Configuration Manual

MSc Research Project
Data Analytics

Sankara Subramanian Venkatraman
Student ID: x18179541

School of Computing
National College of Ireland

Supervisor: Mr. Hicham Rifai

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Sankara Subramanian Venkatraman
Student ID:	x18179541
Programme:	Data Analytics
Year:	2020
Module:	MSc Research Project
Supervisor:	Mr. Hicham Rifai
Submission Due Date:	17/08/2020
Project Title:	Configuration Manual
Word Count:	1279
Page Count:	15

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:	16th August 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

Sankara Subramanian Venkatraman
x18179541

1 Introduction

This configuration manual instructs the user to replicate the research project **"Big Data-driven Performance Improvement of Traffic Flow Prediction and Speed Limit Classification using Deep Learning"**. It represents the storage, databases, software and hardware requirements, programming languages, and system setup used in the implementation of the research.

2 System Configuration

Data sourced from the United Kingdom website [1] is huge and difficult to processes in the local system, so, the cloud system is preferred. Out of different cloud platforms Google Cloud Platform (GCP) *Google Cloud Platform (GCP) documentation* (n.d.) is chosen. It also provides free promotional credits of 276.60 euros for students shown in Figure 1.

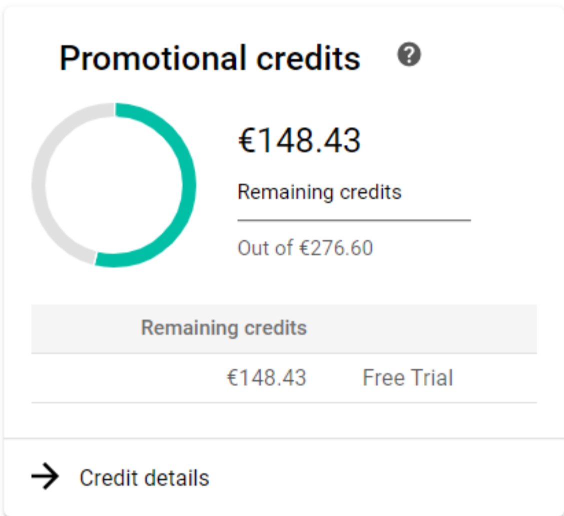


Figure 1: Promotional Credits in GCP

2.1 Storage

The raw data sourced from the United Kingdom is stored in the Google Cloud Storage (GCS) of Google Cloud Platform [2]. The road safety data from the year 2010 to 2018 for

[1] https://data.gov.uk/dataset/

[2] https://cloud.google.com/storage/docs/creating-buckets

the United Kingdom is stored in the folder **accidents** and traffic flow data is stored in **traffic_flow** folder. After data pre-processing, the data is stored in **cleaned_data** folder.

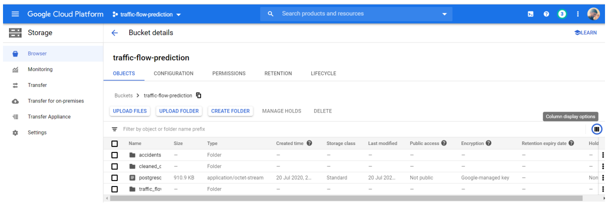
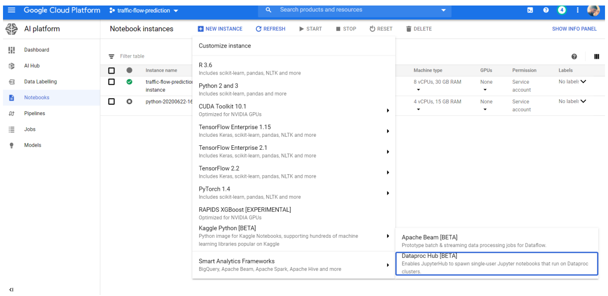


Figure 2: Google Cloud Storage Bucket

2.2 Hardware

This step can be configured using **Navigation Menu** —> **AI platform** —> **Notebook** —> **New Instance** —> **Smart Analytics Framework** —> **DataProc Hub** [**BETA**] Figure 3a [3]. The required machine type, memory, CPU cores, GPU type and storage disk can be configured according to the requirements. The region of the compute engine should be in the EU according to GDPR. The instance provisioned in GCP compute engine has the below system configuration shown in the Figure 3b *Data Engineering with Google Cloud Professional Certificate* (n.d.).



(a) Machine Hardware setup

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                8
On-line CPU(s) list:   0-7
Thread(s) per core:    2
Core(s) per socket:    4
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 79
Model name:            Intel(R) Xeon(R) CPU @ 2.20GHz
Stepping:              0
CPU MHz:               2200.190
BogoMIPS:              4400.38
Hypervisor vendor:     KVM
Virtualization type:   full
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              56320K
NUMA node0 CPU(s):     0-7
```

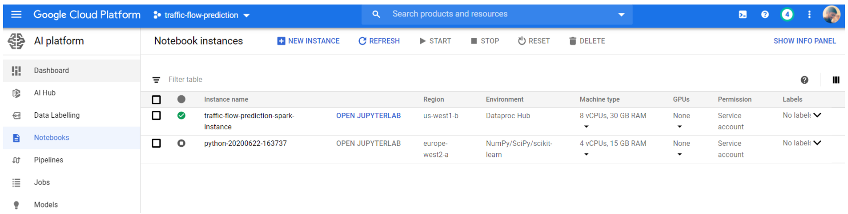
(b) System Configuration

Figure 3: Hardware and System configuration

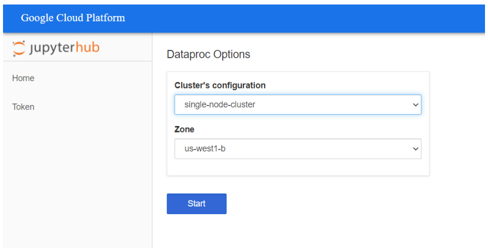
[3] https://cloud.google.com/ai-platform/notebooks/docs/create-new#before_you_begin

2.3 Software

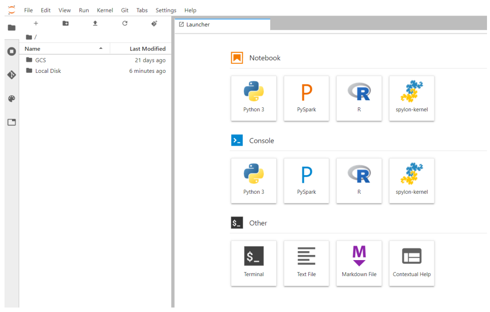
The software required for the analysis can also be configured using the option available in AI Notebook of GCP. Once the instance is provisioned, Jupyter Notebook with PySpark, Python3 and shell kernels are chosen using DataProc cluster's configuration option [4]. **Navigation Menu —> AI platform —> Notebook —> OPEN JUPYTERLAB (Instance name) —> Cluster's configuration**. The DataProc cluster is launched in a separate instance with Python3, PySpark and shell kernels software installed by default. Single-node-cluster is provisioned for this research. Finally, the Jupyter notebook with all the software components required for data cleaning, transformation, feature extraction and data mining models are obtained and shown in Figure 4.



(a)



(b)



(c)

Figure 4: Software requirements

2.4 Database

After the data cleaning and transformation process using SparkSQL, the data is stored in the PostgreSQL database for Exploratory Data Analysis (EDA) and performing the final analysis using deep learning models. It can also provisioned in GCP [5] using the option **SQL —> CREATE INSTANCE —> PostgreSQL** (Figure 6). The instance name, password, location, region of data storage and database version can be chosen according to the requirements. The instance created for the research is shown in Figure 5.

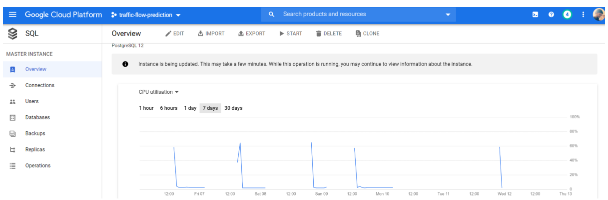


Figure 5: Project Instance

[4] https://cloud.google.com/dataproc/docs/concepts/components/jupyter

[5] https://cloud.google.com/sql/docs/postgres/

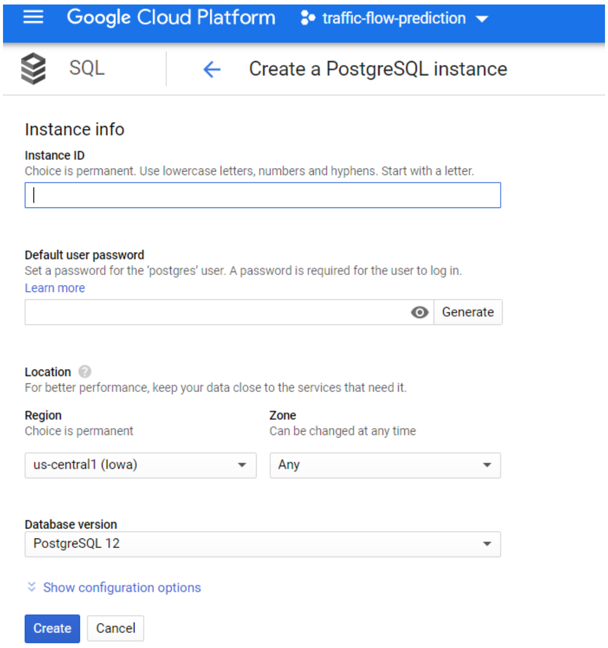
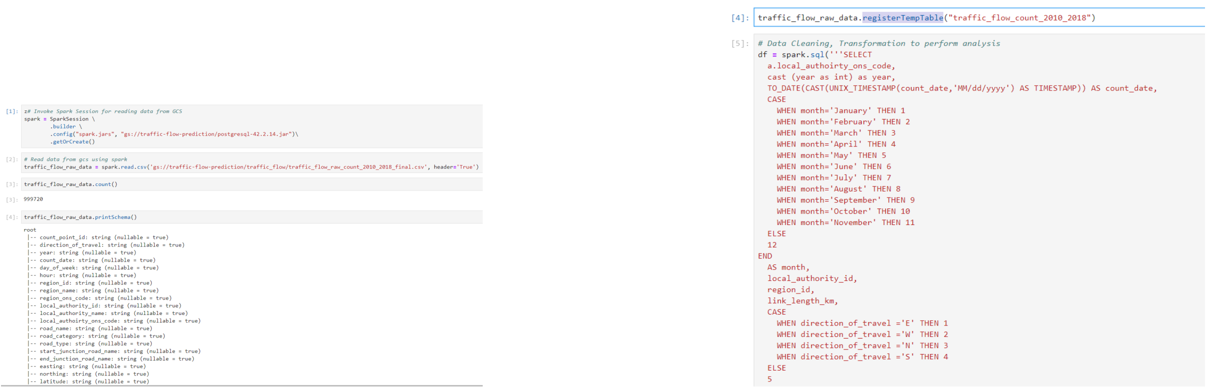


Figure 6: PostgreSQL in GCP

3 Data Preparation and Pre-Processing

Initially, the data downloaded from the United Kingdom website was in zip format for traffic flow data and CSV format for road safety data. Then the data from the zip file is extracted manually and stored in CSV format. Both, the files are then moved to GCP storage as mentioned in Section 2.1. Using the Dataproc instance mentioned in Section 2.2 and 2.3 with PySpark, Python3 and shell kernels installed, data cleaning and transformation is performed. The first part of all 4 **.ipynb** files uploaded in the code artefacts is used for data cleaning and transformation. SparkSession is initiated to read the data from GCS as shown in Figure 7a.

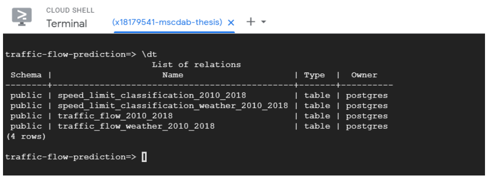


(a) SparkSession

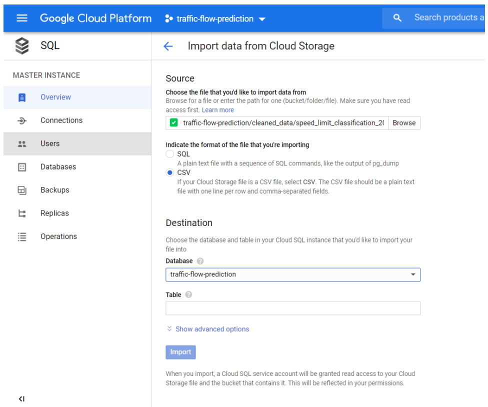
(b) Spark TempTable

Figure 7: SparkSession and TempTable

Once the data read from GCS, the data is stored in Temporary tables **(registerTempTable)**, for performing data cleaning and transformation using spark.sql is shown in Figure 7b. In the next step, the data is stored back to the **cleaned_data** folder. As the research is carried as 4 experiments, the final tables required for all the experiments is created in PostgreSQL in prior using Data Definition Language (DDL) function of SQL shown in Figure 9a. **PostgreSQL_DDL_Tables.sql** available in the code artefacts is used to create tables in the PostgreSQL database. Finally, the cleaned transformed data stored in the GCS is uploaded from the Web UI using the option **SQL –> Overview –> Import –> Source (GCS bucket), Destination (traffic-flow-prediction), Table (Table Name)** as shown in Figure 9b.



(a) Postgres Tables

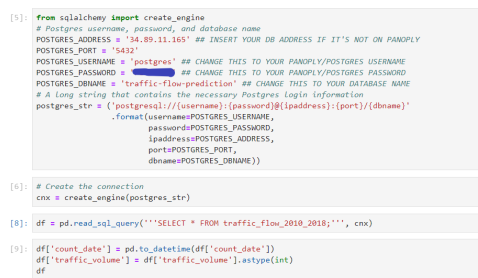


(b) Data Load in PostgreSQL

Figure 8: PostgreSQL Tables and Data Load

4 Exploratory Data Analysis

The data stored in PostgreSQL is read through create_engine of **sqlalchemy** and **Psycopg2** package of Python3 shown in Figure 9 and stored in pandas dataframe. The second part of **.ipynb** files is used for Exploratory Data Analysis (EDA). It is carried using the stats function of **scipy** module, ADF test is conducted using the module **statsmodels.tsa.stattools** and analysis are visualized using ticker, pyplot of **matplotlib** and **seaborn** modules. The below codes are designed using Python language only.



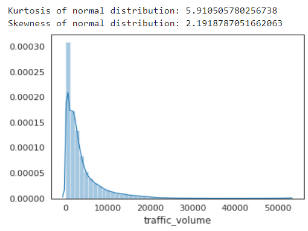
(a) Data read from Pandas

(b) Exploratory Data Analysis

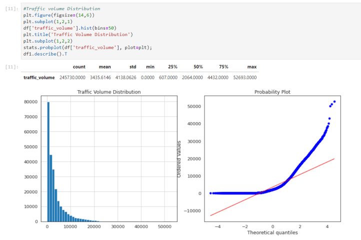
Figure 9: Pandas data read and Data Analysis

4.1 Experiment-1 Statistical Analysis Results

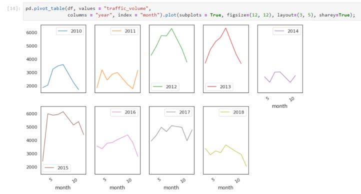
Figure 10 represents the code for skewness, kurtosis, traffic volume distribution, probability plot, seasonality from the year 2010 to 2018 without non-traffic parameters.



(a) Skewness and Kurtosis



(b) Traffic Distribution without non-traffic parameters

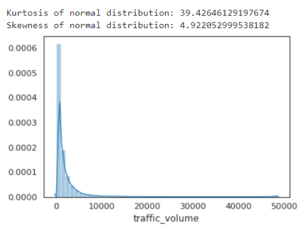


(c) Seasonality Traffic Volume 2010-2018

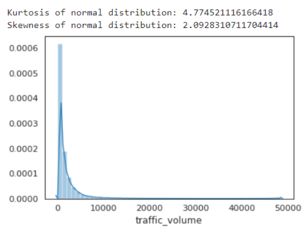
Figure 10: Exploratory Data Analysis Exp-1

4.2 Experiment-2 Statistical Analysis Results

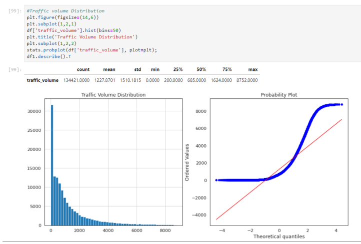
Figure 11 represents the code for skewness, kurtosis, traffic volume distribution, probability plot, seasonality from the year 2010 to 2018 with non-traffic parameters of weather, light and road surface conditions. Initially, the skewness value is out of range, due to outliers, then it has been removed and brought into the range of -10 to 10.



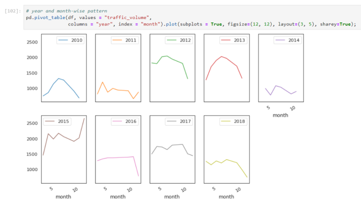
(a) Skewness and Kurtosis before removing outliers



(b) Skewness and Kurtosis after removing outliers



(c) Traffic Distribution with non-traffic parameters

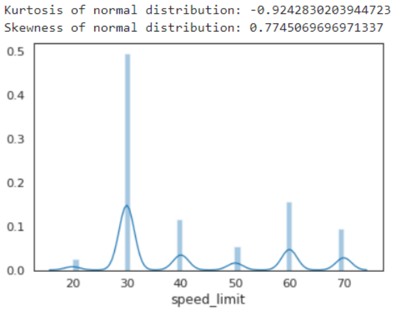


(d) Seasonality Traffic Volume with non-traffic parameters 2010-2018

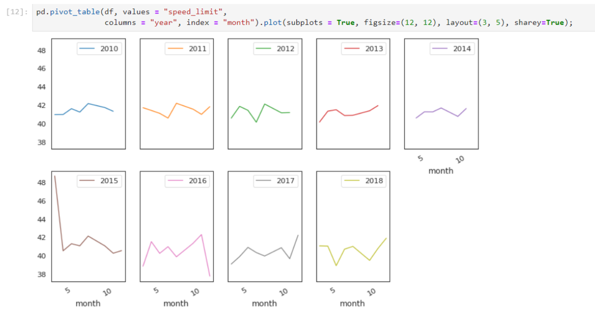
Figure 11: Exploratory Data Analysis Exp-2

4.3 Experiment-3 Statistical Analysis Results

Figure 12 represents the code for skewness, kurtosis and seasonality of traffic speed from the year 2010 to 2018 without non-traffic parameters.



(a) Skewness and Kurtosis

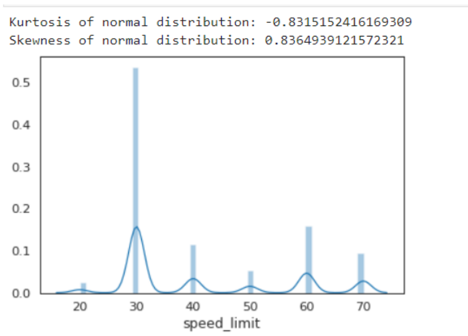


(b) Seasonality of Speed Limit without non-traffic parameters 2010-2018

Figure 12: Exploratory Data Analysis Exp-3

4.4 Experiment-4 Statistical Analysis Results

Figure 13 represents the code for skewness, kurtosis and seasonality of traffic speed from the year 2010 to 2018 with non-traffic parameters of weather, light and road surface conditions.



(a) Skewness and Kurtosis

(b) Seasonality of Speed Limit with non-traffic parameters 2010-2018

Figure 13: Exploratory Data Analysis Exp-4

5 Feature Selection

After, data stationary check using the dickey-fuller test, the data is pivoted using pandas **pd.pivot_table** and aggregate the traffic flow based on "year" from 2010 to 2018. After pivoting the data, MinMaxScaler normalization is applied from **sklearn.preprocessing** to normalize the data. The data is split into train and test using train_test_split of **sklearn.model_selection**. Finally, using reshape of **numpy**, the data is reshaped from (data, features) to (training data, time steps, features) and (testing data, time steps, features). The below code snippets show the pivoting, normalization, train-test split and reshape feature incorporated in the analysis.

5.1 Experiment-1 Feature Selection

Traffic-only parameter of traffic volume is used in the experiment.

```python
[17]: # Converting traffic-volume to time-series based data
import pandas as pd
import numpy as np
traffic_data_year_wise = pd.pivot_table(df, values='traffic_volume', index=['local_authoirty_ons_code', 'count_date','local_authority_id', 'reg
                                        'travel_direction', 'road_category', 'road_type','peak_non_peak_hours','month'],
                    columns= ['year'], aggfunc=np.sum, fill_value=0).rename_axis(None, axis=1).reset_index()
```

```python
[114]: import numpy
import matplotlib.pyplot as plt
import pandas
import math
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from tensorflow.keras.callbacks import EarlyStopping
from keras.layers import Dropout
from keras.layers import *
```

(a) Pivot Table Structure

```python
[121]: # normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
```

```python
[122]: # split into train and test sets
train = dataset[:,0:-1]
print(train.shape)
test = dataset[:,13]
print(test.shape)

(245730, 13)
(245730,)
```

```python
[123]: from sklearn.model_selection import train_test_split
trainX, testX, trainY, testY = train_test_split(train,test, test_size = 0.10, random_state = 42)
```

```python
[124]: # reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
```

```python
[125]: print(trainX.shape, testX.shape,trainY.shape, testY.shape)

(221157, 1, 13) (24573, 1, 13) (221157,) (24573,)
```

(b) Data Normalization

Figure 14: Feature Selection Exp-1

5.2 Experiment-2 Feature Selection

Traffic volume and non-traffic parameters of weather, light and road surface conditions are used in this experiment.

```python
[115]: Final_data = pd.merge(pd.merge(weather_data_year_wise, (pd.merge(light_data_year_wise, road_data_year_wise,
                                             on =['local_authoirty_ons_code','count_date','local_authority_id','region_id','link_length_km',
                                                'travel_direction', 'road_category', 'road_type',
                                                  'peak_non_peak_hours', 'month'], how='outer'))), traffic_data_year_wise,
                                        on =['local_authoirty_ons_code','local_authority_id','region_id','link_length_km',
                                                'travel_direction', 'road_category', 'road_type',
                                               'peak_non_peak_hours', 'month','count_date'], how='outer')
```

```python
[116]: Final_data
```

[116]:

	local_authoirty_ons_code	count_date	local_authority_id	region_id	link_length_km	travel_direction	road_category	road_type	peak_non_peak_hours	month	...
0	E06000001	2010-03-15	132	11	0.0000	1	5	2	2	3	...
1	E06000001	2010-03-15	132	11	0.0000	2	5	2	2	3	...
2	E06000001	2010-04-22	132	11	6.6000	3	3	1	1	4	...
3	E06000001	2010-04-22	132	11	6.6000	4	3	1	1	4	...
4	E06000001	2010-06-15	132	11	0.0000	3	5	2	1	6	...

(a) Pivot Table Structure

```python
[159]: # load the dataset and overall traffic dataset
dataframe = Final_data.iloc[:,5:]
#dataframe = dataframe[dataframe['peak_non_peak_hours']==2]
dataset = dataframe.values
dataset = dataset.astype('float32')
```

```python
[160]: # normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
```

```python
[161]: # split into train and test sets
train = dataset[:,0:-1]
print(train.shape)
test = dataset[:,40]
print(test.shape)

(116561, 40)
(116561,)
```

```python
[163]: from sklearn.model_selection import train_test_split
trainX, testX, trainY, testY = train_test_split(train,test, test_size = 0.10, random_state = 42)
```

```python
[164]: # reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
```

```python
[165]: print(trainX.shape,testX.shape)

(104904, 1, 40) (11657, 1, 40)
```

(b) Data Normalization

Figure 15: Feature Selection Exp-2

5.3 Experiment-3 Feature Selection

The traffic speed limit is classified into 2 categories of Low Speed (1) and High Speed (2). Using to_categorical of **Keras.utils.np_utils**, the target column is converted to a categorical value. As the data is not a continuous value, normalization is not required for the classification problem.

```
[107]: # Converting traffic-volume to time-series based data
       import pandas as pd
       import numpy as np
       speed_limit_data_year_wise = pd.pivot_table(df, values='speed_limit', index=['local_authoirty_ons_code', 'count_date' ,'local_authority_id', 'r
                                   'travel_direction', 'road_category', 'road_type','peak_non_peak_hours'],
                  columns= 'year',aggfunc=np.median, fill_value=0).rename_axis(None, axis=1).reset_index()

[108]: speed_limit_data_year_wise[2010].unique()

[108]: array([30,  0, 45, 35, 20, 50, 40, 60, 70, 25, 55, 65])

[109]: speed_limit_data_year_wise[2010] = np.where(speed_limit_data_year_wise[2010].between (0,20),1, 2)
       speed_limit_data_year_wise[2011] = np.where(speed_limit_data_year_wise[2011].between (0,20),1, 2)
       speed_limit_data_year_wise[2012] = np.where(speed_limit_data_year_wise[2012].between (0,20),1, 2)
       speed_limit_data_year_wise[2013] = np.where(speed_limit_data_year_wise[2013].between (0,20),1, 2)
       speed_limit_data_year_wise[2014] = np.where(speed_limit_data_year_wise[2014].between (0,20),1, 2)
       speed_limit_data_year_wise[2015] = np.where(speed_limit_data_year_wise[2015].between (0,20),1, 2)
       speed_limit_data_year_wise[2016] = np.where(speed_limit_data_year_wise[2016].between (0,20),1, 2)
       speed_limit_data_year_wise[2017] = np.where(speed_limit_data_year_wise[2017].between (0,20),1, 2)
       speed_limit_data_year_wise[2018] = np.where(speed_limit_data_year_wise[2018].between (0,20),1, 2)
```

(a) Pivot Table Structure

```
[115]: x = speed_limit_data_year_wise.iloc[:,5:-1]

[116]: y = speed_limit_data_year_wise.iloc[:,-1]

[117]: from keras.utils import to_categorical
       from keras.utils.np_utils import to_categorical
       y = to_categorical(y)

[118]: y.shape

[118]: (45374, 3)

[119]: x = np.array(x[:])

[120]: from sklearn.model_selection import train_test_split
       x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.1,random_state = 42)

[121]: x_train = x_train.reshape(x_train.shape[0],x_train.shape[1], 1)
       x_test = x_test.reshape(x_test.shape[0],x_test.shape[1], 1)

[122]: print(x_train.shape)
       print(x_test.shape)

       (40836, 13, 1)
       (4538, 13, 1)

[123]: print(y_train.shape)
       print(y_test.shape)

       (40836, 3)
       (4538, 3)
```

(b) Data Reshape

Figure 16: Feature Selection Exp-3

5.4 Experiment-4 Feature Selection

The traffic speed limit is classified into 2 categories of Low Speed (1) and High Speed (2) along with the non-traffic parameter of weather, light and road surface conditions. The target column is converted to a categorical value.

```
[26]: Final_data = pd.merge(pd.merge(weather_data_year_wise, (pd.merge(light_data_year_wise, road_data_year_wise,
                                on =['local_authoirty_ons_code','count_date','local_authority_id','region_id','link_length_km',
                                  'travel_direction', 'road_category', 'road_type',
                                   'peak_non_peak_hours', 'month'], how='outer'))), speed_limit_data_year_wise,
                             on =['local_authoirty_ons_code','local_authority_id','region_id','link_length_km',
                                  'travel_direction', 'road_category', 'road_type',
                                 'peak_non_peak_hours', 'month','count_date'], how='outer')

[115]: Final_data.head(5)
```

[115]:

	local_authoirty_ons_code	count_date	local_authority_id	region_id	link_length_km	travel_direction	road_category	road_type	peak_non_peak_hours	month	...	road_
0	E06000001	2010-03-15	132	11	0.0	1	5	2	2	3	...	
1	E06000001	2010-03-15	132	11	0.0	2	5	2	2	3	...	
2	E06000001	2010-04-22	132	11	6.6	3	3	1	1	4	...	
3	E06000001	2010-04-22	132	11	6.6	4	3	1	1	4	...	
4	E06000001	2010-06-15	132	11	0.0	3	5	2	1	6	...	

5 rows × 46 columns

(a) Pivot Table Structure

```
[33]: from keras.utils import to_categorical
      y = to_categorical(y)
      #from sklearn.preprocessing import OneHotEncoder
      #enc = OneHotEncoder(drop='if_binary')
      #y = pd.DataFrame(enc.fit_transform(speed_limit_data_year_wise[['speed_limit_2018']]).toarray())

[34]: x = np.array(x[:])

[35]: from sklearn.model_selection import train_test_split
      x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.1,random_state = 42)

[36]: x_train = x_train.reshape(x_train.shape[0],x_train.shape[1], 1)
      x_test = x_test.reshape(x_test.shape[0],x_test.shape[1], 1)

[37]: print(x_train.shape)
      print(x_test.shape)
      print(y_train.shape)
      print(y_test.shape)

      (113399, 40, 1)
      (12600, 40, 1)
      (113399, 3)
      (12600, 3)
```

(b) Data Reshape

Figure 17: Feature Selection Exp-4

6 Code used for Deep Learning Models

Using Keras and TensorFlow package of Python3, the implementation of deep learning models is carried out. Also, hyper-parameters of epochs, batch size and train-test split ratio are changed and tested *Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization* (n.d.).

6.1 Experiment-1 LSTM Model Traffic-only parameters

Different LSTM models of vanilla-LSTM, stacked-LSTM and Bi-directional LSTM models are applied to the trained dataset, and models are predicted and evaluated.

```
[126]: # create and fit the LSTM network
from keras.layers import Bidirectional
model = Sequential()

#model.add(LSTM(100, activation='softmax', return_sequences=True, input_shape=(1, 13))) #stacked-LSTM
model.add(LSTM(100, activation='relu', input_shape=(1, 13))) # Vanilla- LSTM
#model.add(Bidirectional(LSTM(50, activation='relu'), input_shape=(1, 13))) #Bi-directional-LSTM
#model.add(LSTM(50, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam', metrics=['accuracy', 'mae'])
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_6 (LSTM)                (None, 100)               45600
_________________________________________________________________
dropout_5 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 101
=================================================================
Total params: 45,701
Trainable params: 45,701
Non-trainable params: 0
```

(a) LSTM Model Summary

```
[127]: history = model.fit(trainX, trainY, epochs=50, batch_size=50, validation_data=(testX, testY),validation_split=0.10,
                    callbacks=[EarlyStopping(monitor='val_loss', patience=10)],verbose=1, shuffle=False)
```

```
Epoch 1/50
3981/3981 [==============================] - 11s 3ms/step - loss: 0.0083 - accuracy: 0.9249 - mae: 0.0083 - val_loss: 0.0074 - val_accuracy: 0.9264 - val_mae: 0.0074
Epoch 2/50
3981/3981 [==============================] - 10s 3ms/step - loss: 0.0068 - accuracy: 0.9249 - mae: 0.0068 - val_loss: 0.0054 - val_accuracy: 0.9264 - val_mae: 0.0054
Epoch 3/50
3981/3981 [==============================] - 10s 3ms/step - loss: 0.0063 - accuracy: 0.9249 - mae: 0.0063 - val_loss: 0.0056 - val_accuracy: 0.9264 - val_mae: 0.0056
Epoch 4/50
3981/3981 [==============================] - 10s 3ms/step - loss: 0.0062 - accuracy: 0.9249 - mae: 0.0062 - val_loss: 0.0054 - val_accuracy: 0.9264 - val_mae: 0.0054
Epoch 5/50
3981/3981 [==============================] - 10s 3ms/step - loss: 0.0062 - accuracy: 0.9249 - mae: 0.0062 - val_loss: 0.0050 - val_accuracy: 0.9264 - val_mae: 0.0050
Epoch 6/50
3981/3981 [==============================] - 10s 3ms/step - loss: 0.0062 - accuracy: 0.9249 - mae: 0.0062 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mae: 0.0055
Epoch 7/50
3981/3981 [==============================] - 10s 3ms/step - loss: 0.0061 - accuracy: 0.9249 - mae: 0.0061 - val_loss: 0.0053 - val_accuracy: 0.9264 - val_mae: 0.0053
Epoch 8/50
3981/3981 [==============================] - 10s 3ms/step - loss: 0.0061 - accuracy: 0.9249 - mae: 0.0061 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mae: 0.0055
Epoch 9/50
3981/3981 [==============================] - 10s 3ms/step - loss: 0.0061 - accuracy: 0.9249 - mae: 0.0061 - val_loss: 0.0045 - val_accuracy: 0.9264 - val_mae: 0.0045
Epoch 10/50
3981/3981 [==============================] - 10s 3ms/step - loss: 0.0061 - accuracy: 0.9249 - mae: 0.0061 - val_loss: 0.0050 - val_accuracy: 0.9264 - val_mae: 0.0050
Epoch 11/50
3981/3981 [==============================] - 11s 3ms/step - loss: 0.0061 - accuracy: 0.9249 - mae: 0.0061 - val_loss: 0.0048 - val_accuracy: 0.9264 - val_mae: 0.0048
Epoch 12/50
3981/3981 [==============================] - 10s 3ms/step - loss: 0.0061 - accuracy: 0.9249 - mae: 0.0061 - val_loss: 0.0051 - val_accuracy: 0.9264 - val_mae: 0.0051
Epoch 13/50
3981/3981 [==============================] - 10s 3ms/step - loss: 0.0061 - accuracy: 0.9249 - mae: 0.0061 - val_loss: 0.0049 - val_accuracy: 0.9264 - val_mae: 0.0049
Epoch 14/50
3981/3981 [==============================] - 10s 3ms/step - loss: 0.0061 - accuracy: 0.9249 - mae: 0.0061 - val_loss: 0.0049 - val_accuracy: 0.9264 - val_mae: 0.0049
Epoch 15/50
3981/3981 [==============================] - 10s 2ms/step - loss: 0.0061 - accuracy: 0.9249 - mae: 0.0061 - val_loss: 0.0046 - val_accuracy: 0.9264 - val_mae: 0.0046
Epoch 16/50
3981/3981 [==============================] - 10s 3ms/step - loss: 0.0061 - accuracy: 0.9249 - mae: 0.0061 - val_loss: 0.0042 - val_accuracy: 0.9264 - val_mae: 0.0042
Epoch 17/50
```

(b) LSTM Model Fit

```
[128]: # make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
```

```
[129]: accr = model.evaluate(testX,testY)
print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr[0],accr[1]))
```

```
768/768 [==============================] - 1s 1ms/step - loss: 0.0051 - accuracy: 0.9236 - mae: 0.0051
Test set
  Loss: 0.005
  Accuracy: 0.924
```

```
[130]: # make a prediction
yhat = model.predict(testX)
testX = testX.reshape((testX.shape[0], testX.shape[2]))
```

(c) LSTM Model Predict

Figure 18: Deep Learning Model Code Snippet Exp-1

6.2 Experiment-2 LSTM Model Traffic and Non-traffic parameters

```
[166]: # create and fit the LSTM network
from keras.layers import Bidirectional
model = Sequential()
#model.add(LSTM(100, activation='softmax', return_sequences=True, input_shape=(1, 40)))
#model.add(Bidirectional(LSTM(100, activation='relu'), input_shape=(1, 40)))
#model.add(LSTM(50, activation='relu'))
model.add(LSTM(100, activation='relu', input_shape=(1, 40))) # Vanilla- LSTM
model.add(Dropout(0.5))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam', metrics=['accuracy','mae'])
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_11 (LSTM)               (None, 100)               56400
_________________________________________________________________
dropout_6 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 101
=================================================================
Total params: 56,501
Trainable params: 56,501
Non-trainable params: 0
_________________________________________________________________
```

(a) LSTM Model Summary

```
[167]: history = model.fit(trainX, trainY, epochs=50, batch_size=50, validation_data=(testX, testY),validation_split=0.10,
                callbacks=[EarlyStopping(monitor='val_loss', patience=10)],verbose=2, shuffle=False)
```

```
Epoch 1/50
1889/1889 - 5s - loss: 0.0046 - accuracy: 0.9329 - mae: 0.0046 - val_loss: 0.0039 - val_accuracy: 0.9355 - val_mae: 0.0039
Epoch 2/50
1889/1889 - 5s - loss: 0.0038 - accuracy: 0.9329 - mae: 0.0038 - val_loss: 0.0039 - val_accuracy: 0.9355 - val_mae: 0.0039
Epoch 3/50
1889/1889 - 5s - loss: 0.0037 - accuracy: 0.9329 - mae: 0.0037 - val_loss: 0.0039 - val_accuracy: 0.9355 - val_mae: 0.0039
Epoch 4/50
1889/1889 - 5s - loss: 0.0036 - accuracy: 0.9329 - mae: 0.0036 - val_loss: 0.0039 - val_accuracy: 0.9355 - val_mae: 0.0039
Epoch 5/50
1889/1889 - 5s - loss: 0.0036 - accuracy: 0.9329 - mae: 0.0036 - val_loss: 0.0036 - val_accuracy: 0.9355 - val_mae: 0.0036
Epoch 6/50
1889/1889 - 6s - loss: 0.0036 - accuracy: 0.9329 - mae: 0.0036 - val_loss: 0.0036 - val_accuracy: 0.9355 - val_mae: 0.0036
Epoch 7/50
1889/1889 - 5s - loss: 0.0036 - accuracy: 0.9329 - mae: 0.0036 - val_loss: 0.0036 - val_accuracy: 0.9355 - val_mae: 0.0036
Epoch 8/50
1889/1889 - 5s - loss: 0.0036 - accuracy: 0.9329 - mae: 0.0036 - val_loss: 0.0037 - val_accuracy: 0.9355 - val_mae: 0.0037
Epoch 9/50
1889/1889 - 5s - loss: 0.0036 - accuracy: 0.9329 - mae: 0.0036 - val_loss: 0.0036 - val_accuracy: 0.9355 - val_mae: 0.0036
Epoch 10/50
1889/1889 - 6s - loss: 0.0035 - accuracy: 0.9329 - mae: 0.0035 - val_loss: 0.0035 - val_accuracy: 0.9355 - val_mae: 0.0035
Epoch 11/50
1889/1889 - 5s - loss: 0.0036 - accuracy: 0.9329 - mae: 0.0036 - val_loss: 0.0036 - val_accuracy: 0.9355 - val_mae: 0.0036
Epoch 12/50
1889/1889 - 5s - loss: 0.0035 - accuracy: 0.9329 - mae: 0.0035 - val_loss: 0.0036 - val_accuracy: 0.9355 - val_mae: 0.0036
Epoch 13/50
1889/1889 - 5s - loss: 0.0036 - accuracy: 0.9329 - mae: 0.0036 - val_loss: 0.0040 - val_accuracy: 0.9355 - val_mae: 0.0040
Epoch 14/50
1889/1889 - 5s - loss: 0.0035 - accuracy: 0.9329 - mae: 0.0035 - val_loss: 0.0036 - val_accuracy: 0.9355 - val_mae: 0.0036
Epoch 15/50
1889/1889 - 5s - loss: 0.0035 - accuracy: 0.9329 - mae: 0.0035 - val_loss: 0.0037 - val_accuracy: 0.9355 - val_mae: 0.0037
Epoch 16/50
1889/1889 - 5s - loss: 0.0035 - accuracy: 0.9329 - mae: 0.0035 - val_loss: 0.0037 - val_accuracy: 0.9355 - val_mae: 0.0037
Epoch 17/50
```

(b) LSTM Model Fit

```
[168]: # make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
```

```
[78]: accr = model.evaluate(testX,testY)
print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr[0],accr[1]))
```

```
118/118 [==============================] - 0s 1ms/step - loss: 0.0084 - accuracy: 0.9285 - mae: 0.0084
Test set
  Loss: 0.008
  Accuracy: 0.928
```

```
[363]: # make a prediction
yhat = model.predict(testX)
```

```
[364]: testX = testX.reshape((testX.shape[0], testX.shape[2]))
```

```
[365]: from numpy import concatenate
# invert scaling for forecast
#inv_yhat = concatenate((yhat[:,0,:], testX[:, 0:]), axis=1)
inv_yhat = concatenate((yhat, testX[:, 0:]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]
```

(c) LSTM Model Predict

Figure 19: Deep Learning Model Code Snippet Exp-2

6.3 Experiment-3 CNN Model Traffic-only parameters

CNN model with 1 convolutional layer, 1 pooling layer and 3 hidden layers are applied to the traffic speed limit data.

```
[126]: model = Sequential()
       intput_shape=(x_train.shape[1], 1)
       model.add(Conv1D(128, kernel_size=3,padding = 'same',activation='relu', input_shape=input_shape))
       model.add(BatchNormalization())
       model.add(MaxPooling1D(pool_size=(2)))
       model.add(Flatten())
       model.add(Dense(64, activation='tanh'))
       model.add(Dropout(0.5))
       model.add(Dense(32, activation='relu'))
       model.add(Dropout(0.5))
       model.add(Dense(16, activation='relu'))
       model.add(Dropout(0.5))
       model.add(Dense(num_classes, activation='softmax'))
       model.summary()

       Model: "sequential_3"
       _________________________________________________________________
       Layer (type)                 Output Shape              Param #
       =================================================================
       conv1d_3 (Conv1D)            (None, 13, 128)           512
       _________________________________________________________________
       batch_normalization_3 (Batch (None, 13, 128)           512
       _________________________________________________________________
       max_pooling1d_3 (MaxPooling1 (None, 6, 128)            0
       _________________________________________________________________
       flatten_3 (Flatten)          (None, 768)               0
       _________________________________________________________________
       dense_12 (Dense)             (None, 64)                49216
       _________________________________________________________________
       dropout_9 (Dropout)          (None, 64)                0
       _________________________________________________________________
       dense_13 (Dense)             (None, 32)                2080
       _________________________________________________________________
       dropout_10 (Dropout)         (None, 32)                0
```

(a) CNN Model Summary

```
[127]: from keras.optimizers import SGD, RMSprop
       #opt = SGD(lr=0.01, momentum=0.9)
       opt = RMSprop(lr=0.001)
       model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

[128]: history = model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size, validation_data=(x_test, y_test),validation_split=0.10,
                           callbacks=[EarlyStopping(monitor='val_loss', patience=10)], verbose=2, shuffle=False)

       Epoch 1/50
       736/736 - 3s - loss: 0.1490 - accuracy: 0.9418 - val_loss: 0.0435 - val_accuracy: 0.9846
       Epoch 2/50
       736/736 - 3s - loss: 0.0527 - accuracy: 0.9812 - val_loss: 0.0372 - val_accuracy: 0.9860
       Epoch 3/50
       736/736 - 3s - loss: 0.0456 - accuracy: 0.9847 - val_loss: 0.0432 - val_accuracy: 0.9829
       Epoch 4/50
       736/736 - 3s - loss: 0.0459 - accuracy: 0.9839 - val_loss: 0.0377 - val_accuracy: 0.9858
       Epoch 5/50
       736/736 - 3s - loss: 0.0456 - accuracy: 0.9847 - val_loss: 0.0369 - val_accuracy: 0.9860
       Epoch 6/50
       736/736 - 3s - loss: 0.0396 - accuracy: 0.9860 - val_loss: 0.0372 - val_accuracy: 0.9860
       Epoch 7/50
       736/736 - 3s - loss: 0.0375 - accuracy: 0.9869 - val_loss: 0.0360 - val_accuracy: 0.9868
       Epoch 8/50
       736/736 - 3s - loss: 0.0376 - accuracy: 0.9865 - val_loss: 0.0359 - val_accuracy: 0.9868
       Epoch 9/50
       736/736 - 3s - loss: 0.0362 - accuracy: 0.9873 - val_loss: 0.0357 - val_accuracy: 0.9858
       Epoch 10/50
       736/736 - 3s - loss: 0.0371 - accuracy: 0.9867 - val_loss: 0.0359 - val_accuracy: 0.9868
       Epoch 11/50
       736/736 - 3s - loss: 0.0367 - accuracy: 0.9871 - val_loss: 0.0359 - val_accuracy: 0.9868
       Epoch 12/50
       736/736 - 3s - loss: 0.0374 - accuracy: 0.9866 - val_loss: 0.0365 - val_accuracy: 0.9863
       Epoch 13/50
       736/736 - 3s - loss: 0.0362 - accuracy: 0.9871 - val_loss: 0.0359 - val_accuracy: 0.9868
       Epoch 14/50
       736/736 - 3s - loss: 0.0372 - accuracy: 0.9866 - val_loss: 0.0359 - val_accuracy: 0.9868
```

(b) CNN Model Fit

```
[129]: y_pred = model.predict(x_test, batch_size=10,
                              verbose=1)
       score = model.evaluate(x_test, y_test,verbose=1)

       print(score)

       454/454 [==============================] - 0s 769us/step
       142/142 [==============================] - 0s 1ms/step - loss: 0.0356 - accuracy: 0.9870
       [0.03563347086310387, 0.9869986772537231]
```

(c) CNN Model Predict

Figure 20: Deep Learning Model Code Snippet Exp-3

6.4 Experiment-4 CNN Model Traffic and Non-traffic parameters

```
[41]: model = Sequential()
intput_shape=(x_train.shape[1], 1)
model.add(Conv1D(128, kernel_size=3,padding = 'same',activation='relu', input_shape=input_shape))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=(2)))
model.add(Flatten())
model.add(Dense(64, activation='tanh'))
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(16, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 40, 128)           512
_________________________________________________________________
batch_normalization (BatchNo (None, 40, 128)           512
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 20, 128)           0
_________________________________________________________________
flatten (Flatten)            (None, 2560)              0
_________________________________________________________________
dense (Dense)                (None, 64)                163904
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                2080
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 16)                528
```

(a) CNN Model Summary

```
[42]: from keras.optimizers import SGD
#opt = SGD(lr=0.01, momentum=0.9)
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
#model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
[43]: history = model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size, validation_data=(x_test, y_test),validation_split=0.10,
                    callbacks=[EarlyStopping(monitor='val_loss', patience=10)], verbose=2, shuffle=False)
```

```
Epoch 1/50
2042/2042 - 15s - loss: 0.0634 - accuracy: 0.9794 - val_loss: 0.0162 - val_accuracy: 0.9958
Epoch 2/50
2042/2042 - 14s - loss: 0.0207 - accuracy: 0.9952 - val_loss: 0.0161 - val_accuracy: 0.9958
Epoch 3/50
2042/2042 - 14s - loss: 0.0196 - accuracy: 0.9948 - val_loss: 0.0162 - val_accuracy: 0.9958
Epoch 4/50
2042/2042 - 14s - loss: 0.0180 - accuracy: 0.9952 - val_loss: 0.0162 - val_accuracy: 0.9958
Epoch 5/50
2042/2042 - 14s - loss: 0.0170 - accuracy: 0.9954 - val_loss: 0.0161 - val_accuracy: 0.9958
Epoch 6/50
2042/2042 - 14s - loss: 0.0171 - accuracy: 0.9953 - val_loss: 0.0162 - val_accuracy: 0.9958
Epoch 7/50
2042/2042 - 14s - loss: 0.0173 - accuracy: 0.9953 - val_loss: 0.0161 - val_accuracy: 0.9958
Epoch 8/50
2042/2042 - 14s - loss: 0.0167 - accuracy: 0.9956 - val_loss: 0.0162 - val_accuracy: 0.9958
Epoch 9/50
2042/2042 - 14s - loss: 0.0180 - accuracy: 0.9952 - val_loss: 0.0161 - val_accuracy: 0.9958
Epoch 10/50
2042/2042 - 14s - loss: 0.0170 - accuracy: 0.9954 - val_loss: 0.0161 - val_accuracy: 0.9958
Epoch 11/50
2042/2042 - 15s - loss: 0.0168 - accuracy: 0.9954 - val_loss: 0.0162 - val_accuracy: 0.9958
Epoch 12/50
2042/2042 - 16s - loss: 0.0168 - accuracy: 0.9954 - val_loss: 0.0162 - val_accuracy: 0.9958
Epoch 13/50
2042/2042 - 17s - loss: 0.0166 - accuracy: 0.9955 - val_loss: 0.0162 - val_accuracy: 0.9958
Epoch 14/50
2042/2042 - 14s - loss: 0.0163 - accuracy: 0.9956 - val_loss: 0.0162 - val_accuracy: 0.9958
Epoch 15/50
```

(b) CNN Model Fit

```
[44]: y_pred = model.predict(x_test, batch_size=10,
                           verbose=0)

score = model.evaluate(x_test, y_test,verbose=0)

print(score)
```

```
[0.013178911991417408, 0.9968253970146179]
```

(c) CNN Model Predict

Figure 21: Deep Learning Model Code Snippet Exp-4

7 Evaluation Output

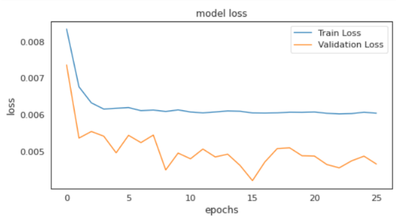
Finally, the models are evaluated using training and testing accuracy, RMSE and MAE values from **sklearn.metrics** and confusion matrix from **scikitplot**. Also, the model's training and validation loss against the number of epochs is visualized to check whether the model is a overfit or underfit or perfect fit.

7.1 Experiment-1 Evaluation Code Snippet

```
[130]: # make a prediction
yhat = model.predict(testX)
testX = testX.reshape((testX.shape[0], testX.shape[2]))

[131]: from numpy import concatenate
# invert scaling for forecast
inv_yhat = concatenate((yhat, testX[:, 0:]), axis=1)
#inv_yhat = concatenate((yhat[:,:], testX[:,0,0:]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]

[132]: # invert scaling for actual
testY = testY.reshape((len(testY), 1))

[133]: inv_y = concatenate((testY, testX[:, 0:]), axis=1)
#inv_y = concatenate((testY, testX[:, 0, 0:]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:,0]

[134]: from math import sqrt
from sklearn.metrics import mean_squared_error, mean_absolute_error
# calculate RMSE
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)

mae = mean_absolute_error(inv_y, inv_yhat)
print('Test MAE: %.3f' % mae)

Test RMSE: 0.118
Test MAE: 0.021
```

(a) Evaluation Code

```
[135]: plt.figure(figsize=(8,4))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epochs')
plt.legend(loc='upper right')
plt.show();
```



(b) Loss Vs Number of Epoch Code

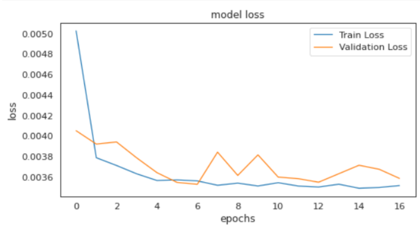
Figure 22: Evaluation Code Snippet Exp-1

7.2 Experiment-2 Evaluation Code Snippet

```
[53]: # make a prediction
yhat = model.predict(testX)

[54]: testX = testX.reshape((testX.shape[0], testX.shape[2]))

[55]: from numpy import concatenate
# invert scaling for forecast
#inv_yhat = concatenate((yhat[:,0,:], testX[:, 0:]), axis=1)
inv_yhat = concatenate((yhat, testX[:, 0:]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]

[56]: # invert scaling for actual
testY = testY.reshape((len(testY), 1))

[57]: inv_y = concatenate((testY, testX[:, 0:]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:,0]

[59]: from math import sqrt
from sklearn.metrics import mean_squared_error, mean_absolute_error
# calculate RMSE
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)

mae = mean_absolute_error(inv_y, inv_yhat)
print('Test MAE: %.3f' % mae)

Test RMSE: 0.082
Test MAE: 0.013
```

(a) Evaluation Code

```
[60]: plt.figure(figsize=(8,4))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epochs')
plt.legend(loc='upper right')
plt.show();
```

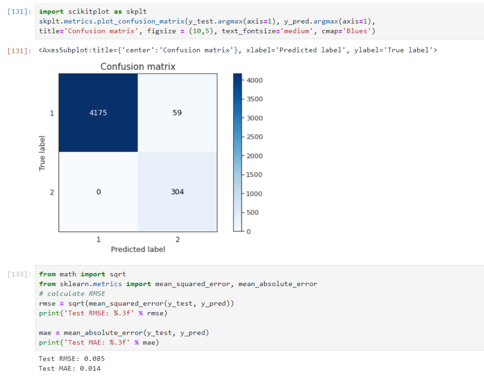


(b) Loss Vs Number of Epoch Code

Figure 23: Evaluation Code Snippet Exp-2

7.3 Experiment-3 Evaluation Code Snippet

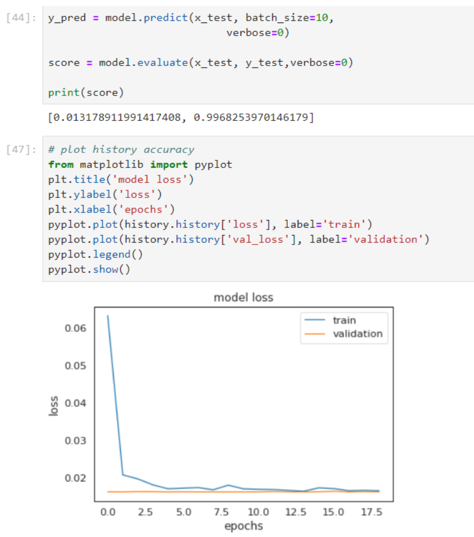
(a) Loss Vs Number of Epoch Code



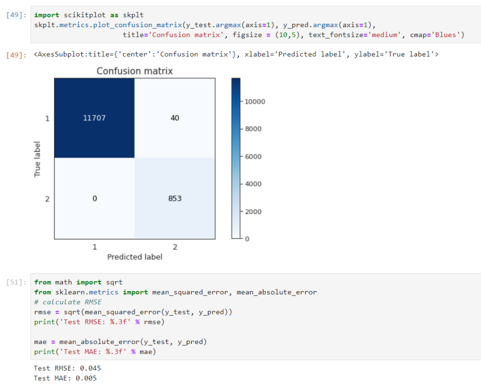
(b) Confusion Matrix Code

Figure 24: Evaluation Code Snippet Exp-3

7.4 Experiment-4 Evaluation Code Snippet



(a) Loss Vs Number of Epoch Code



(b) Confusion Matrix Code

Figure 25: Evaluation Code Snippet Exp-4

References

Data Engineering with Google Cloud Professional Certificate (n.d.). `https://www.coursera.org/professional-certificates/gcp-data-engineering#courses`. Accessed: 2020-07-15.

Google Cloud Platform (GCP) documentation (n.d.). `https://cloud.google.com/docs`. Accessed: 2020-08-10.

Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization (n.d.). `https://www.coursera.org/learn/deep-neural-network/home/welcome`. Accessed: 2020-07-29.