Configuration Manual

MSc Research Project

Data Analytics

Caroline Vincent
Student ID: x22153926

School of Computing
National College of Ireland

Supervisor: Mr. Taimur Hafeez

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Caroline Vincent		
Student ID:	22153926		
Programme:	MSc. Data Analytics	**Year:**	2023
Module:	Research in Computing		
Supervisor:	Mr. Taimur Hafeez		
Submission Due Date:	14/12/2023		
Project Title:	Enhancing Urban Traffic Flow Management and Analysis through Deep learning Techniques		
Word Count:	1420	**Page Count**	21

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:	Caroline Vincent
Date:	14/12/2023

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

Caroline Vincent
x22153926

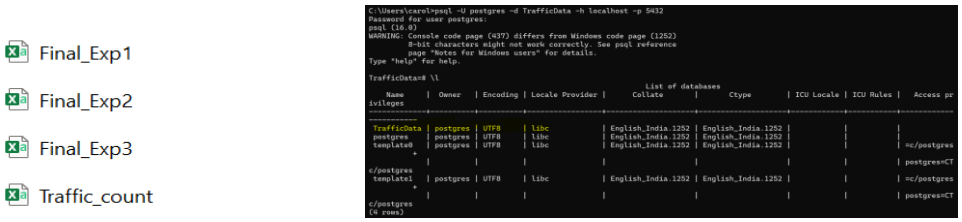
1 Introduction

This configuration manual directs the user to replicate the research project “Enhancing Urban Traffic Flow Management and Analysis through Deep learning Techniques”. The manual explains the storage, databases, hardware and software requirements, programming languages, and system setup used in the implementation of the research.

2 System Configuration

2.1 Storage

Research Project’s data storage layer consists of CSV files as shown in **Fig** and a PostgreSQL database. For initial data entry and intermediate data storage, CSV files provide an adaptable and user-friendly format. An advanced open-source database system, PostgreSQL, ensures data integrity and supports sophisticated queries and transaction for more reliable and secure data management.



a) CSV storage

b) PostgreSQL storage

Figure 1: Storage Specifications

2.2 Hardware

The research project is developed on a computer with the following specifications:

Device specifications

Device name	DESKTOP-SCE4OID
Processor	12th Gen Intel(R) Core(TM) i5-1235U 1.30 GHz
Installed RAM	16.0 GB (15.7 GB usable)
Device ID	27BDD030-416B-4A59-81BA-9C014A189224
Product ID	00342-42631-48716-AAOEM
System type	64-bit operating system, x64-based processor

Figure 2: Hardware Specifications

These specifications offer an energy-efficient performance balance that is appropriate for data processing and analytical applications requiring compute.

2.3 Software

The software components utilized in this project are:

2.3.1 Operating System:

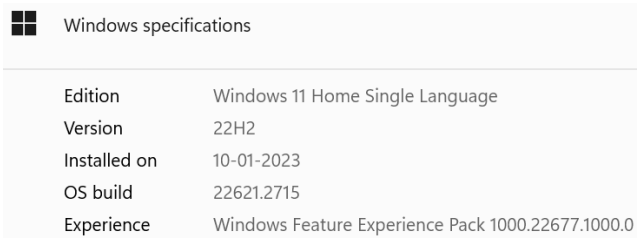


Figure 3: Software Specifications

2.3.2 Development Environment

Python programming language is combined with Jupyter Notebook, also a range of python tools, including scikit-learn for deep leaning algorithms and evaluation metrics, pandas for data manipulation, TensorFlow for building deep learning models.

a) Python version

b) jupyter files

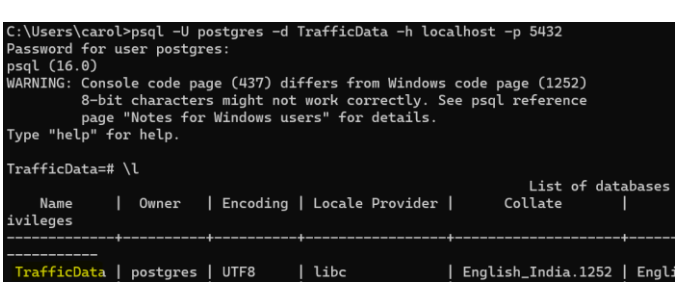
Figure 4: Development Specifications

2.4 Database

For this project, PostgreSQL is the preferred database management tool.

Command Prompt: To directly run SQL commands and scripts in command-line interface.
pgAdmin 4: A web-based platform which offers a graphical user interface for database development and administration tasks.

In order to facilitate efficient data retrieval for project's analytics and visualization layers, the database schema is created.



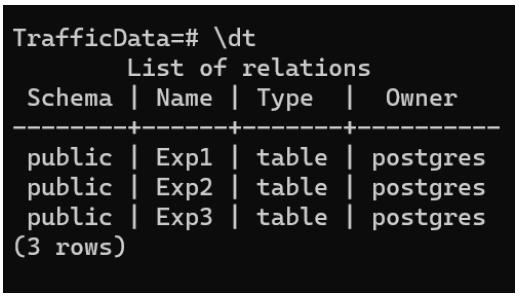
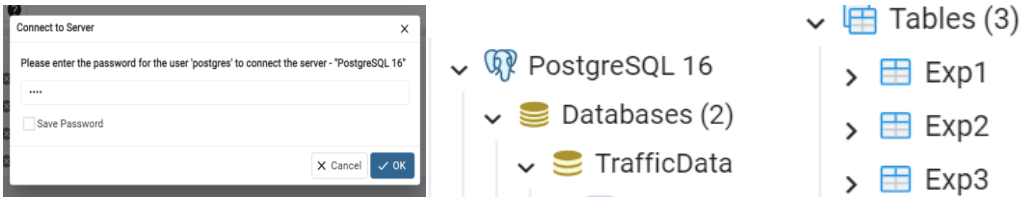


Figure 5: Command Prompt DB



a) Connecting pgadmin 4 server b) TrafficData database c) Tables created

Figure 6: pgAdmin 4

3 Data Preparation and Pre-Processing

The dataset used for the research is ethically sourced from the UK government website[1]. The data from the year 2000 is available, for this study, data ranging from 2017 to 2021 comprising 1,099,968 records. This dataset makes it possible to derive insightful information from raw traffic numbers.

Experiment 1 - Traffic Prediction Based on Vehicles

Predicting traffic volume based on vehicles.

1. Data Fetching

In [35]:
```
import pandas as pd

# Loading dataset
dataset1 = pd.read_csv('Traffic_count.csv')

# Filtering rows
years = [2017, 2018, 2019, 2020, 2021]
vehdata = dataset1[dataset1['Year'].isin(years)].copy()

# Sorting in ascending order
vehdata.sort_values(by='Year', ascending=True, inplace=True)

# Saving the filtered data to new dataset
vehdata.to_csv('Trafficdataset.csv')
```

In [36]:
```
# printing the columns
print(vehdata.columns)
```

```
Index(['Count_point_id', 'Direction_of_travel', 'Year', 'Count_date', 'hour',
       'Region_id', 'Region_name', 'Region_ons_code', 'Local_authority_id',
       'Local_authority_name', 'Local_authority_code', 'Road_name',
       'Road_category', 'Road_type', 'Start_junction_road_name',
       'End_junction_road_name', 'Easting', 'Northing', 'Latitude',
       'Longitude', 'Link_length_km', 'Link_length_miles', 'Pedal_cycles',
       'Two_wheeled_motor_vehicles', 'Cars_and_taxis', 'Buses_and_coaches',
       'LGVs', 'HGVs_2_rigid_axle', 'HGVs_3_rigid_axle',
       'HGVs_4_or_more_rigid_axle', 'HGVs_3_or_4_articulated_axle',
       'HGVs_5_articulated_axle', 'HGVs_6_articulated_axle', 'All_HGVs',
       'All_motor_vehicles'],
      dtype='object')
```

a) Data Preparation for Experiment 1

Experiment 2 - Traffic Prediction Based on Hour of the day

Predicting traffic volume based on hour of the day.

1. Data Fetching

In [6]:
```
import pandas as pd

# Loading dataset
dataset1 = pd.read_csv('Traffic_count.csv')

# Filtering rows
years = [2017, 2018, 2019, 2020, 2021]
hourdata = dataset1[dataset1['Year'].isin(years)].copy()

# Sorting in ascending order
hourdata.sort_values(by='Year', ascending=True, inplace=True)

# Saving the filtered data to new dataset
hourdata.to_csv('Trafficdataset1.csv')
```

In [4]:
```
# printing columns
print(hourdata.columns)
```

```
Index(['Count_point_id', 'Direction_of_travel', 'Year', 'Count_date', 'hour',
       'Region_id', 'Region_name', 'Region_ons_code', 'Local_authority_id',
       'Local_authority_name', 'Local_authority_code', 'Road_name',
       'Road_category', 'Road_type', 'Start_junction_road_name',
       'End_junction_road_name', 'Easting', 'Northing', 'Latitude',
       'Longitude', 'Link_length_km', 'Link_length_miles', 'Pedal_cycles',
       'Two_wheeled_motor_vehicles', 'Cars_and_taxis', 'Buses_and_coaches',
       'LGVs', 'HGVs_2_rigid_axle', 'HGVs_3_rigid_axle',
       'HGVs_4_or_more_rigid_axle', 'HGVs_3_or_4_articulated_axle',
       'HGVs_5_articulated_axle', 'HGVs_6_articulated_axle', 'All_HGVs',
       'All_motor_vehicles'],
      dtype='object')
```

b) Data Preparation for Experiment 2

Experiment 3 - Traffic Prediction Based on Geospatial.

Predicting traffic volume based influenced by their geographical locations and distance from Trafalgar Square.

1. Data Fetching

In [3]:
```
import pandas as pd

# Loading dataset
dataset1 = pd.read_csv('Traffic_count.csv')

# Filtering rows
years = [2017, 2018, 2019, 2020, 2021]
spatialdata = dataset1[dataset1['Year'].isin(years)].copy()

# Sorting in ascending order
spatialdata.sort_values(by='Year', ascending=True, inplace=True)

# Saving the filtered data to new dataset
spatialdata.to_csv('Trafficdataset2.csv')
```

In [4]:
```
# printing columns
print(spatialdata.columns)
```

```
Index(['Count_point_id', 'Direction_of_travel', 'Year', 'Count_date', 'hour',
       'Region_id', 'Region_name', 'Region_ons_code', 'Local_authority_id',
       'Local_authority_name', 'Local_authority_code', 'Road_name',
       'Road_category', 'Road_type', 'Start_junction_road_name',
       'End_junction_road_name', 'Easting', 'Northing', 'Latitude',
       'Longitude', 'Link_length_km', 'Link_length_miles', 'Pedal_cycles',
       'Two_wheeled_motor_vehicles', 'Cars_and_taxis', 'Buses_and_coaches',
       'LGVs', 'HGVs_2_rigid_axle', 'HGVs_3_rigid_axle',
       'HGVs_4_or_more_rigid_axle', 'HGVs_3_or_4_articulated_axle',
       'HGVs_5_articulated_axle', 'HGVs_6_articulated_axle', 'All_HGVs',
       'All_motor_vehicles'],
      dtype='object')
```

c) Data Preparation for Experiment 3

Figure 7: Data Preparation

[1] https://www.data.gov.uk/dataset/208c0e7b-353f-4e2d-8b7a-1a7118467acc/gb-road-traffic-counts

3.1 Data Preparation, Cleaning and Transformation

The foundational phase of study entails considerable data preparation, cleaning, and transformation. In Experiment 1, using Pandas in Python for data cleaning (such as removing null values and irrelevant columns) and transformation (such as converting 'Count_date' to determine format and aggregating vehicle counts). In Experiment 2, null values are removed, and median values are imputed to missing data. Hourly column classified into peak and non-peak periods. In order to investigate the impact of geography on traffic volume, experiment 3 involved extensive data cleaning, median imputation for missing values, and the creation of new variables such as the distance to central London. The preparation, cleaning, and transformation of the data for three experiments are mentioned in the code artefacts **.ipynb** file.

Figure 8: Data after cleaning & transformation for Experiment 1, 2 &3

4 Exploratory Data Analysis

The data stored in PostgreSQL is read through create engine of sqlalchemy and Psycopg2 package of python as shown in Figure 9 and stored in pandas' data frame.

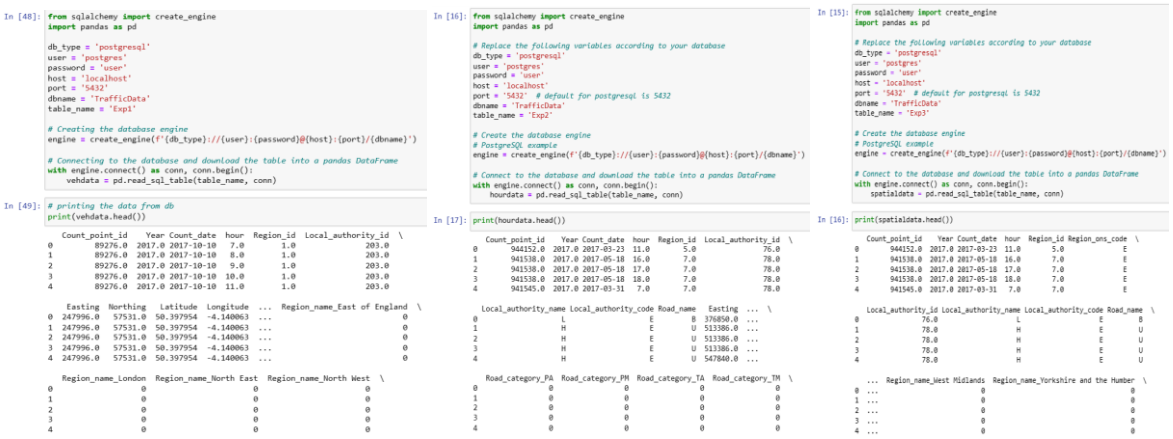
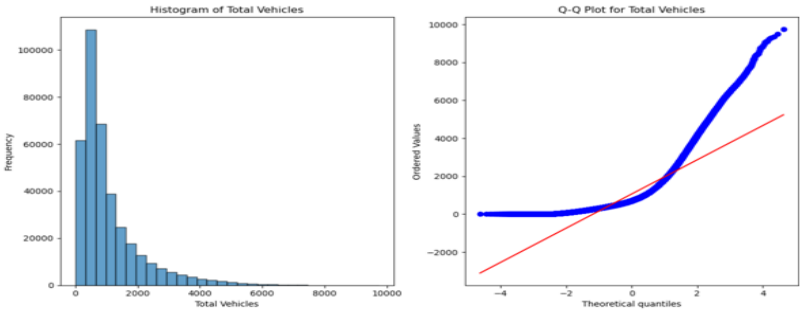


Figure 9: Data from DB stored in pandas' data frame for Experiment 1, 2 & 3

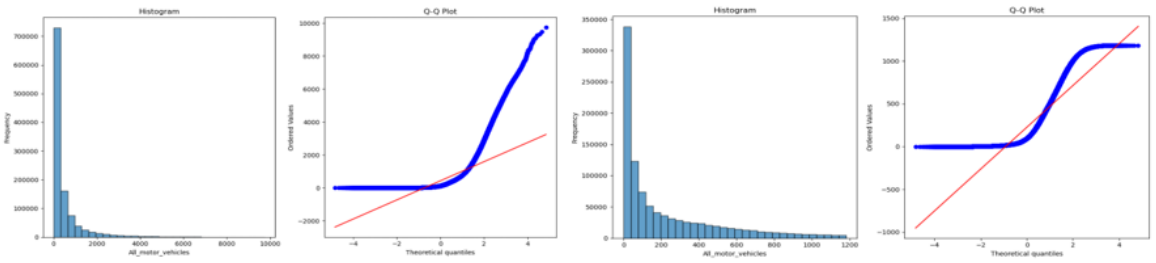
4.1 Statistical Analysis

Figure 10, Figure 11, and Figure 12 represents the code for skewness and kurtosis before and after outlier removal.




```
Skewness for 'total_vehicles' column: 2.15774557302573
Kurtosis for 'total_vehicles' column: 5.661693913006115
```

Figure 10 : Statistical Analysis for Experiment 1



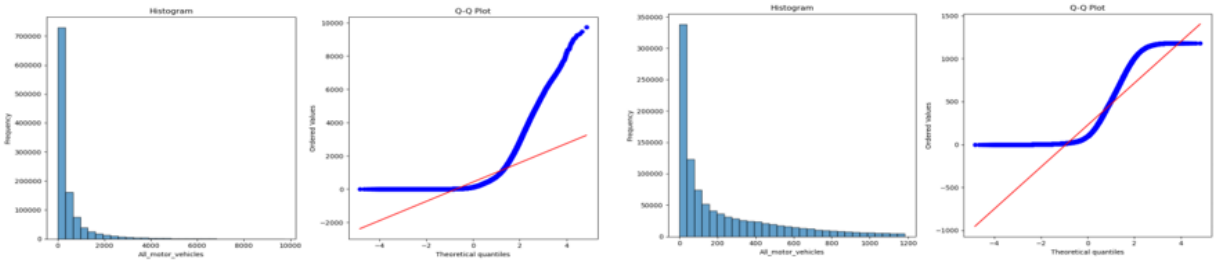
```
Skewness for 'All_motor_vehicles' column: 3.4800262525591337
Kurtosis for 'All_motor_vehicles' column: 15.56151524337654
```

a) Before Outlier removal

```
Skewness for 'All_motor_vehicles' column: 1.4839501562874622
Kurtosis for 'All_motor_vehicles' column: 1.4088428811460156
```

b) After outlier removal

Figure 11 : Statistical Analysis for Experiment 2



a) Before Outlier removal

b) After outlier removal

Figure 12 : Statistical Analysis for Experiment 3

5 Feature Selection

After data stationarity check using dickey-fuller test, MinMaxScaler normalization is applied from sklearn.preprocessing to normalize the data. Using train test split of sklearn.model, the data is split into train and test set. The normalization, train-test split and reshape feature in the analysis are displayed in the code below.

5.1 Feature Selection and Data Splitting – Experiment 1

Traffic flow prediction based on the influence of vehicle shown in Figure 13.

```
# Feature Selection based on Correlation Analysis
selected_features = [
    'Cars_and_taxis', 'LGVs', 'HGVs_2_rigid_axle', 'All_HGVs',
    'HGVs_3_rigid_axle', 'HGVs_6_articulated_axle', 'HGVs_3_or_4_articulated_axle',
    'HGVs_4_or_more_rigid_axle', 'HGVs_5_articulated_axle',
    'Two_wheeled_motor_vehicles', 'Buses_and_coaches', 'Year', 'hour', 'Pedal_cycles'
]

# Preparing the dataset for modeling
features = vehdata[selected_features]
target = vehdata['total_vehicles']
```

a) Feature Selection for Experiment 1

```
# Scale features
scaler = MinMaxScaler(feature_range=(0, 1))
features_scaled = scaler.fit_transform(features)

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(features_scaled, target, test_size=0.2, random_state=42)

# Reshape for LSTM
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))
```

b) Normalization and Data Split for LSTM

```
# Scale features
scaler = MinMaxScaler(feature_range=(0, 1))
features_scaled = scaler.fit_transform(features)

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(features_scaled, target, test_size=0.2, random_state=42)

# Reshape for GRU
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))
```

c) Normalization and Data Split for GRU

```
# Scale features
scaler = MinMaxScaler(feature_range=(0, 1))
features_scaled = scaler.fit_transform(features)

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(features_scaled, target, test_size=0.2, random_state=42)

# Reshape input for CNN [samples, time steps, features]
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))
```

d) Normalization and Data Split for CNN

Figure 13: Feature Selection and Data Splitting Experiment 1

5.2 Feature Selection and Data Splitting – Experiment 2

Traffic flow prediction based on the influence of hour of the day shown in Figure 14.

```
# Feature Selection based on Correlation Analysis
selected_features = [
    'peak_non_peak_hour_evening_peak', 'hour', 'peak_non_peak_hour_morning_peak', 'peak_non_peak_hour_non_peak',
    'month', 'Year', 'Cars_and_taxis','All_HGVs', 'Two_wheeled_motor_vehicles'
]

# Preparing the dataset for modeling
features = hourdata[selected_features]
target = hourdata['All_motor_vehicles']
```

a) Feature Selection for Experiment 2

```
# Scale features
scaler = MinMaxScaler(feature_range=(0, 1))
features_scaled = scaler.fit_transform(features)

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(features_scaled, target, test_size=0.2, random_state=42)

# Reshape for LSTM
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))
```

b) Normalization and Data Split for LSTM

```
# Scale features
scaler = MinMaxScaler(feature_range=(0, 1))
features_scaled = scaler.fit_transform(features)

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(features_scaled, target, test_size=0.2, random_state=42)

# Reshape for GRU
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))
```

c) Normalization and Data Split for GRU

```
# Scale features
scaler = MinMaxScaler(feature_range=(0, 1))
features_scaled = scaler.fit_transform(features)

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(features_scaled, target, test_size=0.2, random_state=42)

# Reshape input for CNN [samples, time steps, features]
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))
```

d) Normalization and Data Split for CNN

Figure 14: Feature Selection and Data Splitting Experiment 2

5.3 Feature Selection and Data Splitting – Experiment 3

Traffic flow prediction based on the influence of geographical locations shown in Figure 15.

```
# Feature Selection based on Correlation Analysis
selected_features = [
    'Cars_and_taxis', 'Two_wheeled_motor_vehicles', 'All_HGVs', 'hour', 'Year',
    'month', 'Latitude', 'Longitude', 'distance_to_central_london', 'Easting', 'Northing', 'Region_id'
]

# Preparing the dataset for modeling
features = spatialdata[selected_features]
target = spatialdata['All_motor_vehicles']
```

a) Feature Selection for Experiment 3

```
# Scale features
scaler = MinMaxScaler(feature_range=(0, 1))
features_scaled = scaler.fit_transform(features)

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(features_scaled, target, test_size=0.2, random_state=42)

# Reshape for LSTM
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))
```

b) Normalization and Data Split for LSTM

```
# Scale features
scaler = MinMaxScaler(feature_range=(0, 1))
features_scaled = scaler.fit_transform(features)

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(features_scaled, target, test_size=0.2, random_state=42)

# Reshape for GRU
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))
```

c) Normalization and Data Split for GRU

```
# Scale features
scaler = MinMaxScaler(feature_range=(0, 1))
features_scaled = scaler.fit_transform(features)

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(features_scaled, target, test_size=0.2, random_state=42)

# Reshape input for CNN [samples, time steps, features]
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))
```

d) Normalization and Data Split for CNN

Figure 15: Feature Selection and Data Splitting Experiment 3

6. Code for deep learning models

The implementation of deep learning models is carried out using Keras and TensorFlow package of Python 3. LSTM, GRU, and CNN are the three models used in model building process to predict the traffic flow based on three different experiments (Coursera, n.d). Model building summary, model fitting using early stopping, and predictions on both train and test set code snippets are given below.

6.1 Experiment 1 – LSTM

LSTM model is applied to trained dataset, and models are predicted and evaluated for predicting traffic flow based on vehicles.

```
In [115]: # Building the LSTM model
          model = Sequential()
          model.add(LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2])))
          model.add(Dropout(0.2))
          model.add(Dense(1))
          model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')

          # Printing the model summary
          model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_4 (LSTM)               (None, 50)                13000

 dropout_5 (Dropout)         (None, 50)                0

 dense_5 (Dense)             (None, 1)                 51

=================================================================
Total params: 13051 (50.98 KB)
Trainable params: 13051 (50.98 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

a) LSTM Model Summary

```
# Early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Fit model with early stopping
history = model.fit(X_train, y_train, epochs=50, batch_size=50, validation_data=(X_test, y_test), verbose=1, shuffle=False,
                    callbacks=[early_stopping])
```

```
Epoch 50/50
5982/5982 [==============================] - 37s 6ms/step - loss: 14303.1416 - val_loss: 2613.6196
2337/2337 [==============================] - 7s 3ms/step
```

b) LSTM Model Fit

```
# Evaluating the model
loss = model.evaluate(X_test, y_test, verbose=0)
```

```
# Predictions on training set
y_train_pred = model.predict(X_train)
```

```
# Predictions
y_pred = model.predict(X_test)
```

c) LSTM Model Prediction

Figure 16: Deep Learning Model- LSTM Exp1

6.2 Experiment 1 – GRU

GRU model is applied to trained dataset, and models are predicted and evaluated for predicting traffic flow based on vehicles.

```
In [119]: # Building the GRU model
          model_gru = Sequential()
          model_gru.add(GRU(50, input_shape=(X_train.shape[1], X_train.shape[2])))
          model_gru.add(Dropout(0.2))
          model_gru.add(Dense(1))
          model_gru.compile(optimizer=Adam(learning_rate=0.001), loss='mse')

          # Print the model summary
          model_gru.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_1 (GRU)                 (None, 50)                9900

 dropout_6 (Dropout)         (None, 50)                0

 dense_6 (Dense)             (None, 1)                 51

=================================================================
Total params: 9951 (38.87 KB)
Trainable params: 9951 (38.87 KB)
Non-trainable params: 0 (0.00 Byte)
```

a) GRU Model Summary

```
# Early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Fit model with early stopping
history = model_gru.fit(X_train, y_train, epochs=50, batch_size=50, validation_data=(X_test, y_test), verbose=1, shuffle=False,
                        callbacks=[early_stopping])
```

```
Epoch 50/50
5982/5982 [==============================] - 38s 6ms/step - loss: 11298.7773 - val_loss: 1183.6598
2337/2337 [==============================] - 7s 3ms/step
```

b) GRU Model Fit

```
# Evaluating the model
loss = model_gru.evaluate(X_test, y_test, verbose=0)
```

```
# Predictions
y_pred = model_gru.predict(X_test)
```

```
In [120]: # Predictions on training set
          y_train_pred = model.predict(X_train)
```

c) GRU Model Predict

Figure 17: Deep Learning Model- GRU Exp1

6.3 Experiment 1 – CNN

CNN model is applied to trained dataset, and models are predicted and evaluated for predicting traffic flow based on vehicles.

```
In [123]: # Building the CNN model
model_cnn = Sequential()
model_cnn.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)))
model_cnn.add(MaxPooling1D(pool_size=2))
model_cnn.add(Flatten())
model_cnn.add(Dense(50, activation='relu'))
model_cnn.add(Dropout(0.5))
model_cnn.add(Dense(1))
model_cnn.compile(optimizer=Adam(learning_rate=0.001), loss='mse')

# Printing the model summary
model_cnn.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 12, 64)            256

 max_pooling1d (MaxPooling1  (None, 6, 64)             0
 D)

 flatten (Flatten)           (None, 384)               0

 dense_7 (Dense)             (None, 50)                19250

 dropout_7 (Dropout)         (None, 50)                0

 dense_8 (Dense)             (None, 1)                 51

=================================================================
Total params: 19557 (76.39 KB)
Trainable params: 19557 (76.39 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

a) CNN Model Summary

```
# Early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Fit model with early stopping
history = model_cnn.fit(X_train, y_train, epochs=50, batch_size=50, validation_data=(X_test, y_test), verbose=1, shuffle=False,
                        callbacks=[early_stopping])
```

```
Epoch 1/50
5982/5982 [==============================] - 37s 6ms/step - loss: 278189.7500 - val_loss: 10043.3613
Epoch 2/50
5982/5982 [==============================] - 37s 6ms/step - loss: 75667.5938 - val_loss: 7782.8794
Epoch 3/50
5982/5982 [==============================] - 34s 6ms/step - loss: 75224.5391 - val_loss: 9625.3047
Epoch 4/50
5982/5982 [==============================] - 35s 6ms/step - loss: 74251.1719 - val_loss: 9064.9932
Epoch 5/50
5982/5982 [==============================] - 36s 6ms/step - loss: 73470.8594 - val_loss: 8624.3555
Epoch 6/50
5982/5982 [==============================] - 36s 6ms/step - loss: 71994.2812 - val_loss: 7837.1328
Epoch 7/50
5982/5982 [==============================] - 35s 6ms/step - loss: 73701.9141 - val_loss: 7858.5166
2337/2337 [==============================] - 6s 3ms/step
```

b) CNN Model Fit

```
# Evaluating the model
loss = model_cnn.evaluate(X_test, y_test, verbose=0)
```

```
In [125]: # predictions on training set
y_train_pred_cnn = model_cnn.predict(X_train)

# predictions on test set
y_pred_cnn = model_cnn.predict(X_test)
```

c) CNN Model Prediction

Figure 18: Deep Learning Model- CNN Exp1

6.4 Experiment 2 – LSTM

LSTM model is applied to trained dataset, and models are predicted and evaluated for predicting traffic flow based on hour of the day.

In [26]:
```
# Building the LSTM model
model = Sequential()
model.add(LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.2))  # Adjust dropout rate as needed
model.add(Dense(1))
model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 50)                12000

 dropout (Dropout)           (None, 50)                0

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 12051 (47.07 KB)
Trainable params: 12051 (47.07 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

a) LSTM Model Summary

```
model.summary()

# Early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Fit model with early stopping
history = model.fit(X_train, y_train, epochs=50, batch_size=50, validation_data=(X_test, y_test), verbose=1, shuffle=False,
                    callbacks=[early_stopping])
```

```
Epoch 28/50
15875/15875 [==============================] - 94s 6ms/step - loss: 765.3450 - val_loss: 283.9316
Epoch 29/50
15875/15875 [==============================] - 92s 6ms/step - loss: 765.4388 - val_loss: 285.4932
Epoch 30/50
15875/15875 [==============================] - 94s 6ms/step - loss: 758.5532 - val_loss: 288.6388
Epoch 31/50
15875/15875 [==============================] - 95s 6ms/step - loss: 756.4000 - val_loss: 285.7403
6202/6202 [==============================] - 18s 3ms/step
```

b) LSTM Model Fit

```
# Evaluating the model
loss = model.evaluate(X_test, y_test, verbose=0)
```

```
# Predictions
y_pred = model.predict(X_test)
```

In [27]:
```
# Predictions on training set
y_train_pred = model.predict(X_train)
```

c) LSTM Model Prediction

Figure 19: Deep Learning Model- LSTM Exp2

6.5 Experiment 2 – GRU

GRU model is applied to trained dataset, and models are predicted and evaluated for predicting traffic flow based on hour of the day.

```
In [30]: # Building the GRU model
         model_gru = Sequential()
         model_gru.add(GRU(50, input_shape=(X_train.shape[1], X_train.shape[2])))
         model_gru.add(Dropout(0.2))
         model_gru.add(Dense(1))
         model_gru.compile(optimizer=Adam(learning_rate=0.001), loss='mse')

         # Print the model summary
         model_gru.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (None, 50)                9150

 dropout_1 (Dropout)         (None, 50)                0

 dense_1 (Dense)             (None, 1)                 51

=================================================================
Total params: 9201 (35.94 KB)
Trainable params: 9201 (35.94 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

a) GRU Model Summary

```
# Early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Fit model with early stopping
history = model_gru.fit(X_train, y_train, epochs=50, batch_size=50, validation_data=(X_test, y_test), verbose=1, shuffle=False,
                        callbacks=[early_stopping])
```

```
Epoch 36/50
15875/15875 [==============================] - 84s 5ms/step - loss: 731.4009 - val_loss: 277.7617
Epoch 37/50
15875/15875 [==============================] - 83s 5ms/step - loss: 727.1448 - val_loss: 276.4960
Epoch 38/50
15875/15875 [==============================] - 85s 5ms/step - loss: 723.4516 - val_loss: 275.9544
6202/6202 [==============================] - 17s 3ms/step
```

b) GRU Model Fit

```
# Evaluating the model
loss = model_gru.evaluate(X_test, y_test, verbose=0)
```

```
# Predictions
y_pred = model_gru.predict(X_test)
```

```
In [31]: # Predictions on training set
         y_train_pred = model.predict(X_train)
```

c) GRU Model Prediction

Figure 20: Deep Learning Model- GRU Exp2

6.6 Experiment 2 – CNN

CNN model is applied to trained dataset, and models are predicted and evaluated for predicting traffic flow based on hour of the day.

```
In [34]: # Building the CNN model
         model_cnn = Sequential()
         model_cnn.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)))
         model_cnn.add(MaxPooling1D(pool_size=2))
         model_cnn.add(Flatten())
         model_cnn.add(Dense(50, activation='relu'))
         model_cnn.add(Dropout(0.5))
         model_cnn.add(Dense(1))
         model_cnn.compile(optimizer=Adam(learning_rate=0.001), loss='mse')

         # Print the model summary
         model_cnn.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 7, 64)             256

 max_pooling1d (MaxPooling1  (None, 3, 64)             0
 D)

 flatten (Flatten)           (None, 192)               0

 dense_2 (Dense)             (None, 50)                9650

 dropout_2 (Dropout)         (None, 50)                0

 dense_3 (Dense)             (None, 1)                 51

=================================================================
Total params: 9957 (38.89 KB)
Trainable params: 9957 (38.89 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

a) CNN Model Summary

```
# Early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Fit model with early stopping
history = model_cnn.fit(X_train, y_train, epochs=50, batch_size=50, validation_data=(X_test, y_test), verbose=1, shuffle=False,
                        callbacks=[early_stopping])
```

```
15875/15875 [==============================] - 65s 4ms/step - loss: 4333.5488 - val_loss: 902.3706
Epoch 38/50
15875/15875 [==============================] - 65s 4ms/step - loss: 4307.4878 - val_loss: 909.3389
Epoch 39/50
15875/15875 [==============================] - 63s 4ms/step - loss: 4289.8818 - val_loss: 1095.8307
```

b) CNN Model Fit

```
# Evaluating the model
loss = model_cnn.evaluate(X_test, y_test, verbose=0)
```

```
In [35]: # Correct predictions on training set for CNN
y_train_pred_cnn = model_cnn.predict(X_train)

# Predictions
y_pred_cnn = model_cnn.predict(X_test)
```

c) CNN Model Prediction

Figure 21: Deep Learning Model- CNN Exp2

6.7 Experiment 3 – LSTM

LSTM model is applied to trained dataset, and models are predicted and evaluated for predicting traffic flow based on geographical location.

```
In [35]: # Building the LSTM model
model = Sequential()
model.add(LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.2))  # Adjust dropout rate as needed
model.add(Dense(1))
model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 50)                12600

 dropout (Dropout)           (None, 50)                0

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 12651 (49.42 KB)
Trainable params: 12651 (49.42 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

a) LSTM Model Summary

```
# Early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Fit model with early stopping
history = model.fit(X_train, y_train, epochs=50, batch_size=50, validation_data=(X_test, y_test), verbose=1, shuffle=False,
                    callbacks=[early_stopping])
```

```
Epoch 41/50
15875/15875 [==============================] - 99s 6ms/step - loss: 732.3796 - val_loss: 283.9303
Epoch 42/50
15875/15875 [==============================] - 95s 6ms/step - loss: 733.1993 - val_loss: 285.9540
6202/6202 [==============================] - 19s 3ms/step
```

b) CNN Model Fit

```
# Evaluating the model
loss = model.evaluate(X_test, y_test, verbose=0)
```

```
# Predictions
y_pred = model.predict(X_test)
```

```
In [36]: # Predictions on training set
         y_train_pred = model.predict(X_train)
```

c) LSTM Model Prediction

Figure 22: Deep Learning Model- LSTM Exp3

6.8 Experiment 3 – GRU

GRU model is applied to trained dataset, and models are predicted and evaluated for predicting traffic flow based on geographical location.

```
In [39]: # Building the GRU model
         model_gru = Sequential()
         model_gru.add(GRU(50, input_shape=(X_train.shape[1], X_train.shape[2])))
         model_gru.add(Dropout(0.2))
         model_gru.add(Dense(1))
         model_gru.compile(optimizer=Adam(learning_rate=0.001), loss='mse')

         # Print the model summary
         model_gru.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (None, 50)                9600

 dropout_1 (Dropout)         (None, 50)                0

 dense_1 (Dense)             (None, 1)                 51

=================================================================
Total params: 9651 (37.70 KB)
Trainable params: 9651 (37.70 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

a) GRU Model Summary

```
# Early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Fit model with early stopping
history = model_gru.fit(X_train, y_train, epochs=50, batch_size=50, validation_data=(X_test, y_test), verbose=1, shuffle=False,
                        callbacks=[early_stopping])
```

```
Epoch 28/50
15875/15875 [==============================] - 85s 5ms/step - loss: 744.0431 - val_loss: 267.4744
Epoch 29/50
15875/15875 [==============================] - 86s 5ms/step - loss: 742.1849 - val_loss: 277.6607
6202/6202 [==============================] - 16s 2ms/step
```

b) GRU Model Fit

```
# Evaluating the model
loss = model_gru.evaluate(X_test, y_test, verbose=0)
```

```
# Predictions
y_pred = model_gru.predict(X_test)
```

```
In [40]: # Predictions on training set
         y_train_pred = model.predict(X_train)
```

c) GRU Model Prediction

Figure 23: Deep Learning Model- GRU Exp3

6.9 Experiment 3 – CNN

CNN model is applied to trained dataset, and models are predicted and evaluated for predicting traffic flow based on geographical location.

```
In [43]: # Building the CNN model
         model_cnn = Sequential()
         model_cnn.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)))
         model_cnn.add(MaxPooling1D(pool_size=2))
         model_cnn.add(Flatten())
         model_cnn.add(Dense(50, activation='relu'))
         model_cnn.add(Dropout(0.5))
         model_cnn.add(Dense(1))
         model_cnn.compile(optimizer=Adam(learning_rate=0.001), loss='mse')

         # Print the model summary
         model_cnn.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 10, 64)            256

 max_pooling1d (MaxPooling1  (None, 5, 64)             0
 D)

 flatten (Flatten)           (None, 320)               0

 dense_2 (Dense)             (None, 50)                16050

 dropout_2 (Dropout)         (None, 50)                0

 dense_3 (Dense)             (None, 1)                 51

=================================================================
Total params: 16357 (63.89 KB)
Trainable params: 16357 (63.89 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

a) CNN Model Summary

```
# Early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Fit model with early stopping
history = model_cnn.fit(X_train, y_train, epochs=50, batch_size=50, validation_data=(X_test, y_test), verbose=1, shuffle=False,
                        callbacks=[early_stopping])
Epoch 1/50
15875/15875 [==============================] - 83s 5ms/step - loss: 10179.1484 - val_loss: 683.9909
Epoch 2/50
15875/15875 [==============================] - 77s 5ms/step - loss: 5143.6597 - val_loss: 502.0124
Epoch 3/50
15875/15875 [==============================] - 79s 5ms/step - loss: 4963.0352 - val_loss: 626.9385
Epoch 4/50
15875/15875 [==============================] - 79s 5ms/step - loss: 4801.7310 - val_loss: 590.7245
Epoch 5/50
15875/15875 [==============================] - 80s 5ms/step - loss: 4664.3931 - val_loss: 641.8464
Epoch 6/50
15875/15875 [==============================] - 78s 5ms/step - loss: 4605.7402 - val_loss: 627.2719
Epoch 7/50
15875/15875 [==============================] - 79s 5ms/step - loss: 4554.5952 - val_loss: 471.8364
Epoch 8/50
15875/15875 [==============================] - 82s 5ms/step - loss: 4475.7485 - val_loss: 544.5089
Epoch 9/50
15875/15875 [==============================] - 78s 5ms/step - loss: 4434.5298 - val_loss: 551.2283
Epoch 10/50
15875/15875 [==============================] - 79s 5ms/step - loss: 4407.6089 - val_loss: 503.3484
Epoch 11/50
15875/15875 [==============================] - 78s 5ms/step - loss: 4365.4771 - val_loss: 525.0794
Epoch 12/50
15875/15875 [==============================] - 81s 5ms/step - loss: 4330.6934 - val_loss: 513.7418
```

b) CNN Model Fit

```
# Evaluating the model
loss = model_cnn.evaluate(X_test, y_test, verbose=0)
```

```
In [44]: # predictions on training set
y_train_pred_cnn = model_cnn.predict(X_train)

# Predictions
y_pred_cnn = model_cnn.predict(X_test)
```

c) CNN Model Prediction

Figure 24: Deep Learning Model- CNN Exp3

7 Evaluation Output

Finally, the models are evaluated using RMSE, MSE, MAE, and R squared values from sklean.metrics. Additionally, to check whether model is an overfit or underfit or perfect fit, training and validation loss is plotted against number of epochs.

7.1 Experiment 1 – Evaluation Code Snippet

```
# Calculating evaluation metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

# Calculating evaluation metrics for the training set
mse_train = mean_squared_error(y_train, y_train_pred)
rmse_train = np.sqrt(mse_train)
mae_train = mean_absolute_error(y_train, y_train_pred)
r2_train = r2_score(y_train, y_train_pred)

# Output the metrics for the training set
print('Training Values for LSTM: ')
print('Training Mean Squared Error:', mse_train)
print('Training Root Mean Squared Error:', rmse_train)
print('Training Mean Absolute Error:', mae_train)
print('Training R-squared:', r2_train)

# Output the metrics for the test set
print('Testing Values for LSTM: ')
print('Test Mean Squared Error:', mse)
print('Test Root Mean Squared Error:', rmse)
print('Test Mean Absolute Error:', mae)
print('Test R-squared:', r2)
```

```
9347/9347 [==============================] - 27s 3ms/step
Training Values for LSTM:
Training Mean Squared Error: 2866.8542693254594
Training Root Mean Squared Error: 53.54301326340776
Training Mean Absolute Error: 9.953320852017534
Training R-squared: 0.9972613596062934
Testing Values for LSTM:
Test Mean Squared Error: 2613.6192631292106
Test Root Mean Squared Error: 51.123568568021646
Test Mean Absolute Error: 9.80746580526023
Test R-squared: 0.9975185974672297
```

a) Evaluation Metrics LSTM

```
# Calculating evaluation metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

# Calculating evaluation metrics for the training set
mse_train = mean_squared_error(y_train, y_train_pred)
rmse_train = np.sqrt(mse_train)
mae_train = mean_absolute_error(y_train, y_train_pred)
r2_train = r2_score(y_train, y_train_pred)

# Output the metrics for the training set
print('Training Values for GRU: ')
print('Training Mean Squared Error:', mse_train)
print('Training Root Mean Squared Error:', rmse_train)
print('Training Mean Absolute Error:', mae_train)
print('Training R-squared:', r2_train)

# Output the metrics for the test set
print('Testing Values for GRU: ')
print('Test Mean Squared Error:', mse)
print('Test Root Mean Squared Error:', rmse)
print('Test Mean Absolute Error:', mae)
print('Test R-squared:', r2)
```

```
9347/9347 [==============================] - 27s 3ms/step
Training Values for GRU:
Training Mean Squared Error: 2866.8542693254594
Training Root Mean Squared Error: 53.54301326340776
Training Mean Absolute Error: 9.953320852017534
Training R-squared: 0.9972613596062934
Testing Values for GRU:
Test Mean Squared Error: 1183.6602182549261
Test Root Mean Squared Error: 34.404363360697815
Test Mean Absolute Error: 8.474487739256475
Test R-squared: 0.9988762183134506
```

b) Evaluation Metrics GRU

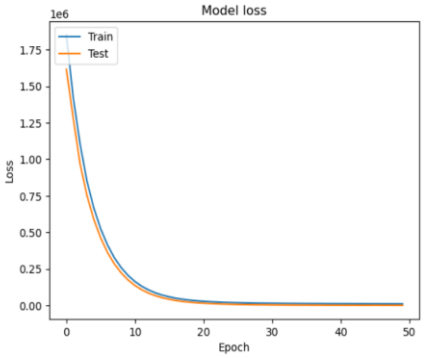
```
# calculations of evaluation metrics for the training set of CNN
mse_train_cnn = mean_squared_error(y_train, y_train_pred_cnn)
rmse_train_cnn = np.sqrt(mse_train_cnn)
mae_train_cnn = mean_absolute_error(y_train, y_train_pred_cnn)
r2_train_cnn = r2_score(y_train, y_train_pred_cnn)

# Output the correct metrics for the training set of CNN
print('Training Values for CNN: ')
print('Training Mean Squared Error:', mse_train_cnn)
print('Training Root Mean Squared Error:', rmse_train_cnn)
print('Training Mean Absolute Error:', mae_train_cnn)
print('Training R-squared:', r2_train_cnn)

# Calculate the evaluation metrics for the test set of CNN
mse_test_cnn = mean_squared_error(y_test, y_pred_cnn)
mae_test_cnn = mean_absolute_error(y_test, y_pred_cnn)
rmse_test_cnn = np.sqrt(mse_test_cnn)
r2_test_cnn = r2_score(y_test, y_pred_cnn)

# Output the metrics for the test set of CNN
print('Testing Values for CNN: ')
print('Test Mean Squared Error:', mse_test_cnn)
print('Test Root Mean Squared Error:', rmse_test_cnn)
print('Test Mean Absolute Error:', mae_test_cnn)
print('Test R-squared:', r2_test_cnn)
```

```
9347/9347 [==============================] - 22s 2ms/step
Training Values for CNN:
Training Mean Squared Error: 8045.51928841366
Training Root Mean Squared Error: 89.69681871958258
Training Mean Absolute Error: 41.0673358894712
Training R-squared: 0.9923142992138279
2337/2337 [==============================] - 5s 2ms/step
Testing Values for CNN:
Test Mean Squared Error: 7782.875646874932
Test Root Mean Squared Error: 88.22060783555581
Test Mean Absolute Error: 40.73667280995265
Test R-squared: 0.9926108413666688
```

c) Evaluation Metrics CNN

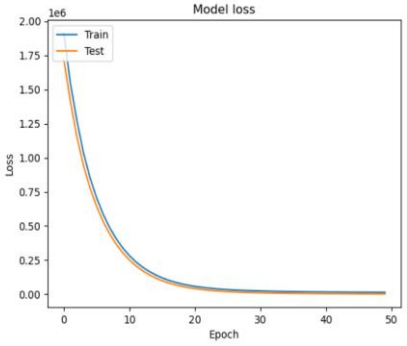
Figure 25: Evaluation code for Experiment 1

```
In [121]: # Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



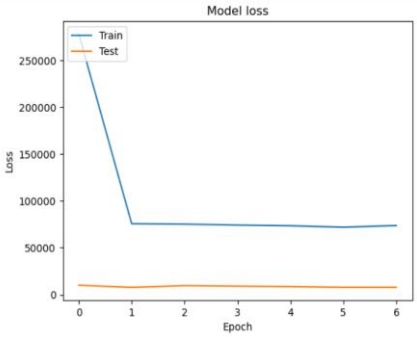
a) Loss Vs Epochs LSTM

```
In [117]: # Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



b) Loss Vs Epochs GRU

```
In [126]: # Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



c) Loss Vs Epochs CNN

Figure 26: Loss Vs Epoch for Experiment 1

7.2 Experiment 2 - Evaluation Code Snippet

```
# Calculating evaluation metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

# Calculating evaluation metrics for the training set
mse_train = mean_squared_error(y_train, y_train_pred)
rmse_train = np.sqrt(mse_train)
mae_train = mean_absolute_error(y_train, y_train_pred)
r2_train = r2_score(y_train, y_train_pred)

# Output the metrics for the training set
print('Training Values for LSTM: ')
print('Training Mean Squared Error:', mse_train)
print('Training Root Mean Squared Error:', rmse_train)
print('Training Mean Absolute Error:', mae_train)
print('Training R-squared:', r2_train)

# Output the metrics for the test set
print('Testing Values for LSTM: ')
print('Test Mean Squared Error:', mse)
print('Test Root Mean Squared Error:', rmse)
print('Test Mean Absolute Error:', mae)
print('Test R-squared:', r2)
```

```
24805/24805 [==============================] - 69s 3ms/step
Training Values for LSTM:
Training Mean Squared Error: 279.60646405247275
Training Root Mean Squared Error: 16.72143726036948
Training Mean Absolute Error: 9.281141248742395
Training R-squared: 0.9963252267312451
Testing Values for LSTM:
Test Mean Squared Error: 280.2695915193105
Test Root Mean Squared Error: 16.741254179998297
Test Mean Absolute Error: 9.28045719389702
Test R-squared: 0.996313474454926
```

a) Evaluation Metrics LSTM

```
# Calculating evaluation metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

# Calculating evaluation metrics for the training set
mse_train = mean_squared_error(y_train, y_train_pred)
rmse_train = np.sqrt(mse_train)
mae_train = mean_absolute_error(y_train, y_train_pred)
r2_train = r2_score(y_train, y_train_pred)

# Output the metrics for the training set
print('Training Values for GRU: ')
print('Training Mean Squared Error:', mse_train)
print('Training Root Mean Squared Error:', rmse_train)
print('Training Mean Absolute Error:', mae_train)
print('Training R-squared:', r2_train)

# Output the metrics for the test set
print('Testing Values for GRU: ')
print('Test Mean Squared Error:', mse)
print('Test Root Mean Squared Error:', rmse)
print('Test Mean Absolute Error:', mae)
print('Test R-squared:', r2)
```

```
24805/24805 [==============================] - 62s 2ms/step
Training Values for GRU:
Training Mean Squared Error: 279.60646405247275
Training Root Mean Squared Error: 16.72143726036948
Training Mean Absolute Error: 9.281141248742395
Training R-squared: 0.9963252267312451
Testing Values for GRU:
Test Mean Squared Error: 273.7050664032105
Test Root Mean Squared Error: 16.544034163504694
Test Mean Absolute Error: 9.106644350197513
Test R-squared: 0.9963998209236977
```

b) Evaluation Metrics GRU

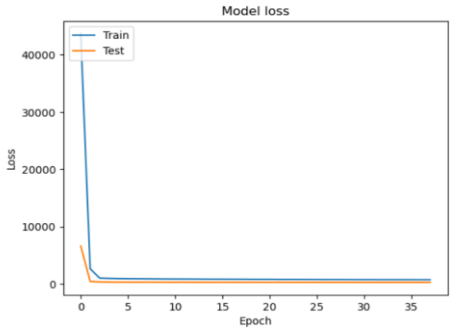
```
#  calculations of evaluation metrics for the training set of CNN
mse_train_cnn = mean_squared_error(y_train, y_train_pred_cnn)
rmse_train_cnn = np.sqrt(mse_train_cnn)
mae_train_cnn = mean_absolute_error(y_train, y_train_pred_cnn)
r2_train_cnn = r2_score(y_train, y_train_pred_cnn)

# Output the correct metrics for the training set of CNN
print('Training Values for CNN: ')
print('Training Mean Squared Error:', mse_train_cnn)
print('Training Root Mean Squared Error:', rmse_train_cnn)
print('Training Mean Absolute Error:', mae_train_cnn)
print('Training R-squared:', r2_train_cnn)

# Calculate the evaluation metrics for the test set of CNN
mse_test_cnn = mean_squared_error(y_test, y_pred_cnn)
mae_test_cnn = mean_absolute_error(y_test, y_pred_cnn)
rmse_test_cnn = np.sqrt(mse_test_cnn)
r2_test_cnn = r2_score(y_test, y_pred_cnn)

# Output the metrics for the test set of CNN
print('Testing Values for CNN: ')
print('Test Mean Squared Error:', mse_test_cnn)
print('Test Root Mean Squared Error:', rmse_test_cnn)
print('Test Mean Absolute Error:', mae_test_cnn)
print('Test R-squared:', r2_test_cnn)
```

```
24805/24805 [==============================] - 52s 2ms/step
Training Values for CNN:
Training Mean Squared Error: 881.9348361536269
Training Root Mean Squared Error: 29.69738769915002
Training Mean Absolute Error: 18.432469772970748
Training R-squared: 0.9884090284834299
6202/6202 [==============================] - 14s 2ms/step
Testing Values for CNN:
Test Mean Squared Error: 893.4755255135284
Test Root Mean Squared Error: 29.891060963330297
Test Mean Absolute Error: 18.529884814214007
Test R-squared: 0.9882476713551098
```

c) Evaluation Metrics CNN

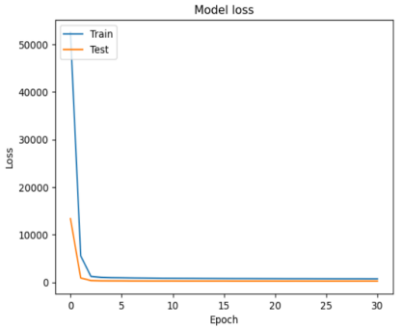
Figure 27: Evaluation code for Experiment 2

```
In [32]: # Plot training & validation loss values
         plt.plot(history.history['loss'])
         plt.plot(history.history['val_loss'])
         plt.title('Model loss')
         plt.ylabel('Loss')
         plt.xlabel('Epoch')
         plt.legend(['Train', 'Test'], loc='upper left')
         plt.show()
```



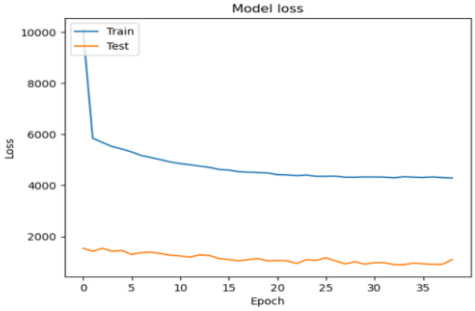
a) Loss Vs Epochs LSTM

```
In [28]: # Plot training & validation loss values
         plt.plot(history.history['loss'])
         plt.plot(history.history['val_loss'])
         plt.title('Model loss')
         plt.ylabel('Loss')
         plt.xlabel('Epoch')
         plt.legend(['Train', 'Test'], loc='upper left')
         plt.show()
```



b) Loss Vs Epochs GRU

```
In [36]: # Plot training & validation loss values
         plt.plot(history.history['loss'])
         plt.plot(history.history['val_loss'])
         plt.title('Model loss')
         plt.ylabel('Loss')
         plt.xlabel('Epoch')
         plt.legend(['Train', 'Test'], loc='upper left')
         plt.show()
```



C) Loss Vs Epochs CNN

Figure 28: Loss Vs Epoch for Experiment 2

7.3 Experiment 3 - Evaluation Code Snippet

```python
# Calculating evaluation metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

# Calculating evaluation metrics for the training set
mse_train = mean_squared_error(y_train, y_train_pred)
rmse_train = np.sqrt(mse_train)
mae_train = mean_absolute_error(y_train, y_train_pred)
r2_train = r2_score(y_train, y_train_pred)

# Output the metrics for the training set
print('Training Values for LSTM: ')
print('Training Mean Squared Error:', mse_train)
print('Training Root Mean Squared Error:', rmse_train)
print('Training Mean Absolute Error:', mae_train)
print('Training R-squared:', r2_train)

# Output the metrics for the test set
print('Testing Values for LSTM: ')
print('Test Mean Squared Error:', mse)
print('Test Root Mean Squared Error:', rmse)
print('Test Mean Absolute Error:', mae)
print('Test R-squared:', r2)
```

```
24805/24805 [==============================] - 76s 3ms/step
Training Values for LSTM:
Training Mean Squared Error: 279.4944180583321
Training Root Mean Squared Error: 16.718086554936008
Training Mean Absolute Error: 9.384604342704892
Training R-squared: 0.9963266993138821
Testing Values for LSTM:
Test Mean Squared Error: 280.5003089613835
Test Root Mean Squared Error: 16.74814344819698
Test Mean Absolute Error: 9.38578708701721
Test R-squared: 0.9963104397134854
```

a) Evaluation Metrics LSTM

```python
# Calculating evaluation metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

# Calculating evaluation metrics for the training set
mse_train = mean_squared_error(y_train, y_train_pred)
rmse_train = np.sqrt(mse_train)
mae_train = mean_absolute_error(y_train, y_train_pred)
r2_train = r2_score(y_train, y_train_pred)

# Output the metrics for the training set
print('Training Values for GRU: ')
print('Training Mean Squared Error:', mse_train)
print('Training Root Mean Squared Error:', rmse_train)
print('Training Mean Absolute Error:', mae_train)
print('Training R-squared:', r2_train)

# Output the metrics for the test set
print('Testing Values for GRU: ')
print('Test Mean Squared Error:', mse)
print('Test Root Mean Squared Error:', rmse)
print('Test Mean Absolute Error:', mae)
print('Test R-squared:', r2)
```

```
24805/24805 [==============================] - 64s 3ms/step
Training Values for GRU:
Training Mean Squared Error: 279.4944180583321
Training Root Mean Squared Error: 16.718086554936008
Training Mean Absolute Error: 9.384604342704892
Training R-squared: 0.9963266993138821
Testing Values for GRU:
Test Mean Squared Error: 267.38715114064496
Test Root Mean Squared Error: 16.35197697957788
Test Mean Absolute Error: 9.261811250598205
Test R-squared: 0.9964829236102246
```

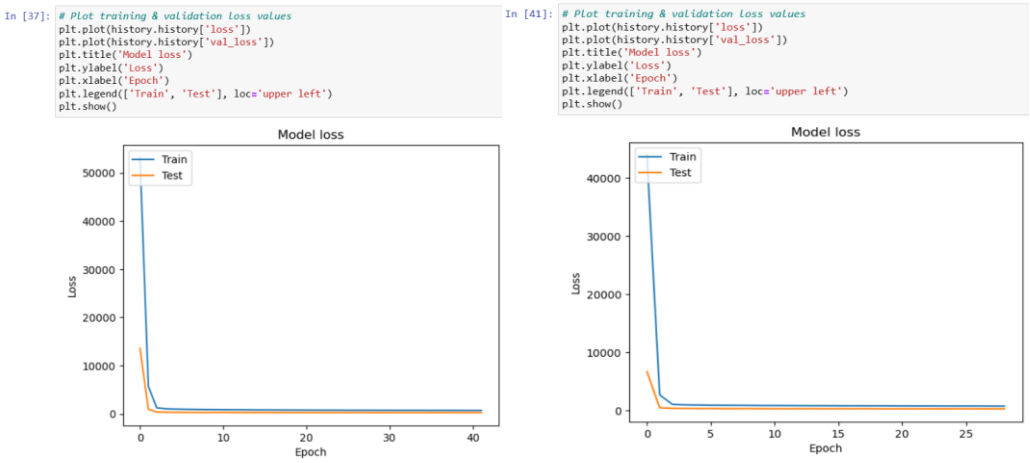
b) Evaluation Metrics GRU

```python
# calculations of evaluation metrics for the training set of CNN
mse_train_cnn = mean_squared_error(y_train, y_train_pred_cnn)
rmse_train_cnn = np.sqrt(mse_train_cnn)
mae_train_cnn = mean_absolute_error(y_train, y_train_pred_cnn)
r2_train_cnn = r2_score(y_train, y_train_pred_cnn)

# Output the correct metrics for the training set of CNN
print('Training Values for CNN: ')
print('Training Mean Squared Error:', mse_train_cnn)
print('Training Root Mean Squared Error:', rmse_train_cnn)
print('Training Mean Absolute Error:', mae_train_cnn)
print('Training R-squared:', r2_train_cnn)

# Calculate the evaluation metrics for the test set of CNN
mse_test_cnn = mean_squared_error(y_test, y_pred_cnn)
mae_test_cnn = mean_absolute_error(y_test, y_pred_cnn)
rmse_test_cnn = np.sqrt(mse_test_cnn)
r2_test_cnn = r2_score(y_test, y_pred_cnn)

# Output the metrics for the test set of CNN
print('Testing Values for CNN: ')
print('Test Mean Squared Error:', mse_test_cnn)
print('Test Root Mean Squared Error:', rmse_test_cnn)
print('Test Mean Absolute Error:', mae_test_cnn)
print('Test R-squared:', r2_test_cnn)
```

```
24805/24805 [==============================] - 57s 2ms/step
Training Values for CNN:
Training Mean Squared Error: 463.7806548600278
Training Root Mean Squared Error: 21.535567205440117
Training Mean Absolute Error: 13.726579847839837
Training R-squared: 0.9939046875800216
6202/6202 [==============================] - 14s 2ms/step
Testing Values for CNN:
Test Mean Squared Error: 471.83643841469024
Test Root Mean Squared Error: 21.721796390139794
Test Mean Absolute Error: 13.773959036985
Test R-squared: 0.99379370029448
```

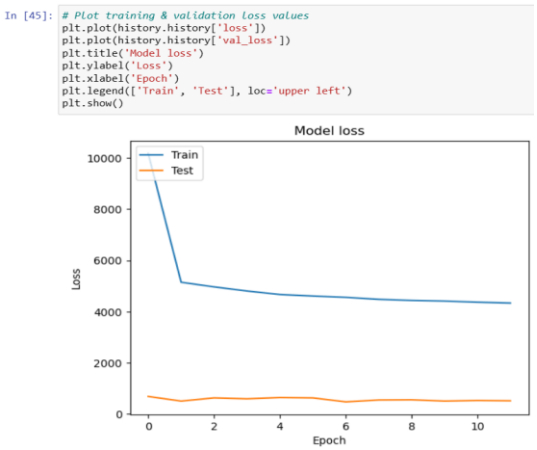
c) Evaluation Metrics CNN

Figure 29: Evaluation code for Experiment 3



a) Loss Vs Epochs LSTM

b) Loss Vs Epochs GRU



c) Loss Vs Epochs CNN

Figure 30: Loss Vs Epoch for Experiment 2

References

Coursera, n.d. Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization. *https://www.coursera.org/learn/deep-neural-network.*