National College of Ireland

Configuration Manual

MSc
Data Analytics

Dawn Walsh
Student ID: x19190352

School of Computing
National College of Ireland

Supervisor: Pramod Pathak & Paul Stynes

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Dawn Walsh
Student ID:	x19190352
Programme:	MScData Analytics
Year:	2021
Module:	Research Project
Supervisor:	Pramod Pathak & Paul Stynes
Submission Due Date:	16/08/2021
Project Title:	Configuration Manual
Word Count:	
Page Count:	11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Dawn Walsh
Date:	14th August 2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Dawn Walsh
x19190352

1 Introduction

This configuration manual lists all the hardware and software requirements to reproduce the experiments in this research. The document lays out the steps taken from acquiring the data to implementing the models.

2 Hardware & Software Requirements

The hardware specifications utilised in this research is shown in Table 1. The programming language and libraries used alongside their respective versions are laid out in Table 2.

RAM	16GB
Processor	Dual-Core Intel Core i5 @ 1.6GHz
OS	macOS 11.5.1 (20G80)

Table 1: Hardware Specifications

Library	Version
Python	3.8.8
Jupyter Lab	3.0.11
pandas	1.2.3
numpy	1.19.2
scipy	1.6.1
scikit learn	0.24.1
matplotlib	3.3.4
seaborn	0.11.1

Table 2: Software Specifications

3 Dataset

Instructions on loading and working with the datasets used in this research are set out in the sections below.

3.1 Folder Structure

The files need to be inside the Project folder as in Fig. 1 for the paths to work correctly, you will also have to specify your own path within the code. When running the code in a Windows or Linux machine you will have to change the paths. The files shown in Fig. 1 will be explained in this manual.

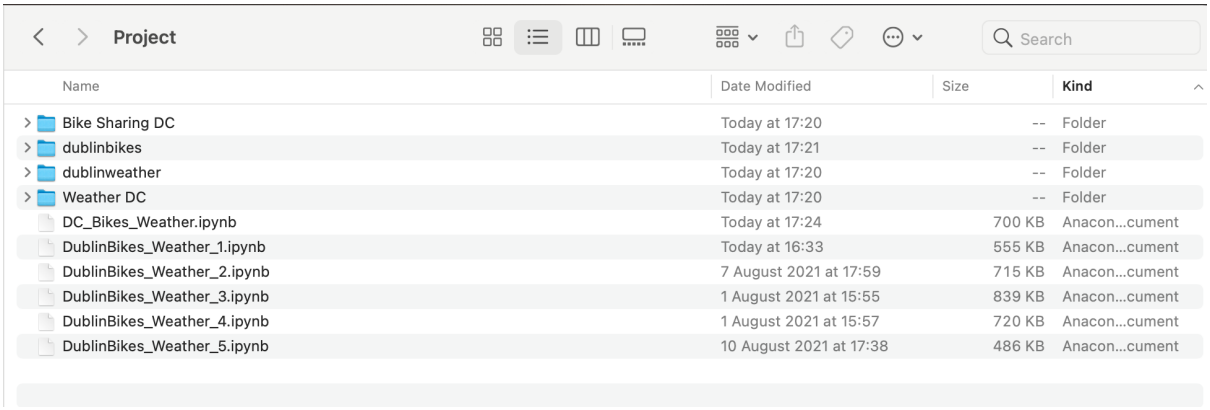

Figure 1: Structure of the Project Folder

3.2 Dataset Creation

The datasets were downloaded from the links shown in Table 3 below. There are multiple files for each dataset all in .csv format and each dataset has its own folder.

Dataset	URL
DC Bikes	https://s3.amazonaws.com/capitalbikesharedata/index.html
DC Weather	https://www.visualcrossing.com/weather-data
Dublin Bikes	https://data.smartdublin.ie/dataset/dublinbikes-api
Dublin Weather	https://www.met.ie/climate/available-data/historical-data

Table 3: Dataset Locations

Figures 2 and 3 shows the code used for loading the datasets. This allows multiple .csv files to be loaded and appended to one another and then concatenated to create one pandas dataframe.

4 Experiments

This section contains a brief outline of the various experiments run in the course of the research. Figure 4 shows the necessary libraries and imports for all of these experiments.

4.1 Reproducing DC Bikes/Weather Clustering

Figures 5, 6, 7 shows how the bikes and weather datasets are amended before being joined together. The joined data needed days of the week Fig. 8 months and seasons Fig. 9 extracted.

```
path = '/Users/dawn/Desktop/MSc Data Analytics/Term 3/Research Project'
all_files = glob.glob(path + "/Project/Bike Sharing DC/*.csv")

li = []

for filename in all_files:
    df = pd.read_csv(filename, sep = ',', index_col=None, header=0)
    li.append(df)

df_DC_Bikes = pd.concat(li, axis=0, ignore_index=True)
df_DC_Bikes.head()
```

Figure 2: Bikes Dataset

```
path1 = '/Users/dawn/Desktop/MSc Data Analytics/Term 3/Research Project'
all_files1 = glob.glob(path1 + "/Project/Weather DC/*.csv")

li1 = []

for filename in all_files1:
    df = pd.read_csv(filename, sep = ',', index_col=None, header=0)
    li1.append(df)

df_DC_Weather = pd.concat(li1, axis=0, ignore_index=True)
df_DC_Weather.head()
```

Figure 3: Weather Dataset

The data was standardised and the principle components extracted shown in Figures 10 and 11. The KMeans models in all the experiments were all optimised using Elbow and Silhouette methods. Figures 12,13 show them.

4.2 Clusters: Dublin Bikes & Weather - Daily Trips

Before fitting the models the two Dublin datasets need some features to be engineered such as the number of daily trips Figure 14.

Most of the rest of the code is the same as in Section 4.1, however since the trips data has to be extrapolated the two datasets remain as Pandas data-frames and are joined rather than mapped together as in the previous experiment shown in Figure 15.

4.3 Clusters: Dublin Bikes & Weather - Hourly Trips

As in Section 4.2 the trips have to be extrapolated but it is on a per hour basis. So the date and hour are extracted and the data is grouped by day and hour to give the hourly number of trips and shown in Figure 16. Figure 17 shows the hourly weather data joined to the hourly trips data.

4.4 Clusters: Dublin Bikes & Weather - Station Daily Trips

As in Section 4.2 the trips per day have to be extrapolated, however it is done per station as an additional detail. So the date is extracted and the data is grouped by day and

```
import pandas as pd
import glob
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
import scipy.stats as stats
import math
import numpy as np
import random
from matplotlib import style
import matplotlib.cm as cm
import seaborn as sns
import sklearn
import datetime as dt
```

```
# Classification
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA
```

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
```

Figure 4: Necessary Imports for Clustering

Dropping 'Conditions' column as it is not numeric and therefore will not work with clustering techniques.

```
df_DC_Weather = df_DC_Weather.drop(['Name', 'Wind Chill', 'Heat Index', 'Wind Gust', 'Conditions'], axis = 1)
df_DC_Weather
```

Figure 5: Weather Data preparation

Ride Counts by day ¶

```
ride_count = df_DC_Bikes['Start date'].value_counts()
ride_count
```

Figure 6: Bikes Data preparation

Appending the ride count to the data frame

```
df_DC_Weather['ride count'] = df_DC_Weather['Date time'].map(ride_count)
```

```
df_DC_Weather.set_index = np.arange(1, len(df_DC_Weather) + 1)
df_DC_Weather
```

Figure 7: Joining the Datasets

```
df_Cluster_Weather['Day'] = df_Cluster_Weather['Date time'].dt.day_name()
df_Cluster_Weather['Day'] = ['Weekend' if d == 'Sunday' or d == 'Saturday' else 'Weekday' for d in df_Cluster_Weather['Day']]
```

Figure 8: Extracting Day of Week, Weekday/Weekend

```
df_Cluster_Weather['Month'] = df_Cluster_Weather['Date time'].dt.month_name()

seasons = {"January": "Winter",
           "February": "Spring",
           "March": "Spring",
           "April": "Spring",
          "May": "Summer",
          "June": "Summer",
          "July": "Summer",
          "August": "Autumn",
          "September": "Autumn",
          "October": "Autumn",
          "November": "Winter",
          "December": "Winter"}

df_Cluster_Weather['Season'] = df_Cluster_Weather['Month'].map(seasons)
```

Figure 9: Extracting Month and Season

```
scaler = StandardScaler()
Standard_Bikes_scaled = scaler.fit_transform(Standard_Bikes)
Standard_Bikes_scaled.shape
```

```
Standard_Bikes_scaled = pd.DataFrame(Standard_Bikes_scaled)
Standard_Bikes_scaled.columns = ['maxtp', 'mintp', 'gmin', 'rain', 'cbl', 'soil', 'trips']
Standard_Bikes_scaled.head()
```

Figure 10: Standardising the data

```
pca = PCA(2)
Standard_Bikes1 = Standard_Bikes.drop(['Cluster ID'], axis = 1)
Standard_Bikes1_scaled = Standard_Bikes_scaled.drop(['Cluster ID'], axis = 1)
Standard_Bikes1 = pca.fit_transform(Standard_Bikes1)
Standard_Bikes1_scaled = pca.fit_transform(Standard_Bikes1_scaled)
```

```
Standard_Bikes1 = pd.DataFrame(Standard_Bikes1)
Standard_Bikes1_scaled = pd.DataFrame(Standard_Bikes1_scaled)
Standard_Bikes1_scaled.columns = ['PC 1', 'PC 2']
```

Figure 11: Principle Components Analysis

```
# Using the elbow method to find the optimal number of clusters
ssd = []
for i in range (2, 9):
    kmeans = KMeans(n_clusters = i, max_iter = 50)
    kmeans.fit(X_scaled)
    ssd.append(kmeans.inertia_)

plt.plot(range(2,9), ssd)
plt.title('The Elbow Method')

plt.show()
```

Figure 12: Elbow Method

```
for i in range (2, 9):
    kmeans = KMeans(n_clusters = i, max_iter = 50)
    kmeans.fit(X_scaled)

    cluster_labels = kmeans.labels_

    silhouette_avg = silhouette_score(X_scaled, cluster_labels)
    print("For n_clusters = {}, the silhouette score is {}".format(i, silhouette_avg))
```

Figure 13: Silhouette

```
df_Bikes_Out['BIKES OUT'] = 1600 - df_Bikes_Out['AVAILABLE BIKES']
df_Bikes_Out['DIFF BIKES OUT'] = df_Bikes_Out['BIKES OUT'].diff()
# getting rid of NaN and negative values
df_Bikes_Out['DIFF BIKES OUT'] = df_Bikes_Out['DIFF BIKES OUT'].fillna(0)
df_Bikes_Out['DIFF BIKES OUT'] = df_Bikes_Out['DIFF BIKES OUT'].mask(df_Bikes_Out['DIFF BIKES OUT'].lt(0),0)
df_Bikes_Out.head()
```

```
df_Bikes_Daily = df_Bikes_Out.drop(['TIME', 'BIKE STANDS', 'AVAILABLE BIKE STANDS', 'AVAILABLE BIKES', 'BIKES OUT'], axis = 1)
df_Bikes_Daily.head()
```

```
df_Bikes_Daily = df_Bikes_Daily.groupby('DATE').sum()
df_Bikes_Daily.head(10)
```

Figure 14: Extracting Trip information from Dublin Bikes

```
df_Bikes_Weather = pd.merge(df_Dub_Weather_daily, df_Bikes_Daily, on = 'date', how='left')
df_Bikes_Weather = df_Bikes_Weather.dropna()
df_Bikes_Weather.reset_index(inplace = True)
df_Bikes_Weather = df_Bikes_Weather.drop(['index'], axis = 1)
df_Bikes_Weather.head()
```

Figure 15: Joining the Data for Dublin

```
df_Bikes_Out['DATE'] = df_Bikes_Out['TIME'].dt.date
df_Bikes_Out['HOUR'] = df_Bikes_Out['TIME'].dt.hour
df_Bikes_Out.head()
```

There are approximately 1600 bikes in the Dublin Bikes scheme from mid-2018 to current

```
df_Bikes_Out['BIKES OUT'] = 1600 - df_Bikes_Out['AVAILABLE BIKES']
df_Bikes_Out['DIFF BIKES OUT'] = df_Bikes_Out['BIKES OUT'].diff()
df_Bikes_Out['DIFF BIKES OUT'] = df_Bikes_Out['DIFF BIKES OUT'].fillna(0)
df_Bikes_Out['DIFF BIKES OUT'] = df_Bikes_Out['DIFF BIKES OUT'].mask(df_Bikes_Out['DIFF BIKES OUT'].lt(0),0)
df_Bikes_Out.head()
```

```
df_Bikes_Daily = df_Bikes_Out.drop(['TIME', 'BIKE STANDS', 'AVAILABLE BIKE STANDS', 'AVAILABLE BIKES', 'BIKES OUT'], axis = 1)
df_Bikes_Daily = df_Bikes_Daily.groupby(['DATE', 'HOUR']).sum()
df_Bikes_Daily = df_Bikes_Daily.reset_index()
df_Bikes_Daily.head()
```

Figure 16: Dublin Bikes Hourly Trips

```
df_Bikes_Weather_hour = pd.merge(df_Dub_Weather_hourly, df_Bikes_Daily, on = ['date', 'hour'], how='left')
df_Bikes_Weather_hour['hour'] = df_Bikes_Weather_hour['hour'].dt.hour
df_Bikes_Weather_hour = df_Bikes_Weather_hour.dropna()
df_Bikes_Weather_hour.reset_index(inplace = True)
df_Bikes_Weather_hour = df_Bikes_Weather_hour.drop(['index'], axis = 1)
df_Bikes_Weather_hour.head()
```

Figure 17: Dublin Bikes & Weather Hourly Trips Joined

station to give the daily number of trips per station and shown in Figure 18. Figure 19 shows the daily weather data joined to the daily trips per station data.

```
df_Bikes_Out['DIFF BIKES OUT'] = df_Bikes_Out['AVAILABLE BIKE STANDS'].diff()
df_Bikes_Out['DIFF BIKES OUT'] = df_Bikes_Out['DIFF BIKES OUT'].fillna(0)
df_Bikes_Out['DIFF BIKES OUT'] = df_Bikes_Out['DIFF BIKES OUT'].mask(df_Bikes_Out['DIFF BIKES OUT'].lt(0),0)
df_Bikes_Out = df_Bikes_Out.reset_index()

df_Bikes_Out['DATE'] = df_Bikes_Out['TIME'].dt.date
df_Bikes_Daily = df_Bikes_Out.drop(['TIME', 'BIKE STANDS', 'AVAILABLE BIKE STANDS', 'AVAILABLE BIKES'], axis = 1)
df_Bikes_Daily = df_Bikes_Daily.groupby(['DATE', 'STATION ID']).sum()
df_Bikes_Out.head()
```

Figure 18: Dublin Bikes Daily Trips per Station

4.5 Clusters: Dublin Bikes & Weather - Quarter Hourly Trips

As in Sections 4.3 the trips per quarter hour have to be extrapolated. So the date, hour and quarter are extracted and the data is grouped by day and hour and quarter to give the quarter hourly number of trips shown in Figure 20. Figure 21 shows the quarter hourly weather data joined to the hourly trips per station data.

4.6 Random Forest Classifier: Dublin Bikes & Weather

For the Random Forest Classifier (RFC) the bikes data is configured differently. Similar to previous sections the day, hour and month are extracted Figures 22. In order to allow the station id to be retained we look at the overall usage statistics for the station Figure 23 by binning the day into time-spans Figure 24. The table is pivoted then to give overall usage statistics for the station on the given time spans and whether or not it is a Weekday, Saturday or Sunday Figure 25.

Figure 26 shows creating the dataset for the RFC model firstly using the occupancy rate to classify the state of the station, with 0 (Needs to be restocked), 1 (Acceptable)

```
df_Bikes_Weather = pd.merge(df_Dub_Weather_daily, df_Bikes_Daily, on = 'date', how='left')
df_Bikes_Weather = df_Bikes_Weather.dropna()
df_Bikes_Weather.reset_index(inplace = True)
df_Bikes_Weather = df_Bikes_Weather.drop(['index'], axis = 1)
df_Bikes_Weather.head()
```

Figure 19: Dublin Bikes & Weather Daily Trips per Station Joined

```
df_Bikes_Out['DATE'] = df_Bikes_Out['TIME'].dt.date
df_Bikes_Out['HOUR'] = df_Bikes_Out['TIME'].dt.hour
df_Bikes_Out['MINUTE'] = df_Bikes_Out['TIME'].dt.minute//15
df_Bikes_Out.head()
```

```
df_Bikes_Out['BIKES OUT'] = 1600 - df_Bikes_Out['AVAILABLE BIKES']
df_Bikes_Out['DIFF BIKES OUT'] = df_Bikes_Out['BIKES OUT'].diff()
df_Bikes_Out['DIFF BIKES OUT'] = df_Bikes_Out['DIFF BIKES OUT'].fillna(0)
df_Bikes_Out['DIFF BIKES OUT'] = df_Bikes_Out['DIFF BIKES OUT'].mask(df_Bikes_Out['DIFF BIKES OUT'].lt(0),0)
df_Bikes_Daily = df_Bikes_Out.drop(['TIME', 'BIKE STANDS', 'AVAILABLE BIKE STANDS', 'AVAILABLE BIKES', 'BIKES OUT'], axis = 1)
df_Bikes_Daily = df_Bikes_Daily.groupby(['DATE', 'HOUR', 'MINUTE']).sum()
df_Bikes_Daily = df_Bikes_Daily.reset_index()
df_Bikes_Out.head()
```

Figure 20: Dublin Bikes Quarter Hourly Trips

Duplicated to allow join on quarter hour

```
df_Dub_Weather_hourly = pd.concat([df_Dub_Weather_hourly]*4, ignore_index=True)
```

```
df_Dub_Weather_hourly = df_Dub_Weather_hourly.sort_values(['date', 'hour'], ascending=[True, True])
df_Dub_Weather_hourly.head()
```

```
df_Dub_Weather_hourly.reset_index(inplace = True)
df_Dub_Weather_hourly = df_Dub_Weather_hourly.drop(['index'], axis = 1)
df_Dub_Weather_hourly.head()
```

```
df_Bikes_Daily.rename(columns={'DATE':'date', 'HOUR': 'hour', 'MINUTE': 'quarter', 'DIFF BIKES OUT':'trips'}, inplace=True)
df_Bikes_Daily.head()
```

Figure 21: Dublin Bikes & Weather Quarter Hourly Trips Joined

or 2(Needs to be emptied) and then splitting the data into train and test sets Figure 27. Then the model is fitted, the confusion matrix generated and the feature importance found in Figures 28,29 and30

```
df_DB['DATE TIME'] = df_DB['LAST UPDATED'].dt.round('H')
df_DB['DAY NUMBER'] = df_DB['DATE TIME'].dt.dayofweek
df_DB['DAY TYPE'] = np.where(df_DB['DAY NUMBER'] <= 4, 'Weekday', (np.where(df_DB['DAY NUMBER'] == 5, 'Saturday', 'Sunday')))
df_DB['HOUR'] = df_DB['LAST UPDATED'].dt.hour
df_DB['MONTH'] = df_DB['LAST UPDATED'].dt.month
df_DB.sample(5)
```

Figure 22: Extracting Features for RFC: day, hour, month

```
df_DB['OCCUPANCY RATE'] = df_DB['AVAILABLE BIKES'] / df_DB['BIKE STANDS']
df_DB['FULL'] = np.where(df_DB['OCCUPANCY RATE'] == 1, 1, 0)
df_DB['EMPTY'] = np.where(df_DB['OCCUPANCY RATE'] == 0, 1, 0)
df_DB.sample(10)
```

Figure 23: Extracting Features for RFC: occupancy rate of station

```
def time_group(x):
    if x.time() < dt.time(6):
        return "Overnight "
    elif x.time() < dt.time(11):
        return "6am-10am "
    elif x.time() < dt.time(16):
        return "11am-3pm "
    elif x.time() < dt.time(20):
        return "4pm-7pm "
    elif x.time() <= dt.time(23):
        return "8pm-11pm "
    else:
        return "Overnight "

df_DB['TIME GROUP'] = df_DB['DATE TIME'].apply(time_group)
df_DB['CLUSTER GROUP'] = df_DB['TIME GROUP'] + df_DB['DAY TYPE']
df_DB.sample(5)
```

Figure 24: Extracting Features for RFC: binning times

```
df_Time_Cluster = df_merge[['STATION ID', 'NAME', 'LATITUDE', 'LONGITUDE', 'DAY TYPE', 'TIME GROUP', 'OCCUPANCY RATE', 'CLUSTER GROUP']]
df_Time_Cluster = df_Time_Cluster.groupby(['STATION ID', 'NAME', 'LATITUDE', 'LONGITUDE', 'CLUSTER GROUP'], as_index = False)['OCCUPANCY RATE'].mean()
df_Time_Cluster = df_Time_Cluster.set_index('STATION ID')
```

```
df_Time_Cluster = df_Time_Cluster.pivot_table(index = ['NAME', 'STATION ID', 'LATITUDE', 'LONGITUDE'], columns = ['CLUSTER GROUP'], values = 'OCCUPANCY RATE')
df_Time_Cluster = df_Time_Cluster.reset_index()
df_Time_Cluster = df_Time_Cluster.set_index('NAME')
df_Time_Cluster = df_Time_Cluster.dropna()
df_Time_Cluster.sample(5)
```

CLUSTER GROUP	STATION ID	LATITUDE	LONGITUDE	11am-3pm Saturday	11am-3pm Sunday	11am-3pm Weekday	4pm-7pm Saturday	4pm-7pm Sunday	4pm-7pm Weekday	6am-10am Saturday	6am-10am Sunday	6am-10am Weekday	8pm-11pm Saturday	8pm-11pm Sunday	8pm-11pm Weekday	Overnight Saturday	Overnight Sunday	Overnight Weekday
NAME																		
SOUTH DOCK ROAD	91	53.341831	-6.231291	0.426138	0.460434	0.454444	0.488100	0.569974	0.501655	0.636543	0.554668	0.391778	0.540896	0.672532	0.648903	0.661672	0.535848	0.660899
WOLFE TONE STREET	77	53.348873	-6.267459	0.734400	0.825428	0.182415	0.775924	0.803139	0.172933	0.338364	0.693000	0.154887	0.695097	0.651187	0.187244	0.270262	0.690417	0.234797
EARLSFORT TERRACE	11	53.334019	-6.258371	0.036352	0.126928	0.554415	0.060549	0.132190	0.138088	0.052886	0.088988	0.506312	0.089826	0.130269	0.034841	0.030968	0.072399	0.039208
ROYAL HOSPITAL	95	53.343899	-6.297060	0.245920	0.131098	0.046437	0.205881	0.159310	0.352989	0.485652	0.236317	0.279501	0.225572	0.236232	0.701544	0.569363	0.269166	0.715598
NORTH CIRCULAR ROAD (O'CONNELL'S)	112	53.357841	-6.251557	0.143325	0.137118	0.123390	0.285592	0.212045	0.454438	0.482955	0.384285	0.226316	0.424983	0.431004	0.824598	0.927685	0.646045	0.848586

Figure 25: Pivoted Table

```
ml_data = df_merge_clusters[df_merge_clusters['REBALANCING'] < 1]
ml_data = ml_data[['STATION ID', 'OCCUPANCY RATE', 'dry', 'warm', 'DAY NUMBER', 'HOUR', 'MONTH']]

def bin_occupancy(x):
    if x < 0.1:
        # Needs to be restocked
        return 0
    elif x < 0.8:
        # Levels are acceptable
        return 0.1
    else:
        # Needs to be emptied
        return 0.2

ml_data['OCCUPANCY GROUP'] = ml_data['OCCUPANCY RATE'].apply(bin_occupancy)
ml_data['OCCUPANCY GROUP'] = ml_data['OCCUPANCY GROUP'] * 10
ml_data['OCCUPANCY GROUP'] = ml_data['OCCUPANCY GROUP'].astype(int)
ml_data.dropna(inplace = True)
mask = np.random.rand(len(ml_data)) < 0.8
train = ml_data[mask]
test = ml_data[~mask]
print(len(train))
print(len(test))
```

Figure 26: Classes for RFC

```
X_train = train.drop(['OCCUPANCY RATE', "OCCUPANCY GROUP"], axis = 1)
X_test = test.drop(['OCCUPANCY RATE', "OCCUPANCY GROUP"], axis = 1)
Y_train = train[["OCCUPANCY GROUP"]]
Y_test = test[["OCCUPANCY GROUP"]]
```

Figure 27: Train-Test Split for RFC

```
clf = RandomForestClassifier(n_estimators = 100)
clf.fit(X_train, Y_train)
Y_pred = clf.predict(X_test)
print("Accuracy: ", metrics.accuracy_score(Y_test, Y_pred))
print(sklearn.metrics.classification_report(Y_test, Y_pred))
```

Figure 28: Fitting RFC Model

```
labels = [0,1,2]
cm = metrics.confusion_matrix(Y_test, Y_pred, labels=labels)
ax= plt.subplot()
sns.heatmap(cm, annot=True, fmt='g', ax=ax, cmap= "rainbow");  #annot=True to annotate cells, ftm='g' to disable scientific notation

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
ax.xaxis.set_ticklabels(labels); ax.yaxis.set_ticklabels(labels)
```

Figure 29: RFC: Confusion Matrix

```
best_feature = pd.Series(clf.feature_importances_, index=['STATION ID', 'DRY', 'WARM', 'DAY NUMBER', 'HOUR', 'MONTH']).sort_values(ascending=False)
sns.barplot(x=best_feature, y=best_feature.index)
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title('Most Important Features')
plt.legend()
plt.show()
```

Figure 30: RFC: Feature Importance