Configuration Manual

MSc Research Project
Data Analytics

Aryan Rajput
Student ID: X20128088

School of Computing
National College of Ireland

Supervisor: Prof. Hicham Rifai

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Aryan Rajput
Student ID:	X20128088
Programme:	Data Analytics
Year:	2018
Module:	MSc Research Project
Supervisor:	Prof. Hicham Rifai
Submission Due Date:	31/01/2022
Project Title:	Configuration Manual
Word Count:	747
Page Count:	7

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:	Aryan Rajput
Date:	30th January 2022

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

Aryan Rajput
X20128088

1 Introduction

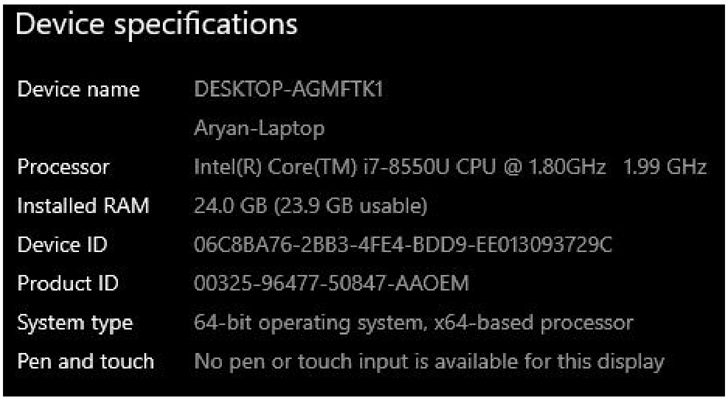
The main purpose of this document is to enlist the tasks that are needed to be executed while implementation of this project. Software and hardware prerequisites are provided in order to duplicate the project in the future. The coding processes are covered in this article, as well as the steps that needed to be followed in order to run the code.

2 System Configuration

The requirement for hardware and software that are used to carry out the research are enlisted in this section.

2.1 Hardware Configuration

The configuration of the hardware that was used in the research is displayed in Figure 1



Device specifications

Device name	DESKTOP-AGMFTK1
	Aryan-Laptop
Processor	Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99 GHz
Installed RAM	24.0 GB (23.9 GB usable)
Device ID	06C8BA76-2BB3-4FE4-BDD9-EE013093729C
Product ID	00325-96477-50847-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Figure 1: Hardware Configuration

Windows specifications are displayed in Figure 2

2.2 Software Configuration

The configuration of the software that were used in the research are explained in this section.

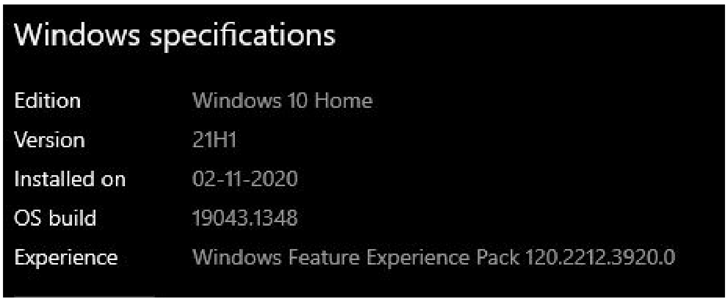


Figure 2: Hardware Configuration

2.2.1 Python

Python is used to carry out this study project. It has a significant number of classes which support Machine Learning techniques. It also comes with a number of libraries and packages that makes pre-processing and project implementation very simple. The latest version of python is downloaded and used to conduct this research. The python version used is 3.10.1

2.2.2 Anaconda

Anaconda is used to provide R and Python integrated development environment (IDE). It makes developers experience easy by integrating several platforms together. The anaconda version used for this project is 2.1.1

2.2.3 Jupyter Notebook

Jupyter Notebook has been used to develop the code and programs as the main IDE for this research. The version for Jupyter Notebook used is 6.3.0

2.2.4 Google Chrome

Google Chrome has been used as main browser to provide Jupyter Notebook runnable platform. The version used for Chrome is 96.0.4664.93

2.2.5 Overleaf

Overleaf has been used to develop the report for the project. Overleaf was also integrated with Jupyter Notebook to get the latest and updated output results.

3 Data Preparation

The dataset which has been used used for this research has been downloaded from open repository website kaggle [1]. The snapshot for the datset is depicted in Figure 3

The dataset consist of 5 columns and 1021064 rows, which consist of product details which is collected for over 6 years. After downloading the dataset, it has been uploaded to jupyter notebook and kept under same directory as the code, so that giving path and

[1] `https://www.kaggle.com/felixzhao/productdemandforecasting/`

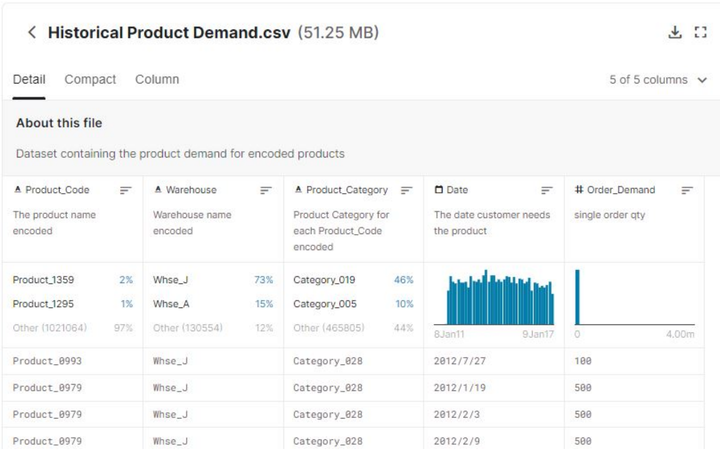


Figure 3: Dataset

importing other libraries was not needed. Refer to Figure 4 for importing dataset related details.

```
#Import the data and parse dates.
df  = pd.read_csv('Historical Product Demand.csv', parse_dates=['Date'])
```

Figure 4: Importing Dataset

4 Implementation

The python libraries that are used in the project needs to be updated to latest version. Specially keras and tensorflow libraries needed to be updated to latest version. The Tensorflow version used for research is 2.7.0 and Keras version is also 2.7.0.

4.1 Implementing SARIMA

All the libraries that were used for implementation of SARIMA are displayed in Figure 5

```
import pandas as pd
import numpy as np
import seaborn as sb

import matplotlib.pyplot as plt
%matplotlib inline

from scipy.stats import norm, skew
from scipy import stats
import statsmodels.api as sm

from statsmodels.tsa.statespace.sarimax import SARIMAX
```

Figure 5: SARIMA Libraries

After Importing and doing pre-processing, for SARIMA model the method which is used to calculate the best variable values has been displayed Figure 6

```
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(y,
                                            order=param,
                                            seasonal_order=param_seasonal,
                                            enforce_stationarity=False,
                                            enforce_invertibility=False)

            results = mod.fit()

            print('SARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal, results.aic))
        except:
            continue
```

Figure 6: Method to find Best-Fit SARIMA

```
SARIMA(0, 0, 0)x(0, 0, 0, 12)12 - AIC:1932.23655778549
SARIMA(0, 0, 0)x(0, 0, 1, 12)12 - AIC:1512.9275832124356
SARIMA(0, 0, 0)x(0, 1, 0, 12)12 - AIC:1338.8201294951011
SARIMA(0, 0, 0)x(0, 1, 1, 12)12 - AIC:3134.0602952352074
SARIMA(0, 0, 0)x(1, 0, 0, 12)12 - AIC:1366.5117513512635
```

Figure 7: Finding Best-Fit SARIMA

After finding the best fit SARIMA model by looking AIC value, main SARIMA model has been applied. A clear representation has been shown in Figure 8. All the AIC value have been compared and then it was decided to go with ARIMA(1,1,1)*(1,1,0,12).

```
#Fit the model with the best params.
#ARIMA(1, 1, 1)x(1, 1, 0, 12)12 - AIC:960.5164122018646

from statsmodels.tsa.statespace.sarimax import SARIMAX
mod = sm.tsa.statespace.SARIMAX(y,
                                order=(1, 1, 1),
                                seasonal_order=(1, 1, 0, 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results = mod.fit()
```

Figure 8: Implementing SARIMA

4.2 Implementing RNN

All the libraries that were used for implementation of RNN are displayed in Figure 9

```
# Importing Libraries
import pandas as pd;
import matplotlib.pyplot as plt
from numpy import array
from numpy import hstack
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import RNN, SimpleRNN
from keras.preprocessing.sequence import TimeseriesGenerator
from keras.layers import Dropout
# from keras.optimizers import Adam
from keras.layers.core import Activation
from keras.callbacks import LambdaCallback
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import LabelEncoder
```

Figure 9: RNN Libraries

After importing the dataset and doing all the necessary pre-processing tasks, data has been split into test, train, and hold data. To make a better understanding about it, please refer to Figure 10

Splitting data into train, test and hold-out data

```
number_of_test_data = 5000
number_of_holdout_data = 5000
number_of_training_data = len(dataset) - number_of_holdout_data - number_of_test_data
print ("total, train, test, holdout:", len(dataset), number_of_training_data, number_of_test_data, number_of_holdout_data)

total, train, test, holdout: 1048575 1038575 5000 5000
```

Figure 10: Splitting Test Train Data

After splitting the data, as it is needed in artificial neural network to provide a sequential three dimensional input, and thus we had to convert the normal input data to 3-dimensional input. The input code has been showed in Figure 11

After getting the data prepared, finally RNN model is defined as per Figure 11

Then created model is trained by fitting it to train data. Figure 12 describes the model fitting.

After model is fitted, order demand is predicted and results are checked by getting mean absolute error. Refer to Figure 14 and Figure 15 for prediction and output result.

Preparing 3-Dimensional Input for Sequential Model

```
in_seq1 = array(datatrain['Product_Code'])
in_seq2 = array(datatrain['Warehouse'])
in_seq3 = array(datatrain['Product_Category'])
in_seq4 = array(datatrain['Year'])
in_seq5 = array(datatrain['Month'])
in_seq6 = array(datatrain['Day'])
out_seq_train = array(datatrain['Order_Demand'])
```

```
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
in_seq3 = in_seq3.reshape((len(in_seq3), 1))
in_seq4 = in_seq4.reshape((len(in_seq4), 1))
in_seq5 = in_seq5.reshape((len(in_seq5), 1))
in_seq6 = in_seq6.reshape((len(in_seq6), 1))
out_seq_train = out_seq_train.reshape((len(out_seq_train), 1))
```

```
datatrain_feed = hstack((in_seq1, in_seq2, in_seq3, in_seq4, in_seq5, in_seq6, out_seq_train))
```

Figure 11: Preparing 3-D Input

Creating RNN model

```
: model = Sequential()

model.add(SimpleRNN(4, activation='linear', input_shape=(n_input, n_features), return_sequences = False))
model.add(Dense(1, activation='linear'))

adam = Adam(lr=0.0001)
model.compile(optimizer='adam', loss='mse')
```

Figure 12: Defining RNN Model

Training the model

```
]: score = model.fit_generator(generator_train, epochs=2000, verbose=2, validation_data=generator_test)
1/1 - 8s - loss: 5598492.0000 - val_loss: 4540064.0000 - 8s/epoch - 8s/step
Epoch 2/2000
1/1 - 7s - loss: 5345647.5000 - val_loss: 4337165.0000 - 7s/epoch - 7s/step
Epoch 3/2000
1/1 - 7s - loss: 5100613.0000 - val_loss: 4141780.0000 - 7s/epoch - 7s/step
Epoch 4/2000
1/1 - 7s - loss: 4863603.0000 - val_loss: 3954070.0000 - 7s/epoch - 7s/step
Epoch 5/2000
1/1 - 7s - loss: 4634810.0000 - val_loss: 3774198.5000 - 7s/epoch - 7s/step
```

Figure 13: Training RNN Model

```
df_result = pd.DataFrame({'Actual' : [], 'Prediction' : []})

for i in range(len(generator_test)):
    x, y = generator_test[i]
    x_input = array(x).reshape((1, n_input, n_features))
    yhat = model.predict(x_input, verbose=2)
    df_result = df_result.append({'Actual': scaler.inverse_transform(y)[0][0], 'Prediction': scaler.inverse_transform(yhat)
```

Figure 14: Predicting Order Demand

```
mean = df_result['Actual'].mean()
mae = (df_result['Actual'] - df_result['Prediction']).abs().mean()

print("mean: ", mean)
print("mae:", mae)
print("mae/mean ratio: ", 100*mae/mean,"%")
print("correctness: ", 100 - 100*mae/mean,"%")
```

```
mean:  5399364.85453817
mae: 3070834.150571028
mae/mean ratio:  56.87398857645247 %
correctness:  43.12601142354753 %
```

Figure 15: Interpreting Result