Configuration Manual

MSc Research Project
Data Analytics

Rashmikant T Shukla
Student ID: x18181236

School of Computing
National College of Ireland

Supervisor: Dr. Rashmi Gupta

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Rashmikant T Shukla
Student ID:	x18181236
Programme:	Data Analytics
Year:	2020
Module:	MSc Research Project
Supervisor:	Dr. Rashmi Gupta
Submission Due Date:	17/08/2020
Project Title:	Configuration Manual
Word Count:	1238
Page Count:	12

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:	
Date:	28th September 2020

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

Rashmikant T Shukla
x18181236

1 Introduction

This configuration manual document includes all the info about the technical environment, coding language, libraries used for the implementation of the project. This gives the description of Integrated Development Environment (IDE) used in the research. The configuration manual should be read in conjunction with the research report to recreate the same outputs.

2 Environment Specification

2.1 Hardware

Operating System: Windows 10 Home

Processor: Intel(R) Core(TM) i5-8265U CPU@ 1.60Hz, 1.80 GHz

Installed Memory (RAM): 8.00 GB

System Type: 64-bit Operating System, x64-based Processor

2.2 Software

Anaconda IDE-Jupyter Notebook: Anaconda IDE is an open-source distribution.This enable user to use Python, R by providing support for the jupyter, spyder and R-studio. This can be downloaded from their Website.[1] This research has used jupyter note book for the data conversion, exploratory data analysis, and visualization.

Microsoft Excel 2016: Excel is used to stored data just before model creation in CSV format.

Google Colaboratory : Google Colaboratory popularly known as Google Colab is free cloud based jupyter environment which allows individual users to train machine learning models on TPU which are much faster than the other systems. This research uses colab for final model creation, training and testing.

Coding Language, Environment and Libraries:

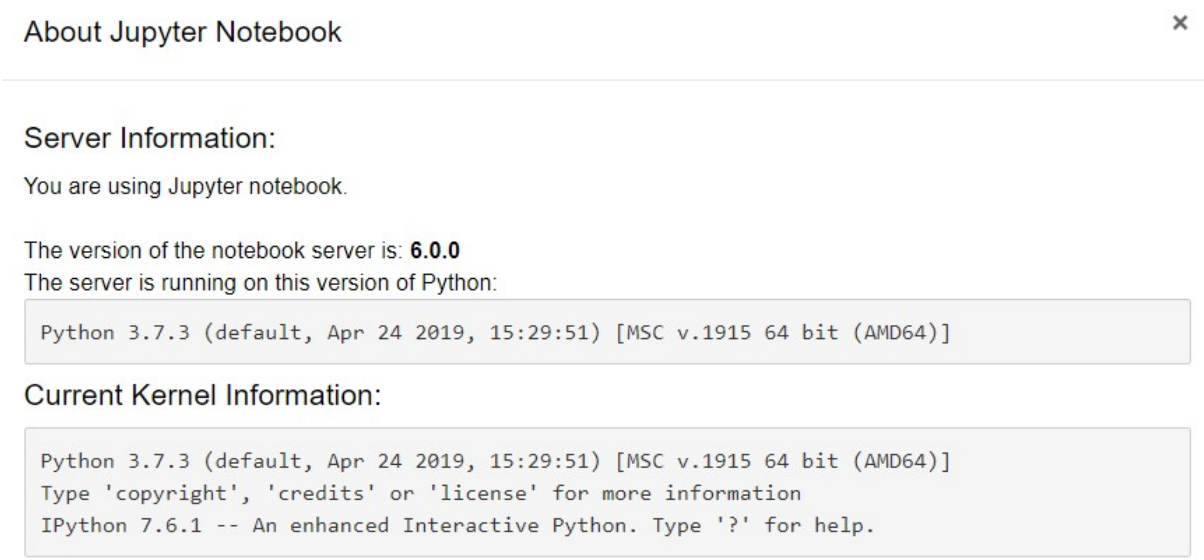


Figure 1: Python and Jupyter Version

- Coding language is python and jupyter notebook is used as platform (see Figure 1).
- Below mentioned libraries are used in this research:

  Numpy
  Scikit-learn
  Pandas
  Matplotlib
  Scipy
  Keras

3 Project Execution

Project execution starts with recognition of appropriate data. Then data preparation, feature extraction and finally model creation and its training and testing.

3.1 Data Selection

Data is taken from NASA AMES laboratory website. This dataset is publicly available it is in MATLAB format which directly cannot be used in python. Data downloaded from the website have .mat extension (see Figure 2).[2] This File includes Charging, Discharging and Impedance cycle.

Charging Cycle and Discharging Cycle: In this dataset Charging of the batteries are done under constant current of 1.5A until the voltage reached to 4.2V (single battery cell's maximum voltage) and then it is continued under this voltage

[1] `https://www.anaconda.com/products/individual`
[2] `https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/`

Battery Data Set
Publications using this data set

Description	Experiments on Li-Ion batteries. Charging and discharging at different temperatures. Records the impedance as the damage criterion. The data set was provided by the Prognostics CoE at NASA Ames.
Format	The set is in .mat format and has been zipped.
Datasets	+ Download Battery Data Set 1 (27686 downloads) + Download Battery Data Set 2 (14606 downloads) + Download Battery Data Set 3 (11723 downloads) + Download Battery Data Set 4 (8798 downloads) + Download Battery Data Set 5 (9489 downloads) + Download Battery Data Set 6 (9997 downloads)
Dataset Citation	B. Saha and K. Goebel (2007). "Battery Data Set", NASA Ames Prognostics Data Repository (http://ti.arc.nasa.gov/project/prognostic-data-repository), NASA Ames Research Center, Moffett Field, CA

Figure 2: Data Download Page

until current dropped to 20mA. Discharging is done at constant current of 2A until battery voltages of B005 reached 2.7V, B006 reached 2.5V, B007 reached 2.2V and B0018 reached 2.5 V (Goebel et al.; 2008). Each Charging and Discharging Cycle has parameters as shown in Table 1; [3]

Table 1: Charging and Discharging Cycle Parameters

Voltage_measured	Battery terminal voltage (Volts)
Current_measured	Battery output current (Amps)
Temperature_measured	Battery temperature (degree C)
Current_charge	Current measured at charger (Amps)
Voltage_charge	Voltage measured at charger (Volts)
Time	Time vector for the cycle (secs)

Impedance Cycle: Impedance measurements are taken by Electrochemical Impedance Spectroscopy (EIS) and selected frequency are from 0.1 Hz to 5kHz (Goebel et al.; 2008). Each Impedance Cycle has parameters as shown in Table 2;[4]

Table 2: Impedance Cycle Parameter

Sense_current	Current in sense branch (Amps)
Battery_current	Current in battery branch (Amps)
Current_ratio	Ratio of the above currents
Battery_impedance	Battery impedance (Ohms) computed from raw data
Rectified_impedance	Calibrated and smoothed battery impedance (Ohms)
Re	Estimated electrolyte resistance (Ohms)
Rct	Estimated charge transfer resistance (Ohms)

[3] `https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/`
[4] `https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/`

3.2 Data Pre-processing

Once data is selected it is necessary to under stand the structure of .mat file in order to convert it into appropriate format (json).

3.2.1 Overview of .mat File

For Each Elements in .mat Data File

Element [0] = charge/discharge/impedance

If element [0] = charge/discharge

- element [1] = ambient temperature
- element [2] = date/time
- element [3] = data

Data Fields:

- Voltage_measured
- Current_measured
- Temperature_measured
- Current_charge
- Voltage_charge
- Time

If element [0] = impedance

- element [1] = ambient temperature
- element [2] = date/time
- element [3] = data

Data Fields:

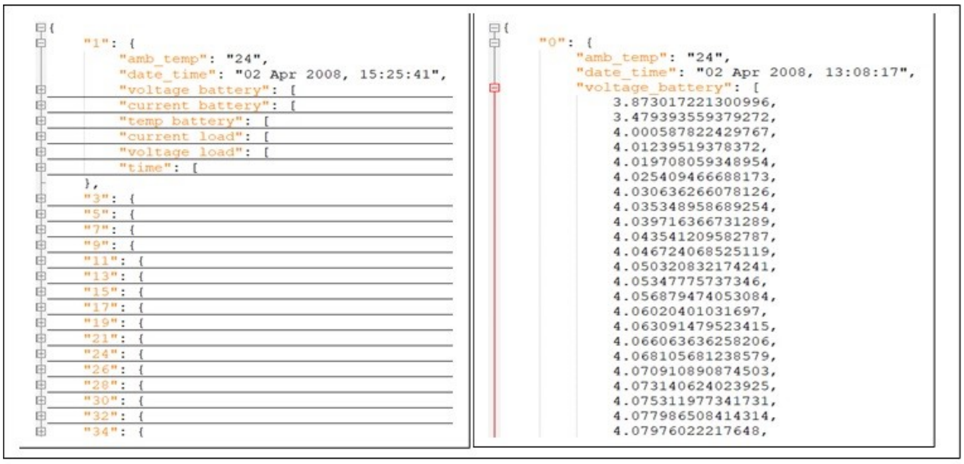
- Sense_current
- Battery_current
- Current_ratio
- Battery_impedance
- Rectified_Impedance
- Re
- Rct

3.2.2 Conversion of .mat File to json Format

This .mat file is converted into json format as shown below. .

```
#Importing All the required libraries
import pandas as pd
import numpy as np
from scipy.io import loadmat, whosmat
import numpy as np
import matplotlib.pyplot as plt
import datetime
import json
import os

#Creating Dictionaries
def build_dictionaries(bat):

    discharge, charge, impedance = fg, fg, fg

    for i, element in enumerate(bat):

        step = element[0][0]

        if step == 'discharge':
            discharge[str(i)] = fg
            discharge[str(i)]["amb_temp"] =
            str(element[1][0][0])
            year = int(element[2][0][0])
            month = int(element[2][0][1])
            day = int(element[2][0][2])
            hour = int(element[2][0][3])
            minute = int(element[2][0][4])
            second = int(element[2][0][5])
            millisecond = int((second % 1) 1000)
            date_time = datetime.datetime
            (year, month, day, hour, minute, second,
            millisecond)
            discharge[str(i)]["date_time"] =
            date_time.strftime
            ("%d %b %Y, %H:%M:%S")
```

After converting .mat to json, three files for each batteries are created for charging, discharging and impedance respectively. These file should be saved at appropriate folder as in next stage this folder path is required. Structure of created json file can be seen in Figure 3. This research uses only charging and discharging cycle.



```
{
    "1": {
        "amb_temp": "24",
        "date_time": "02 Apr 2008, 15:25:41",
        "voltage battery": [
        "current battery": [
        "temp battery": [
        "current load": [
        "voltage load": [
        "time": [
    },
    "3": {
    "5": {
    "7": {
    "9": {
    "11": {
    "13": {
    "15": {
    "17": {
    "19": {
    "21": {
    "24": {
    "26": {
    "28": {
    "30": {
    "32": {
    "34": {
```

```
{
    "0": {
        "amb_temp": "24",
        "date_time": "02 Apr 2008, 13:08:17",
        "voltage_battery": [
            3.873017221300996,
            3.479393559379272,
            4.000587822429767,
            4.01239519378372,
            4.019708059348954,
            4.025409466688173,
            4.030636266078126,
            4.035348958689254,
            4.039716366731289,
            4.043541209582787,
            4.046724068525119,
            4.050320832174241,
            4.05347775737346,
            4.056879474053084,
            4.06020401031697,
            4.063091479523415,
            4.066063636258206,
            4.068105681238579,
            4.070910890874503,
            4.073140624023925,
            4.075311977341731,
            4.077986508414314,
            4.07976022217648,
```


Figure 3: .mat to json File

3.3 Feature Extraction

As the file are converted into json file, now it can be opened in jupyter notebook. The next step is to extract features based on geometric feature of Li-ion batteries as discussed in research report. To execute this files are read in the jupyter notebook and based on various equation presented in research report coding for feature extraction is performed as shown below. Similarly operation is performed on each battery data.

```
#Feature Extraction of Battery B006

with open('D:nSEM 3nResearchnBatteryAgingARC FY08Q4
/B0006_charge.json') as f:
  charging_data = json.load(f)

charging_data.keys()
len(charging_data.keys())
170

Feature Extarction For Charging Cycle

#Feature extraction for terminal Voltage
cvoltage_max=[]
t_cvoltage_max=[]

for i in charging_data.keys():

    for j in range(len(charging_data[i]['voltage_battery'])):
        #print(len(charging_data[i]['voltage_battery']))
        if charging_data[i]['voltage_battery'][j]>=4.2:
            temp=charging_data[i]['voltage_battery'][j]
```

Finally extracted data along with corresponding capacity feature converted into a dataframe and saved into CSV format (as shown below). Appropriate folder path is needed as these file will be used during model creation.

In similar way four csv files are created.

1. test1.csv

2. test2.csv

3. test3.csv

4. test4.csv

```
df=pd.DataFrame(
f'Charge_Voltage':cvoltage_max,
 'Charge_Voltage_time':t_cvoltage_max,
 'Charge_Current':ccurrent_drop,
 'Charge_Current_time':t_ccurrent_drop,
 'Charge_Temperature':ctemperature_max,
 'Charge_Temperature_time':t_ctemperature_max,
 'Charge_Loadcurrent':ccurrent_loaddrop,
 'Charge_Loadcurrent_Time':t_ccurrent_loaddrop,
 'Charge_Loadvoltage':cvoltage_loadmax,
 'Charge_Loadvoltage_Time':t_cvoltage_loadmax,
 'Discharge_Voltage':dvoltage_max,
 'Discharge_Voltage_Time':t_dvoltage_max,
 'Discharge_Current':dcurrent_max,
 'Discharge_Current_Time':t_dcurrent_max,
 'Discharge_Temperature':dtemperature_max,
 'Discharge_Temperature_Time':t_dtemperature_max,
 'Discharge_Loadcurrent':dcurrent_loadmax,
 'Discharge_Loadcurrent_Time':t_dcurrent_loadmax,
 'Discharge_Loadvoltage':dvoltage_loaddrop,
 'Discharge_Loadvoltage_Time':t_dvoltage_loaddrop,g
)

#Capacity Calculation for corresponding Features
df['capacity']=df['Discharge_Voltage']
                df['Discharge_Current']


#Exporting the final file in desired path
df.to_csv('D:nnSEM 3nnResearchnntest2.csv')
```

These files have 21 dimensions, which will be used in model creation step.

3.4 Modelling

Model creation is done in google colab. At the start all the required libraries were imported into the colab and then all extracted parameter file loaded into the colab environment (see codes below).

```
# Importing all the required libraries
from keras.layers import Input, Dense
from keras.models import Model
from sklearn.preprocessing import MinMaxScaler
import pandas as pd
import numpy as np
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from math import sqrt
#Loading data file from the saved path to the
colab Environment
datafile1=pd.read_csv('/content/test1.csv')
datafile2=pd.read_csv('/content/test2.csv')
datafile3=pd.read_csv('/content/test3.csv')
#testfile=pd.read_csv('/content/test4.csv')
datafile1.drop('Unnamed: 0', inplace=True, axis=1)
datafile2.drop('Unnamed: 0', inplace=True, axis=1)
datafile3.drop('Unnamed: 0', inplace=True, axis=1)
#testfile.drop('Unnamed: 0', inplace=True, axis=1)
```

Once the data is in the colab environment, all 21 dimension battery files are concatenated and the format of the file can be seen in below codes.

```
#Combining all the files
datafile1['rul']=temp2
datafile2['rul']=temp2
datafile3['rul']=temp2
data=pd.concat([datafile1,datafile2,datafile3])
(data)

#Output
        Charge_Voltage    Charge_Voltage_time     RUL
0       4.207509          712.453  1.500215       168
1       4.211770          3397.672                167
2       4.212005          3381.891                166
...     ...       ...     ...            ...
163     4.208010          2031.812                5
164     4.207997          2027.657                4
```

RUL for each reading is stored into 'train_y'. This is the target variable which is used during the training of the prediction model (see codes below).

```
train_y=train_y.values
print(train_y)

#Output
[168 167 166 165 164 163 162 161 160 159 158 157 156 155
154 153 152 151 150 149 148 147 146 145 144 143 142 141
140 139 138 137 136 135 134 133 132 131 130 129 128 127
126 125 124 123 122 121 120 119 118 117 116 115 114 113
112 111 110 109 108 107 106 105 104 103 102 101 100  99
98   97 96   95   94   93   92   91   90   89   88   87   86   85
84   83   82   81   80   79 78   77   76   75   74   73   72   71
70   69   68   67   66 65   64   63   62   61 60   59   58   57
56   55   54   53   52   51   50   49   48 47   46   45   44   43
```

At this stage data was normalized using minimum-maximum scalar (see codes below).

```
#Normalization of the data

scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(data)
```

Once data is normalized, next step is to split the data into training and testing subset (see codes below).

```
#Split Data in Training and Testing Subsets

from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split
(dataset, train_y, test_size=0.10, random_state=42)
```

As the data is split in training and testing data. Now we move to the autoencoder training part. It is trained for the 1000 epoch with batch size of 12 (see below codes).

```
encoding_dim = 15

input_df = Input(shape=(21,))
encoded = Dense(encoding_dim, activation='relu')(input_df)
decoded = Dense(21, activation='sigmoid')(encoded)

# encoder
autoencoder = Model(input_df, decoded)

# intermediate result
encoder = Model(input_df, encoded)

autoencoder.compile(optimizer='adadelta',
                loss='mean_squared_error')

history=autoencoder.fit(X_train, X_train,
                epochs=1000,
                batch_size=12,
                shuffle=True,
                validation_data=(X_val, X_val)
                )

#Output
Epoch 1/1000
38/38   0s 3ms/step   loss: 0.1238   val_loss: 0.1165
Epoch 2/1000
38/38   0s 1ms/step   loss: 0.1237   val_loss: 0.1165
Epoch 3/1000
38/38   0s 2ms/step   loss: 0.1237   val_loss: 0.1164
Epoch 4/1000
38/38   0s 1ms/step   loss: 0.1236   val_loss: 0.1164
Epoch 5/1000
38/38   0s 2ms/step   loss: 0.1236   val_loss: 0.1163
Epoch 6/1000
38/38   0s 2ms/step   loss: 0.1235   val_loss: 0.1163
Epoch 7/1000
38/38   0s 2ms/step   loss: 0.1235   val_loss: 0.1162
Epoch 8/1000
38/38   0s 1ms/step   loss: 0.1234   val_loss: 0.1162
.
.
Epoch 998/1000
38/38   0s 1ms/step   loss: 0.0778   val_loss: 0.0719
Epoch 999/1000
38/38   0s 1ms/step   loss: 0.0778   val_loss: 0.0718
Epoch 1000/1000
38/38   0s 1ms/step   loss: 0.0777   val_loss: 0.0718
```

At this stage 15-dimensional data from the autoencoder is combined with the target variable. Next neural network for the prediction is coded and trained on the 15-dimensional data with corresponding target variable (see codes below). This model is trained for 500 epochs with the batch size of 12. This model uses 'adadelta' as optimizer and 'Mean square Error as loss function'

```
#Prediction Neural Network
from keras.layers import Input, Dense, Dropout
def nn_model():

    inp = Input((15,))

    dense = Dense(512, activation='relu')(inp)
    dense = Dropout(0.2)(dense)
    out = Dense(1, activation='sigmoid')(dense)

    model = Model(inputs=inp, outputs=out)
    #print (model.summary())

    return model
#Taining the prediction model
model = nn_model()
model.compile(loss='mean_squared_error', optimizer='adam')
history=model.fit(X_train, y_train,
                      epochs=500,
                      batch_size=16,

                      )

#Output
Epoch 1/500
29/29   0s 1ms/step   loss: 0.0682
Epoch 2/500
29/29   0s 1ms/step   loss: 0.0502
Epoch 3/500
29/29   0s 1ms/step   loss: 0.0434
.
.
Epoch 497/500
29/29   0s 1ms/step   loss: 0.0062
Epoch 498/500
29/29   0s 1ms/step   loss: 0.0057
Epoch 499/500
29/29   0s 1ms/step   loss: 0.0061
Epoch 500/500
29/29   0s 1ms/step   loss: 0.0062
```

3.5 Model Evaluation

Model is tested on the testing subset for various combination. Below is the best result from the model. Below code shows the coding of R-square and Mean Square Error (MSE).

```
#Evaluation of the Prediction Model

r2_score(pred, y_test)
mean_squared_error(y_test, pred)


#Output
0.003787241922082548

#R square of the model
r2_score(pred, y_test)


#Output
0.9569411206628073
```

References

Goebel, K., Saha, B., Saxena, A., Celaya, J. R. and Christophersen, J. P. (2008). Prognostics in battery health management, *IEEE Instrumentation Measurement Magazine* **11**(4): 33–40.