Configuration Manual

MSc Research Project
Data Analytics

Nihad Kazi
Student ID:22166009

School of Computing
National College of Ireland

Supervisor: Abdul Qayum

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Nihad Kazi
Student ID:	22166009
Programme:	Data Analytics
Year:	2023
Module:	Research Project
Supervisor:	Abdul Qayum
Submission Due Date:	31-01-2024
Project Title:	Enhancing Traffic Flow Prediction using Real Time Data with deep learning Techniques
Word Count:	625
Page Count:	10

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:	Nihad Kazi
Date:	31st January 2024

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

Nihad Kazi
Student ID: 22166009

1 Introduction

This manual illustrates how to execute and configure implementation code for this research project. This manual highlights technical study of traffic prediction using deep learning models such as GRU and LSTM with all libraries which were used for implementation. The aim is to make a user friendly manual which explains everything from start to end.

2 System Specification

2.1 System Specification

Following are the hardware specification of the system that was used to develop the project:

Component	**Specification**
Processor	12th Gen Intel(R) Core(TM) i5-1235U
RAM	16GB
Storage	512GB
Operating System	Windows 11

Table 1: System Specifications

2.2 Software Specification

The specifications of the software utilized for the system were as follows:

Software	**Specifications**
Operating System	Windows 11 (64 bit)
IDE	Jupyter Notebook
Scripting Language	Python 3.7

Table 2: Software Specifications

3 Tools Used

This section contains list of tools used to implement the project.

3.1 Programming Language: Python

To construct the project, the Python programming language was utilized. Python was selected mostly because of its practical packages for deep learning models, dataset preparation, and visualization. One downloaded Python from [1].
The official Python website's download page is displayed in 1.

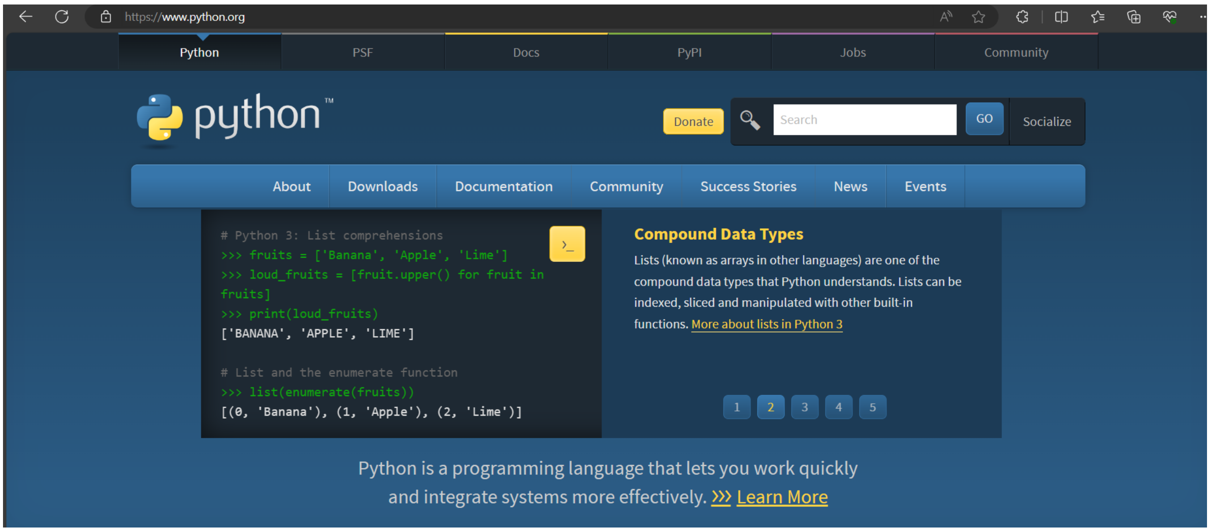


Figure 1: Python Website

3.2 IDE: Jupyter Notebooks

Jupyter Notebooks was used as an IDE to develop the research project. Due to the cell wise execution functionality, it allows to user to check the output of each cell with ease. Jupyter notebook from Anaconda Navigator software hub was used. Figure 2 shown below show Jupyter Notebooks in Anaconda Navigator. Click the Launch Action to launch Jupyter Notebook in Anaconda Navigator.

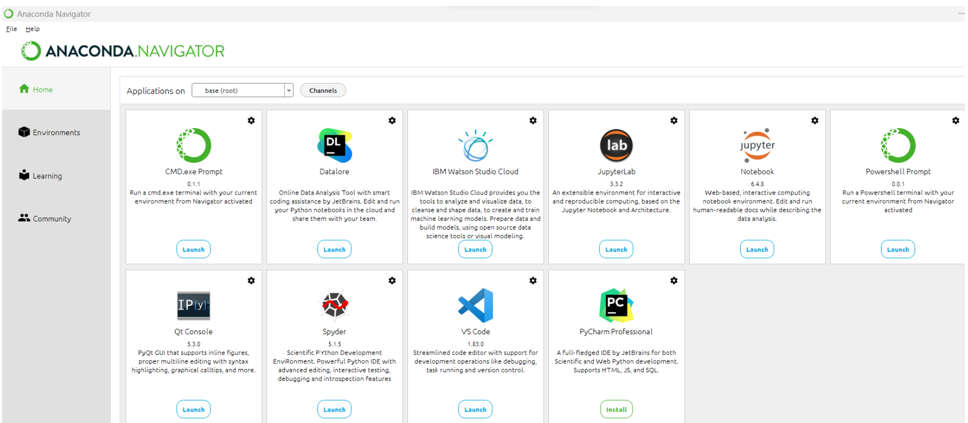


Figure 2: Jupyter Notebook in Anaconda Navigator

[1] https://www.python.org/downloads/

4 Implementation

4.1 Dataset

The Dataset is stored in a folder which is pasted on the desktop. The dataset names are Multiple_Region_Data.csv and MultipleSensorsData.csv. Folder path is mentioned as C://Users//Nihad Kazi//Desktop//Research Datasets. Replace the users name with the system name and then run the file in jupyter notebooks.

4.2 Libraries Used for Implementation

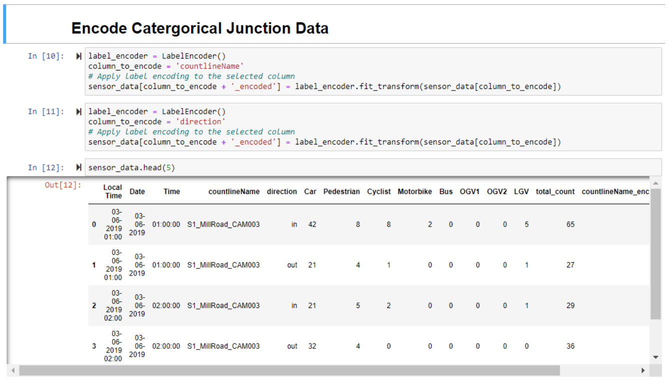
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
import tensorflow
from statsmodels.tsa.stattools import adfuller
from sklearn.preprocessing import MinMaxScaler
from tensorflow import keras
from keras import callbacks
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D, Flatten, Dense, LSTM, Dropout, GRU, Bidirectional
from tensorflow.keras.optimizers import SGD, Adam, RMSprop
import math
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder
import warnings
from sklearn.metrics import mean_squared_error, r2_score,mean_absolute_error
import xgboost as xgb
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from keras.callbacks import EarlyStopping
warnings.filterwarnings("ignore")
```

Figure 3: Libraries Imported

4.3 Data Preprocessing

In data pre processing we have checked for null and collinearity. None were found for sensor data but regional data had both null values and collinear data. Columns will null values and high collinearity were dropped as a part of treatment. Both the datasets were splitted into junctions and regions and normalized and differenced to make it stationary as a part of treatment. Implementation shown in the figures below are of sensor data as both have same implementation.

We have also implemented Dickey fuller test to check the stationarity of the data. Figure 4.3 shows implementation of Dickey Fuller Test.



Encode Catergorical Junction Data

In [10]:
```
label_encoder = LabelEncoder()
column_to_encode = 'countlineName'
# Apply label encoding to the selected column
sensor_data[column_to_encode + '_encoded'] = label_encoder.fit_transform(sensor_data[column_to_encode])
```

In [11]:
```
label_encoder = LabelEncoder()
column_to_encode = 'direction'
# Apply label encoding to the selected column
sensor_data[column_to_encode + '_encoded'] = label_encoder.fit_transform(sensor_data[column_to_encode])
```

In [12]:
```
sensor_data.head(5)
```

Out[12]:

	Local Time	Date	Time	countlineName	direction	Car	Pedestrian	Cyclist	Motorbike	Bus	OGV1	OGV2	LGV	total_count	countlineName_enc
0	03-06-2019 01:00	03-06-2019	01:00:00	S1_MillRoad_CAM003	in	42	8	8	2	0	0	0	5	65	
1	03-06-2019 01:00	03-06-2019	01:00:00	S1_MillRoad_CAM003	out	21	4	1	0	0	0	0	1	27	
2	03-06-2019 02:00	03-06-2019	02:00:00	S1_MillRoad_CAM003	in	21	5	2	0	0	0	0	1	29	
3	03-06-2019 02:00	03-06-2019	02:00:00	S1_MillRoad_CAM003	out	32	4	0	0	0	0	0	0	36	

Figure 4: Label Encoding Categorical Sensor Data

```
def Normalize(dataframe,column):
    average = dataframe[column].mean()
    stdev = dataframe[column].std()
    df_normalized = (dataframe[column] - average) / stdev
    df_normalized = df_normalized.to_frame()
    return df_normalized, average, stdev

# Differencing Function
def Difference(dataframe,column, interval):
    diff = []
    for i in range(interval, len(dataframe)):
        value = dataframe[column][i] - dataframe[column][i - interval]
        diff.append(value)
    return diff
```

Figure 5: Differencing and Normalizing Function

5 Design Specification

5.1 Split the data for modelling:

This section covers splitting the data into different frames. Reference Figure 5.1

5.2 Train GRU model for Sensor and Regional Data:

This section covers splitting the data into different frames. Reference Figure 5.3

5.3 Train LSTM model for Sensor and Regional Data:

This section covers splitting the data into different frames. Reference Figure 5.3

6 Results

Results is divided in 4 parts namely -

- Result for GRU with Sensor Data 6
- Result for LSTM with Sensor Data 6

Dickey Fuller Test to check Stationarity

```python
def Stationary_check(dataframe):
    check = adfuller(dataframe.dropna())
    print(f"ADF Statistic: {check[0]}")
    print(f"p-value: {check[1]}")
    print("Critical Values:")
    for key, value in check[4].items():
        print('\t%s: %.3f' % (key, value))
    if check[0] > check[4]["1%"]:
        print("Time Series is Non-Stationary")
    else:
        print("Time Series is Stationary")


List_df_ND = [ dataframe_N1["Diff"], dataframe_N2["Diff"], dataframe_N3["Diff"], dataframe_N4["Diff"], dataframe_N5["Diff"],
print("Checking the transformed series for stationarity:")
for i in List_df_ND:
    print("\n")
    Stationary_check(i)
```

Figure 6: Dickey Fuller Test Implementation Function

- Result for GRU with Regional Data 6
- Result for LSTM with Regional Data 6

```python
# Splitting the dataset
def Split_data(dataframe):
    training_size = int(len(dataframe)*0.90)
    data_len = len(dataframe)
    train, test = dataframe[0:training_size],dataframe[training_size:data_len]
    train, test = train.values.reshape(-1, 1), test.values.reshape(-1, 1)
    return train, test
# Splitting the training and test datasets
Region1_train, Region1_test = Split_data(dataframe_J1)
Region2_train, Region2_test = Split_data(dataframe_J2)
Region3_train, Region3_test = Split_data(dataframe_J3)
Region4_train, Region4_test = Split_data(dataframe_J4)
Region5_train, Region5_test = Split_data(dataframe_J5)
Region6_train, Region6_test = Split_data(dataframe_J6)


# Target and Feature
def target_and_feature(dataframe):
    end_len = len(dataframe)
    X = []
    y = []
    steps = 32
    for i in range(steps, end_len):
        X.append(dataframe[i - steps:i, 0])
        y.append(dataframe[i, 0])
    X, y = np.array(X), np.array(y)
    return X ,y

# fixing the shape of X_test and X_train
def FeatureFixShape(train, test):
    train = np.reshape(train, (train.shape[0], train.shape[1], 1))
    test = np.reshape(test, (test.shape[0],test.shape[1],1))
    return train, test

# Assigning features and target
X_train_Region1, y_train_Region1 = target_and_feature(Region1_train)
X_test_Region1, y_test_Region1 = target_and_feature(Region1_test)
X_train_Region1, X_test_Region1 = FeatureFixShape(X_train_Region1, X_test_Region1)

X_train_Region2, y_train_Region2 = target_and_feature(Region2_train)
X_test_Region2, y_test_Region2 = target_and_feature(Region2_test)
X_train_Region2, X_test_Region2 = FeatureFixShape(X_train_Region2, X_test_Region2)

X_train_Region3, y_train_Region3 = target_and_feature(Region3_train)
X_test_Region3, y_test_Region3 = target_and_feature(Region3_test)
X_train_Region3, X_test_Region3 = FeatureFixShape(X_train_Region3, X_test_Region3)

X_train_Region4, y_train_Region4 = target_and_feature(Region4_train)
X_test_Region4, y_test_Region4 = target_and_feature(Region4_test)
X_train_Region4, X_test_Region4 = FeatureFixShape(X_train_Region4, X_test_Region4)

X_train_Region5, y_train_Region5 = target_and_feature(Region5_train)
X_test_Region5, y_test_Region5 = target_and_feature(Region5_test)
X_train_Region5, X_test_Region5 = FeatureFixShape(X_train_Region5, X_test_Region5)

X_train_Region6, y_train_Region6 = target_and_feature(Region6_train)
X_test_Region6, y_test_Region6 = target_and_feature(Region6_test)
X_train_Region6, X_test_Region6 = FeatureFixShape(X_train_Region6, X_test_Region6)
```

Figure 7: Data Split

```
GRU_model(X_Train, y_Train, X_Test):

early_stopping = EarlyStopping(min_delta=0.001, patience=10, restore_best_weights=True)

model = Sequential()
model.add(GRU(units=100, return_sequences=True, input_shape=(X_Train.shape[1],1), activation='tanh'))
model.add(Dropout(0.2))
model.add(GRU(units=50, return_sequences=True, activation='tanh'))
model.add(Dropout(0.2))
model.add(GRU(units=50, activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(units=1))

optimizer = RMSprop(learning_rate=0.001)

model.compile(optimizer=optimizer, loss='mean_squared_error')
history = model.fit(X_Train, y_Train, epochs=50, batch_size=32, callbacks=[early_stopping], validation_split=0.2, verbose=

pred_GRU = model.predict(X_Test)

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title('Loss over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.title('Training Loss Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

return pred_GRU

determine the root mean squared prediction error
RMSE_Value(test,predicted):
rmse = math.sqrt(mean_squared_error(test, predicted))
print("The root mean squared error is {}.".format(rmse))
return rmse

calculate_r2(test, predicted):
r2 = r2_score(test, predicted)
print("The R-squared (R2) score is {:.2f}.".format(r2))
return r2

calculate_mae(test, predicted):
# Using sklearn's mean_absolute_error function to calculate MAE
mae = mean_absolute_error(test, predicted)
print(f"The Mean Absolute Error (MAE) is: {mae:.2f}.")
return mae


lotting the goal and forecast comparison plot
PredictionsPlot(test,predicted,m):
plt.figure(figsize=(12,5),facecolor="#627D78")
plt.plot(test, color=colors[m],label="True Value",alpha=0.5 )
plt.plot(predicted, color="#627D78",label="Predicted Values")
plt.title("GRU Traffic Prediction Vs True values")
plt.xlabel("DateTime")
plt.ylabel("Number of Vehicles")
plt.legend()
```

Figure 8: GRU Implementation

Inversion and Plotting Predicted vs Original Values for GRU model

```
def inverse_difference(last_ob, value):
    inversed = value + last_ob
    return inversed

#Plotting the comparison
def Sub_Plots2(df_1, df_2,title,m):
    fig, axes = plt.subplots(1, 2, figsize=(18,4), sharey=True,facecolor="#627D78")
    fig.suptitle(title)

    pl_1=sns.lineplot(ax=axes[0],data=df_1,color=colors[m])
    axes[0].set(ylabel ="Prediction")

    pl_2=sns.lineplot(ax=axes[1],data=df_2["total_count"],color="#627D78")
    axes[1].set(ylabel ="Orignal")
```

Predicting, Plotting and Evaluating GRU Model for Junction 1

```
PredJ1 = GRU_model(X_train_Region1,y_train_Region1,X_test_Region1)
```

```
Epoch 1/50
127/127 [==============================] - 31s 191ms/step - loss: 0.1929 - val_loss: 0.1251
Epoch 2/50
127/127 [==============================] - 22s 170ms/step - loss: 0.1396 - val_loss: 0.1217
Epoch 3/50
127/127 [==============================] - 25s 193ms/step - loss: 0.1391 - val_loss: 0.1215
Epoch 4/50
127/127 [==============================] - 24s 192ms/step - loss: 0.1375 - val_loss: 0.1266
Epoch 5/50
127/127 [==============================] - 24s 190ms/step - loss: 0.1377 - val_loss: 0.1358
Epoch 6/50
127/127 [==============================] - 25s 194ms/step - loss: 0.1367 - val_loss: 0.1318
Epoch 7/50
127/127 [==============================] - 24s 192ms/step - loss: 0.1399 - val_loss: 0.1196
Epoch 8/50
127/127 [==============================] - 24s 192ms/step - loss: 0.1358 - val_loss: 0.1244
Epoch 9/50
127/127 [==============================] - 25s 195ms/step - loss: 0.1372 - val_loss: 0.1268
Epoch 10/50
```

Figure 9: Inversion and Prediction

```python
def LSTM_model(X_Train, y_Train, X_Test):
    early_stopping = EarlyStopping(min_delta=0.001, patience=10, restore_best_weights=True)

    model = Sequential()
    model.add(LSTM(units=128, return_sequences=True, input_shape=(X_Train.shape[1], 1), activation='tanh'))
    model.add(Dropout(0.2))
    model.add(LSTM(units=64, return_sequences=True, activation='tanh'))
    model.add(Dropout(0.2))
    model.add(LSTM(units=32, return_sequences=True, activation='tanh'))
    model.add(Dropout(0.2))
    model.add(LSTM(units=32, return_sequences=True, activation='tanh'))
    model.add(Dropout(0.2))
    model.add(LSTM(units=32, activation='tanh'))
    model.add(Dropout(0.2))
    model.add(Dense(units=1))

    optimizer = Adam(learning_rate=0.001)  # Adam optimizer with a lower learning rate

    # Compiling the model
    model.compile(optimizer=optimizer, loss='mean_squared_error')
    history = model.fit(X_Train, y_Train, epochs=50, batch_size=32, callbacks=[early_stopping], validation_split=0.2)

    pred_LSTM = model.predict(X_Test)

    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.legend()
    plt.title('Loss over Epochs')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')

    # Plot the training loss over epochs
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.title('Training Loss Over Epochs')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()

    return pred_LSTM


def RMSE_Value(test,predicted):
    rmse = math.sqrt(mean_squared_error(test, predicted))
    print("The root mean squared error is {}.".format(rmse))
    return rmse

def calculate_r2(test, predicted):
    r2 = r2_score(test, predicted)
    print("The R-squared (R2) score is {:.2f}.".format(r2))
    return r2

def calculate_mae(test, predicted):
    # Using sklearn's mean_absolute_error function to calculate MAE
    mae = mean_absolute_error(test, predicted)
    print(f"The Mean Absolute Error (MAE) is: {mae:.2f}.")
    return mae

# Plotting the goal and forecast comparison plot
def PredictionsPlot(test,predicted,m):
    plt.figure(figsize=(12,5),facecolor="#627D78")
    plt.plot(test, color=colors[m],label="True Value",alpha=0.5 )
    plt.plot(predicted, color="#627D78",label="Predicted Values")
    plt.title("GRU Traffic Prediction Vs True values")
    plt.xlabel("DateTime")
    plt.ylabel("Number of Vehicles")
    plt.legend()
    plt.show()
```

Figure 10: LSTM Implementation

Predicting, Plotting and Evaluating LSTM Model for Junction 1

```
]: PredJ1_LSTM = LSTM_model(X_train_Region1,y_train_Region1,X_test_Region1)

Epoch 1/50
127/127 [==============================] - 74s 416ms/step - loss: 0.3536 - val_loss: 0.3692
Epoch 2/50
127/127 [==============================] - 47s 371ms/step - loss: 0.3340 - val_loss: 0.3586
Epoch 3/50
127/127 [==============================] - 47s 369ms/step - loss: 0.2907 - val_loss: 0.1952
Epoch 4/50
127/127 [==============================] - 45s 356ms/step - loss: 0.1773 - val_loss: 0.1412
Epoch 5/50
127/127 [==============================] - 46s 364ms/step - loss: 0.1548 - val_loss: 0.1275
Epoch 6/50
127/127 [==============================] - 47s 370ms/step - loss: 0.1454 - val_loss: 0.1243
Epoch 7/50
127/127 [==============================] - 48s 377ms/step - loss: 0.1455 - val_loss: 0.1235
Epoch 8/50
127/127 [==============================] - 46s 364ms/step - loss: 0.1384 - val_loss: 0.1184
Epoch 9/50
127/127 [==============================] - 46s 360ms/step - loss: 0.1393 - val_loss: 0.1185
Epoch 10/50
```

Figure 11: Prediction LSTM

	Junction	RMSE	R-Squared	MAE
0	Junction1	0.306659	0.739601	0.225379
1	Junction2	0.267081	0.705212	0.184387
2	Junction3	0.261878	0.794659	0.191852
3	Junction4	0.229130	0.890899	0.168262
4	Junction5	0.274593	0.674920	0.187274
5	Junction6	0.271298	0.717538	0.173206

Figure 12: Result for GRU with Sensor Data

	Junction	RMSE	R-Squared	MAE
0	Junction1	0.285303	0.774607	0.210253
1	Junction2	0.218244	0.803162	0.158405
2	Junction3	0.221888	0.852584	0.164549
3	Junction4	0.226033	0.893829	0.163932
4	Junction5	0.252805	0.724461	0.172743
5	Junction6	0.190241	0.861108	0.126074

Figure 13: Result for LSTM with Sensor Data

	Region	RMSE	R-Squared	MAE
0	Region1	1.002521	0.401282	0.682273
1	Region2	0.981422	0.363964	0.629490
2	Region3	0.999625	0.404736	0.653231

Figure 14: Result for GRU with Regional Data

	Junction	RMSE	R-Squared	MAE
0	Region1	1.001367	0.402660	0.676723
1	Region2	0.974499	0.372905	0.634605
2	Region3	1.006856	0.396093	0.648143

Figure 15: Result for LSTM with Regional Data