values with “rare” and the subsequent mapping will be shown together.

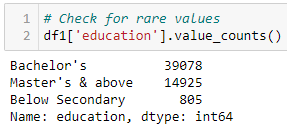
# Replace values with <500 count  
# Replace region\_3, region\_9, region\_12, region\_18, region\_21, region\_33, region\_34 with rare  
df1['region'] = df1['region'].replace(['region\_12', 'region\_9', 'region\_21','region\_3', 'region\_34', 'region\_33', 'region\_18'], 'rare')

# Map values to a number  
df1['region'] = df1['region'].map({'region\_1': 0, 'region\_2': 1, 'region\_4': 2, 'region\_5': 3, 'region\_6': 4, 'region\_7': 5, 'region\_8': 6, 'region\_10': 7, 'region\_11': 8, 'region\_13': 9, 'region\_14': 10, 'region\_15': 11, 'region\_16': 12, 'region\_17': 13, 'region\_19': 14, 'region\_20': 15, 'region\_22': 16, 'region\_23': 17, 'region\_24': 18, 'region\_25': 19, 'region\_26': 20, 'region\_27': 21, 'region\_28': 22, 'region\_29': 23, 'region\_30': 24, 'region\_31': 25, 'region\_32': 26, 'rare': 27}).astype(int)

Table

Description automatically generated

#### education



Since only three unique values are present, OHE will be used. The code used to create the encoder, fit the encoder, and transform the dataset will be shown.

# Create encoder  
education\_enc = ohe()  
  
# Create variable to store fitted and transformed encoder  
temp = education\_enc.fit\_transform(df1[['education']]).toarray()  
  
# Get labels  
labels = education\_enc.categories\_  
  
# Create 'features' dataframe  
features = pd.DataFrame(temp, columns = labels)  
  
# Concatenate 'features' dataframe with df\_copy1  
df1\_enc = pd.concat([df1, features], axis = 1)  
  
# Drop 'education' column  
df1\_enc = df1\_enc.drop('education', axis = 1)  
  
df1\_enc.head()

Table

Description automatically generated

#### gender

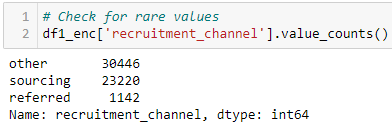
OHE will also be used for “gender”

# Create encoder  
gender\_enc = ohe()  
  
# Create variable to store fitted and transformed encoder  
temp = gender\_enc.fit\_transform(df1[['gender']]).toarray()  
  
# Get labels  
labels = gender\_enc.categories\_  
  
# Create 'features' dataframe  
features = pd.DataFrame(temp, columns = labels)  
  
# Concatenate 'features' dataframe with df\_copy1  
df1\_enc = pd.concat([df1\_enc, features], axis = 1)  
  
# Drop 'education' column  
df1\_enc = df1\_enc.drop('gender', axis = 1)  
  
df1\_enc.head()

A screenshot of a cell phone

Description automatically generated with low confidence

#### recruitment\_channel



OHE will also be used for “recruitment\_channel”

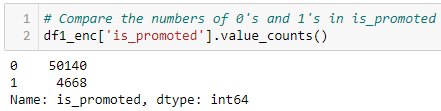
# Create encoder  
recruitment\_enc = ohe()  
  
# Create variable to store fitted and transformed encoder  
temp = recruitment\_enc.fit\_transform(df1[['recruitment\_channel']]).toarray()  
  
# Get labels  
labels = recruitment\_enc.categories\_  
  
# Create 'features' dataframe  
features = pd.DataFrame(temp, columns = labels)  
  
# Concatenate 'features' dataframe with df\_copy1  
df1\_enc = pd.concat([df1\_enc, features], axis = 1)  
  
# Drop 'education' column  
df1\_enc = df1\_enc.drop('recruitment\_channel', axis = 1)  
  
df1\_enc.head()

Table

Description automatically generated

### Stratified Sampling

Because this is a classifier problem, the number of 0s and 1s in the “is\_promoted” column must be equal. The *.value\_counts()* function checks the numbers 0s and 1s.



Next, temporary dataframes are created to store both values separately. The code used is shown.

# Create a dataframe that filters out rows with 1's  
df1\_sampled = df1\_enc[df1\_enc['is\_promoted'] == 1]  
  
# Create a dataframe that filters out rows with 0's  
df0 = df1\_enc[df1\_enc['is\_promoted'] == 0]

Graphical user interface, text, application

Description automatically generated

The code above shows that df1\_sampled has 4668 rows and df0 has 50140 rows. Since df0 has more rows, 4668 rows will be randomly sampled from it. A new dataframe, df0\_sampled, will store the sampled rows. Then, df1\_sampled and df0\_sampled will be combined to create a new dataframe. The code used for sampling and concatenating is shown below.

# Create a dataframe that filters out rows with 0's  
df0\_sampled = df0.sample(n = len(df1\_sampled), random\_state = 0).copy()  
  
# Create a new dataframe that has equal numbers of 0's and 1's  
df1\_new = pd.concat([df1\_sampled, df0\_sampled], axis = 0)  
  
# Show the number of 1’s and 0’s  
df1\_new['is\_promoted'].value\_counts()

A picture containing shape

Description automatically generated

### Train-test Split

After creating the sampled dataframe, it will be split into a training set to build the model and a testing set to test the accuracy of the model. The code used is shown below.

# Split both Inputs (X) and Output (y) into training set (70%) and testing set (30%)  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(df1\_new.drop(['is\_promoted'], axis = 1), df1\_new['is\_promoted'], test\_size = 0.3, random\_state = 10)

### Check for Outliers

Next, X\_train will be checked for any outliers. A method will be created that will be used for both datasets. The code used is shown below.

# Create method to plot boxplot and histogram to see data distribution  
**def** **boxplot\_kdeplot**(df, column):  
 plt.figure(figsize = (14, 4))  
 plt.subplot(1, 2, 1)  
 sns.boxplot(data = df, x = column)  
 plt.subplot(1, 2, 2)  
 sns.kdeplot(data = df, x = column, shade = True)

Chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated

It shows that “age” and “length\_of\_service” have outliers while “avg\_training\_score” does not. As such, “avg\_training\_score” will not be changed. To remove the outliers, the YeoJohnsonTransformer from Feature Engine will be used. This transformer also aids in making the data more gaussian-like. The code used to trim the outliers is shown below.

# Create list to store column names  
col = ['age', 'length\_of\_service']  
  
# initialize the transformer with a subset of variables to transform  
yjt = YeoJohnsonTransformer(variables = col)  
  
# fit transformer to the dataframe  
yjt.fit(X\_train)  
  
# transform indicated variables  
X\_train\_tf = yjt.transform(X\_train)

After transforming, the image below shows what the new column values look like.

Chart, histogram

Description automatically generated

### Standardising data

All the columns are numeric but do not have the same units. To standardise the scale of all the columns, a min-max scaler will be used. This ensures the values fall between the range of -1 to 1. Before standardising, the values are plotted to show a before-and-after change. A method is created to plot the data and the code used is shown below.

**def** **PlotScale**(data):  
 # Set plot size  
 fig, (ax1) = plt.subplots(figsize = (9, 6))  
   
 # Plot values  
 **for** column **in** data:  
 sns.kdeplot(data[column], ax = ax1, label = column)  
   
 # Show legend  
 ax1.legend()  
   
 plt.show()

The image below shows the data before standardising.

Graphical user interface

Description automatically generated with low confidence

The image below shows the code used to standardise the data and the data after standardising.

Chart

Description automatically generated

## Correlation Analysis

For correlation analysis, a model is used from statsmodel.api. The model is created using the Ordinary Least Squares (OLS) method and identifies which columns have a high and low significance. The code used to create the model is shown below.

# Create a model using statsmodel.api: the Ordinary Least Squares (OLS) method and fit function  
ols = sm.OLS(y\_train, X\_train\_scaled).fit()  
  
# Summary statistics from the model  
ols.summary()

Table

Description automatically generated

From the image above, it shows that the columns “no\_of\_trainings”, “length\_of\_service”, “Bachelor's”, “Below Secondary”, “Master's & above”, “f”, “m”, “other”, “referred”, and “sourcing” have P>|t| values more than 0.05. This implies that they have a low significance level and are removable without affecting the final predictive model too much.

The code used to remove those columns, and re-transform the data is shown below.

# Remove columns with P>|t| values more than 0.05  
# no\_of\_trainings, length\_of\_service, Bachelor's, Below Secondary, Master's & above, f, m, other, referred, sourcing  
df1\_final = df1\_new.drop(['no\_of\_trainings', 'length\_of\_service', "Bachelor's", 'Below Secondary', "Master's & above", 'f', 'm', "other", "referred", "sourcing"], axis = 1)  
  
# Split both Inputs (X) and Output (y) into training set (70%) and testing set (30%)  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(df1\_final.drop(['is\_promoted'], axis = 1), df1\_final['is\_promoted'], test\_size = 0.3, random\_state = 10)  
  
# Create list to store column names  
col = ['age', 'avg\_training\_score']  
  
# initialize the transformer with a subset of variables to transform  
yjt = YeoJohnsonTransformer(variables = col)  
  
# fit transformer to the dataframe  
yjt.fit(X\_train)  
  
# transform indicated variables  
X\_train\_tf = yjt.transform(X\_train)

# Scale the data  
X\_train\_scaled = (X\_train\_tf - X\_train\_tf.mean()) / (X\_train\_tf.max() - X\_train\_tf.min())

The OLS model is created again using the same code, and the table is shown below.

Text

Description automatically generated

Finally, the X\_train, X\_test, y\_train, and y\_test sets are exported. The code used is shown below.

hr\_final\_Xtrain = X\_train\_scaled  
hr\_final\_Xtest = X\_test  
hr\_final\_ytrain = y\_train  
hr\_final\_ytest = y\_test  
  
hr\_final\_Xtrain.to\_csv('hr\_final\_Xtrain.csv', index = False, sep = ',', encoding = 'utf-8')  
hr\_final\_Xtest.to\_csv('hr\_final\_Xtest.csv', index = False, sep = ',', encoding = 'utf-8')  
hr\_final\_ytrain.to\_csv('hr\_final\_ytrain.csv', index = False, sep = ',', encoding = 'utf-8')  
hr\_final\_ytest.to\_csv('hr\_final\_ytest.csv', index = False, sep = ',', encoding = 'utf-8')

# Airbnb

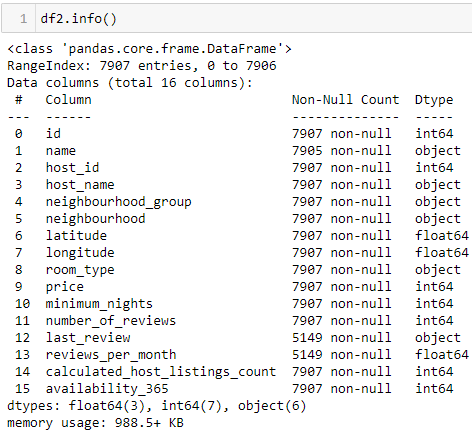
## Problem Understanding

The company, Airbnb, wants to predict the rental prices of listed properties.

## Data Exploration

The data dictionary provided shows the names and descriptions of the columns. An important feature is that the target column, “price”, has continuous values. Hence, a linear regressor is needed.

Like HR Analytics, the *.info()* function analyses how many rows and columns are present and the data types for each column. For now, there are 7906 rows and 16 columns.



The *.isnull()* and *.sum()* functions identify which columns contain nulls and calculate how many nulls are present in the respective column. From here, it shows that the columns “name”, “last\_review” and “reviews\_per\_month” have 2, 2758, and 2758 nulls respectively.

Text

Description automatically generated

## Data Cleansing and Transformation

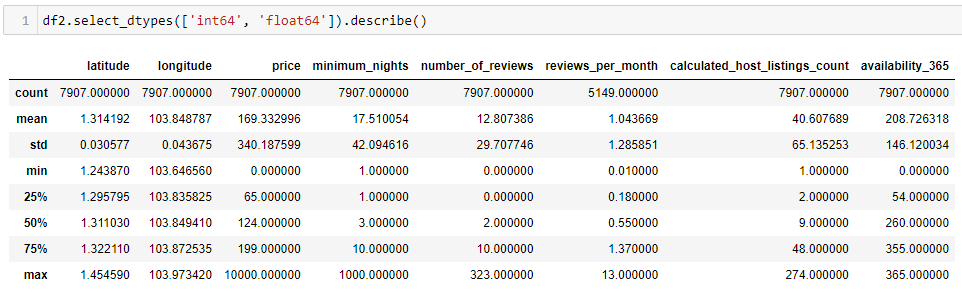
### Dropping Redundant Columns

The columns “id”, “name”, “host\_id”, “host\_name”, and “last\_review” are dropped as they are unnecessary for analysis. The code used is shown below.

# Drop columns as they are unnecessary for analysis  
df2 = df2.drop(['id', 'name', 'host\_id', 'host\_name', last\_review'], axis = 1)

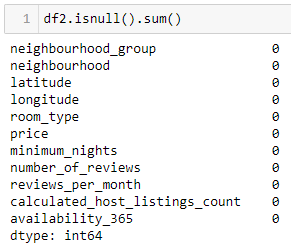
### Removing Null Values

After removing the columns, “reviews\_per\_month” is the only column with null values. The statistical breakdown of the column is found, and the null values are replaced with the median.



# Replace nulls in reviews\_per\_month with median  
df2['reviews\_per\_month'] = df2['reviews\_per\_month'].fillna(df2['reviews\_per\_month'].median()).astype(float)

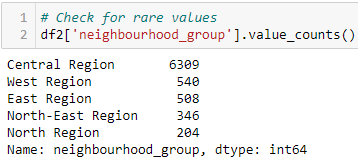
After replacing the null values, the dataframe is checked once more for null values.



### One-hot Encoding (OHE) and Mapping

There are three categorical columns: “neighbourhood\_group”, “neighbourhood”, and “room\_type”. Their string values will be replaced by integers for the machine-learning model.

#### neighbourhood\_group



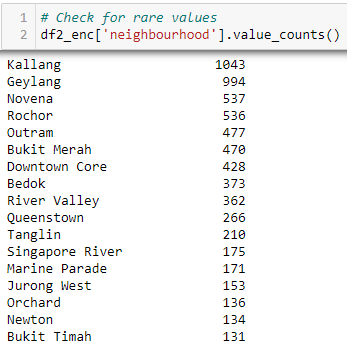
Since there are only five unique values, OHE will be used. The code is shown below.

# Create encoder  
neighbourhood\_enc = ohe()  
  
# Create variable to store fitted and transformed encoder  
temp = neighbourhood\_enc.fit\_transform(df2[['neighbourhood\_group']]).toarray()  
  
# Get labels  
labels = neighbourhood\_enc.categories\_  
  
# Create dataframe for 'neighbourhood\_group'  
features = pd.DataFrame(temp, columns = labels)  
  
# Concatenate dataframe with df\_copy1  
df2\_enc = pd.concat([df2, features], axis = 1)  
  
# Drop 'education' column  
df2\_enc = df2\_enc.drop('neighbourhood\_group', axis = 1)  
  
df2\_enc.head()

A picture containing table

Description automatically generated

#### neighbourhood



Since there are values that appear less than 100 times, they will be considered rare. The code used to replace the values with “rare” and the subsequent mapping will be shown together.

# Replace values with <100 count  
# Pasir Ris, Serangoon, Sengkang, Woodlands, Bukit Batok, Tampines, Museum, Choa Chu Kang, Ang Mo Kio, Bishan, Yishun, Punggol, Sembawang, Bukit Panjang, Central Water Catchment, Southern Islands, Sungei Kadut, Western Water Catchment, Mandai, Tuas, Marina South, and Lim Chu Kang with ‘rare’

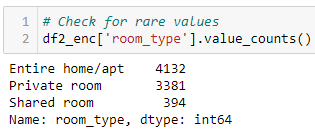
df2\_enc['neighbourhood'] = df2\_enc['neighbourhood'].replace(['Pasir Ris', 'Serangoon', 'Sengkang', 'Woodlands', 'Bukit Batok', 'Tampines', 'Museum', 'Choa Chu Kang', 'Ang Mo Kio', 'Bishan', 'Yishun', 'Punggol', 'Sembawang', 'Bukit Panjang', 'Central Water Catchment', 'Southern Islands', 'Sungei Kadut', 'Western Water Catchment', 'Mandai', 'Tuas', 'Marina South', 'Lim Chu Kang'], 'rare')

# Map values to integers  
df2\_enc['neighbourhood'] = df2\_enc['neighbourhood'].map({'Kallang': 0, 'Geylang': 1, 'Novena': 2, 'Rochor': 3, 'Outram': 4, 'Bukit Merah': 5, 'Downtown Core': 6, 'Bedok': 7, 'River Valley': 8, 'Queenstown': 9, 'Tanglin': 10, 'Singapore River': 11, 'Marine Parade': 12, 'Jurong West': 13, 'Orchard': 14, 'Newton': 15, 'Bukit Timah': 16, 'Jurong East': 17, 'Hougang': 18, 'Clementi': 19, 'Toa Payoh': 20, 'rare': 21}).astype(int)

Graphical user interface, table

Description automatically generated

#### room\_type



Since there are only five unique values, OHE will be used. The code is shown below.

# Create encoder  
room\_enc = ohe()  
  
# Create variable to store fitted and transformed encoder  
temp = room\_enc.fit\_transform(df2\_enc[['room\_type']]).toarray()  
  
# Get labels  
labels = room\_enc.categories\_  
  
# Create dataframe for 'room\_type'  
features = pd.DataFrame(temp, columns = labels)  
  
# Concatenate dataframe with df\_copy1  
df2\_enc = pd.concat([df2\_enc, features], axis = 1)  
  
# Drop 'room\_type' column  
df2\_enc = df2\_enc.drop('room\_type', axis = 1)  
  
df2\_enc.head()

Table

Description automatically generated

### Train-test Split

# Split both Inputs (X) and Output (y) into training set (70%) and testing set (30%)  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(df2\_enc.drop(['price'], axis = 1), df2\_enc['price'], test\_size = 0.3, random\_state = 10)

### Check for Outliers

X\_train will be checked for any outliers. The previous method used to plot will be used here.

Graphical user interface

Description automatically generated

Chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated

It shows that “minimum\_nights”, “number\_of\_reviews”, “reviews\_per\_month”, and “calculated\_host\_listings\_count” have outliers while “availability\_365” does not. As such, “availability\_365” will not be changed. Like before, a YeoJohnsonTransformer will be used to remove the outlier. The code used to trim the outliers is shown below.

# Create list to store column names  
col = ['minimum\_nights', 'number\_of\_reviews', 'reviews\_per\_month', 'calculated\_host\_listings\_count']  
  
# initialize the transformer with a subset of variables to transform  
yjt = YeoJohnsonTransformer(variables = col)  
  
# fit transformer to the dataframe  
yjt.fit(X\_train)  
  
# transform indicated variables  
X\_train\_tf = yjt.transform(X\_train)

After transforming, the images below show what the new column values look like.

Chart, histogram

Description automatically generated

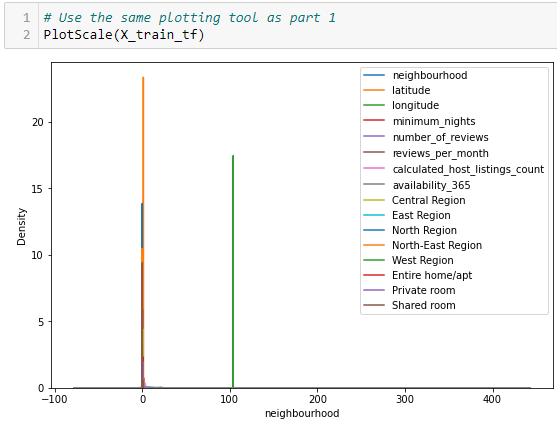
Chart, histogram

Description automatically generated

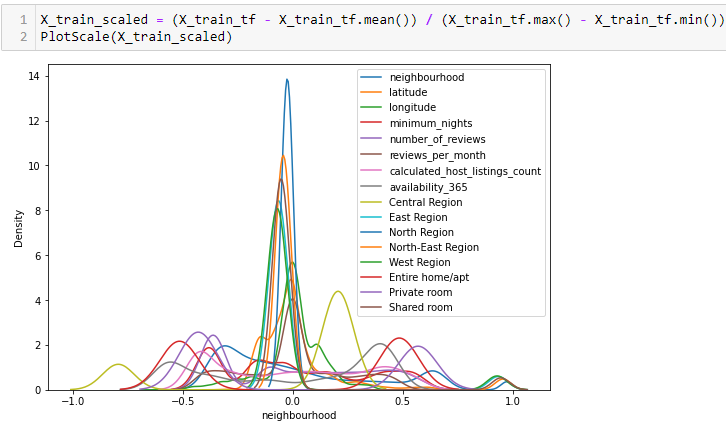
### Standardising data

Like before, a min-max scaler will be used to standardise the scale of all the columns. The method created previously will be reused, and the procedures will be the same – plot the values before standardising, standardise the values, and plot values after standardising.

The image below shows the data before standardising.

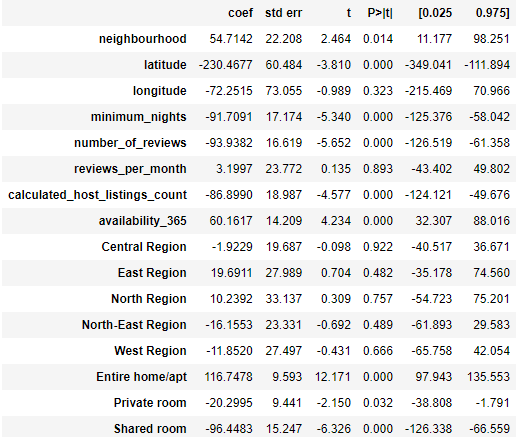


The image below shows the code used to standardise the data and the data after standardising.



## Correlation Analysis

Like previously, an OLS model will be used to identify which columns have a high and low significance. The code used to create the model is similar. The resulting table is shown below.

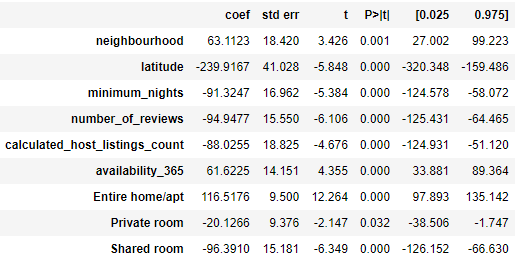


From the image above, it shows that the columns “longitude”, “reviews\_per\_month”, “Central Region”, “East Region”, “North Region”, “North-East Region”, and “West Region” have P>|t| values more than 0.05. This implies that they have a low significance level and are removable without affecting the final predictive model too much.

The code used to remove those columns, and re-transform the data is shown below.

# Remove columns with P>|t| values more than 0.05  
# longitude, reviews\_per\_month, Central Region, East Region, North Region, North-East Region, West Region  
df2\_final = df2\_enc.drop(['longitude', 'reviews\_per\_month', "Central Region", "East Region", "North Region", "North-East Region", 'West Region'], axis = 1)  
  
# Split both Inputs (X) and Output (y) into training set (70%) and testing set (30%)  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(df2\_final.drop(['price'], axis = 1), df2\_final['price'], test\_size = 0.3, random\_state = 10)  
  
# Create list to store column names  
col = ['minimum\_nights', 'number\_of\_reviews', calculated\_host\_listings\_count']  
  
# initialize the transformer with a subset of variables to transform  
yjt = YeoJohnsonTransformer(variables = col)  
  
# fit transformer to the dataframe  
yjt.fit(X\_train)  
  
# transform indicated variables  
X\_train\_tf = yjt.transform(X\_train)  
  
X\_train\_scaled = (X\_train\_tf - X\_train\_tf.mean()) / (X\_train\_tf.max() - X\_train\_tf.min())

The OLS model is created again using the same code and the table is shown below.



Finally, the X\_train, X\_test, y\_train, and y\_test sets are exported. The code used is shown below.

airbnb\_final\_Xtrain = X\_train\_scaled  
airbnb\_final\_Xtest = X\_test  
airbnb\_final\_ytrain = y\_train  
airbnb\_final\_ytest = y\_test  
  
airbnb\_final\_Xtrain.to\_csv('airbnb\_final\_Xtrain.csv', index = False, sep = ',', encoding = 'utf-8')  
airbnb\_final\_Xtest.to\_csv('airbnb\_final\_Xtest.csv', index = False, sep = ',', encoding = 'utf-8')  
airbnb\_final\_ytrain.to\_csv('airbnb\_final\_ytrain.csv', index = False, sep = ',', encoding = 'utf-8')  
airbnb\_final\_ytest.to\_csv('airbnb\_final\_ytest.csv', index = False, sep = ',', encoding = 'utf-8')

# Summary and Further Improvements

## Summary

### HR Analytics

Since the target column, “is\_promoted”, has boolean values, a logistic regressor is needed. The data also must be sampled to ensure an equal proportion of 1s and 0s.

Through the OLS model, the number of training, length of service, education, gender, and recruitment channel were insignificant in predicting promotion. This is astonishing as one would expect staff with a long service length, with many trainings, or with a bachelor’s would be more likely to be promoted. It is also reassuring that gender does not affect promotion.

The original train set had 17 columns. Through OLS, this number was reduced by 10 to 7.

### Airbnb

Since the target column, “price”, has continuous values, a linear regressor is needed.

Through the OLS model, the longitude, reviews per month, and neighbourhood group were insignificant in predicting price. It is odd how latitude is more significant than longitude. It is also unusual how neighbourhood group is insignificant since some of Singapore’s places of interest are located near the central business district (CBD). As such, it is plausible to infer most people would stay in neighbourhood groups near the CBD like Central or East.

The original train set had 16 columns. Through OLS, this number was reduced by 7 to 9.

## Further Improvements

1. After standardising the data for both HR Analytics and Airbnb, there are some columns with values of more than 1. This implies that not all outliers were removed, or the chosen method was ineffective. As such, using a different method to remove outliers (i.e., interquartile range capping, standard deviation capping, and mean absolute difference (MAD) capping) may produce better results.
2. For HR Analytics, the final OLS model shows that the “age” column has a P>|t| value of more than 0.05. Hence, it may be possible to remove the column. However, without a logistic regression model to test, it is difficult to conclude.