INSTITUTE FOR ADVANCED COMPUTING AND SOFTWARE DEVELOPMENT, AKURDI, PUNE

Develop a predictive model to accurately forecast hourly traffic volumes at different road junctions based on historical traffic data

PG-DBDA March 2024

Submitted By:

Group No.: 8

RollNo.	Name
243539	Pranjal Pratap Singh
243525	Yash Ramesh Kunjir

Mr. Abhijit Nagargoje
Project Guide

Mr. Rohit Puranik
Centre Coordinator

ABSTRACT

Ride-sharing services like Uber have revolutionized transportation by providing convenient and affordable rides. As their core business of transportation services solidifies, ride-sharing companies are now solving other relevant problems that optimize their core business offering. One such opportunity is to solve problem of traffic congestion, a growing problem across metropolitan cities around the world.

Traffic volume is a critical piece of information in many applications, such as managing traffic at peak hours, companies like uber can optimize their ride flow, transportation long-range planning and traffic operation analysis. Effectively capturing traffic volumes on a network scale is beneficial to Transport department, private mobility organizations like Uber, Ola, Rapido etc.

This report presents an innovative prediction method of hourly traffic volume on a network scale. To achieve this, we applied a state-of-the-art tree ensemble model - extreme gradient boosting tree (XGBoost) - to handle the large-scale features and hourly traffic volume samples, due to the model's powerful scalability, Autoregressive Integrated Moving Average (ARIMA) – to handle timeseries data and Long Short Term Memory (LSTM) – to handle timeseries data and long term dependencies.

ACKNOWLEDGEMENT

I take this occasion to thank God, almighty for blessing us with his grace and taking our endeavor to a successful culmination. I extend my sincere and heartfelt thanks to our esteemed guide, **Mr. Abhijit Nagargoje, Dr. Shantanu Pathak**, **Mrs. Priti Take**, **and Mrs. Priyanka Bhor** for providing me with the right guidance and advice at the crucial juncture for showing me the right way. Centre Co-Ordinator **Mr. Rohit Puranik**, for allowing us to use the facilities available. I would like to thank the other faculty members also, at this occasion. Last but not the least, I would like to thank my friends and family for the support and encouragement they have given me during the course of our work.

Pranjal Pratap Singh (243539)
Yash Ramesh Kunjir (243525)

Table of Contents

List of Figures

1. Introduction

Traffic monitoring refers to the real-time collection and analysis of data related to vehicle movement and traffic conditions in order to facilitate traffic management and improve road safety.

Ride-sharing services like Uber have revolutionized transportation by providing convenient and affordable rides. As their core business of transportation services solidifies, ride-sharing companies are now solving other relevant problems that optimize their core business offering. One such opportunity is to solve problem of traffic congestion, a growing problem across metropolitan cities around the world.

1.1 Problem Statement:

Develop a predictive model to accurately forecast hourly traffic volumes at different road junctions based on historical traffic data.

1.2 Product Scope:

The main use of this predictive models is to accurately forecast hourly traffic volumes at different road junctions based on historical traffic data. Commonly seen in urban areas during peak hours, traffic congestion can result from various factors such as road capacity limitations, traffic incidents, road work, and high vehicle density, the last one becoming a more serious problem with the growth of vehicle ownership in urban households.

1.3 Aims & Objectives:

The primary goal of this project is to extract quality patterns from the Uber dataset and use the trained models to predict any question given by the Uber. Before giving the question the training of the models will be done using a bunch of different ML models and after the training is done the ML models will be compared based on their accuracy score and mean square error and the best model will be selected which will then be used to make prediction for the question given by the user. The prediction which is given by our system will give the user the quite insight of the traffic flow and accordingly uber can plan their rides.

2. Overall Description

2.1 Tool Description:

- **Google Colab:** Colab is a hosted Jupyter Notebook service that requires no setup to use and provides free of charge access to computing resources, including GPUs and TPUs.
- **Excel Sheet:** Excel is a spreadsheet program from Microsoft and a component of its Office product group for business applications.
- **Tableau:** Tableau is Business Intelligence Software that helps people see and understand their data. Tableau is a data visualization tool.

2.2 Data Source:

- Kaggle

2.3 Data Description:

- Data is hourly time series.
- Data have four features:
  - Hourly Date Time series
  - Junctions
  - Vehicle Count
  - ID
- Weather Data: It has four features
  - Precipitation
  - Humidity
  - Wind Speed
  - Temperature
- Event Data: It consist of list of Holidays both gazette and restricted.

2.4 Workflow of Project:

The diagram below shows the workflow of this project.

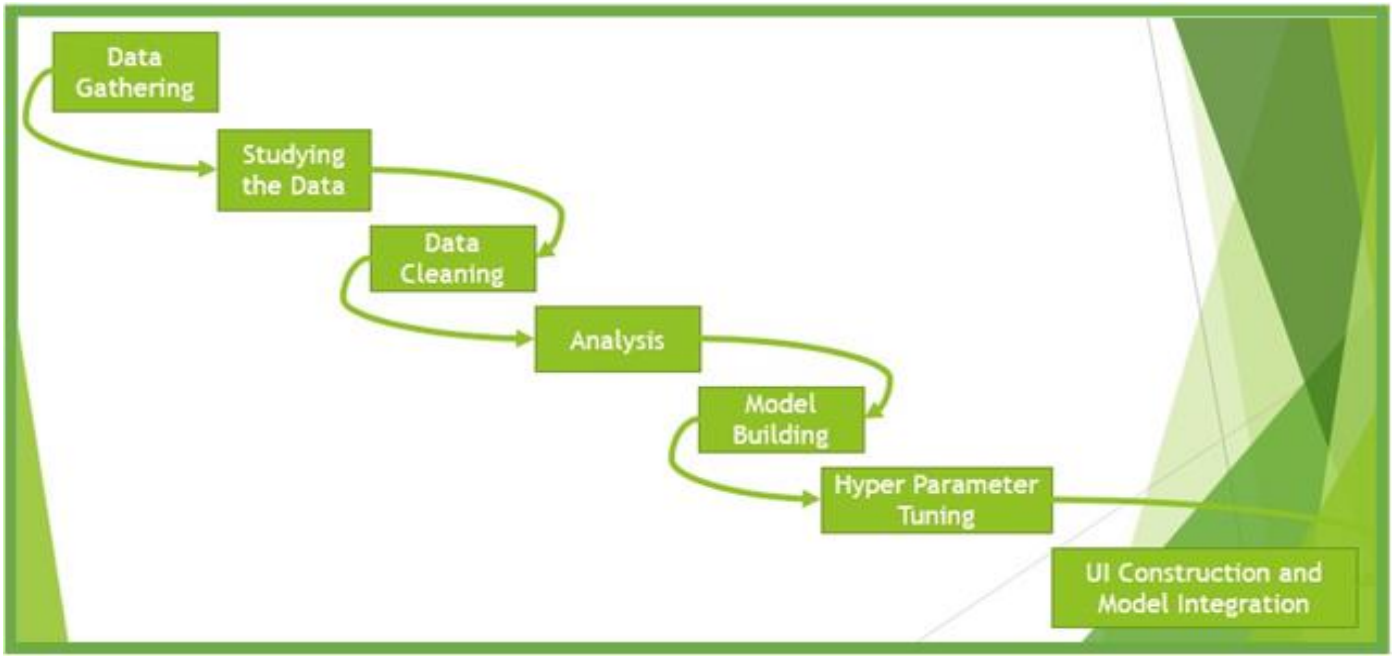

Figure 1 Workflow Diagram

3. Data Collection, Preprocessing and Cleaning

3.1 Data Collection:

We have collected the data form Kaggle. We have collected three different types of data from Kaggle – hourly traffic data, weather data, and holiday data.

3.2 Data Cleaning:

The data can have many irrelevant, missing parts, duplicates values, outliers, etc. To handle this part, data cleaning is done.

- **Missing Values:** It is very important to check for missing values in the dataset as they can impact our analyses. We have two options to deal with missing values and options are either you remove them or impute them. I have imputed the missing values using fillna() method.

```
#Impute NaN values in features/column 'holiday', 'holiday_type'
df_merge1['holiday'] = df_merge1['holiday'].fillna('No Special Event')
df_merge1['holiday_type'] = df_merge1['holiday_type'].fillna('No Holiday')
```

- **Duplicates Values:** It is often seen that in a dataset we have multiple rows that are same, so in order to remove duplicate values I have used drop_duplicates() method.

```
#Remove duplicate rows in the DataFrame.
df.drop_duplicates(inplace=True)
```

- **Outliers:** Outliers are the values that differ significantly from other observations in a dataset. Outliers are important to identify because they can influence the results of statistical analyses. Here, I have used Inter Quartile Ratio(IQR) method to impute outliers.

```
#Check outliers in the dataframe
column = ['Humidity', 'Temperature(C)', 'Wind Speed(Kmph)', 'Vehicles',
    'Precipitation(MM)']
```

```
for e in column:
    q1, q3 = df_merge1[e].quantile([0.25,0.75])
    iqr = q3 - q1
    min_valid_value = q1 - 1.5 * iqr
    max_valid_value = q3 + 1.5 * iqr
    print(e, q1, q3, iqr, min_valid_value, max_valid_value)

   #Impute outliers
   df_merge1.loc[df_merge1[e] < min_valid_value, e]=min_valid_value
  df_merge1.loc[df_merge1[e] > max_valid_value, e]=max_valid_value
```

3.3 Data Merging: We have used inner join to merge our datasets.

```
df_merge = pd.merge(df1, df2, how = "inner", on = ["DateTime", "DateTime"])
df_merge1 = pd.merge(df_merge, df3, how = "left", on = ["date", "date"])
```

4. Model Building

4.1 Train/Test split:

One important aspect of all machine learning models is to determine their accuracy. Now, in order to determine their accuracy, one can train the model using the given dataset and then predict the response values for the same dataset using that model and hence, find the accuracy of the model. A better option is to split our data into two parts: first one for training our machine learning model, and second one for testing our model.

- Split the dataset into two pieces: a training set and a testing set.
- Train the model on the training set.
- Test the model on the testing set, and evaluate how well our model did.

Advantages of train/test split:

- Model can be trained and tested on different data than the one used for training.
- Response values are known for the test dataset, hence predictions can be evaluated
- Testing accuracy is a better estimate than training accuracy of out-of-sample performance.

Machine learning consists of algorithms that can automate analytical model building. Using algorithms that iteratively learn from data, machine learning models facilitate computers to find hidden insights from Big Data without being explicitly programmed where to look.

We have used the following three algorithms to build predictive model

- Autoregressive Integrated Moving Average (ARIMA)
- XGBoost
- Long Short Term Memory (LSTM)

4.2 Autoregressive Integrated Moving Average (ARIMA)

ARIMA stands for Autoregressive Integrated Moving Average and it's a technique for time series analysis and for forecasting possible future values of a time series. Autoregressive modeling and Moving Average modeling are two different approaches to forecasting time series data. ARIMA integrates these two approaches, hence the name.

1. **AutoRegressive (AR) part:** This component models the relationship between an observation and several lagged observations (i.e., past values). It captures the influence of previous time steps on the current value.
2. **Integrated (I) part:** This involves differencing the data to make it stationary. Stationarity means that the statistical properties of the series, such as mean and variance, remain constant over time. Differencing is a technique used to remove trends or seasonality from the data.
3. **Moving Average (MA) part:** This component models the relationship between an observation and a residual error from a moving average model applied to lagged observations. It captures the influence of past forecast errors on the current value.

ARIMA models are generally denoted as ARIMA(p, d, q), where:

- p is the number of lag observations included in the model (the autoregressive part),
- d is the number of times that the raw observations are differenced (the integrated part),
- q is the size of the moving average window.

ARIMA models are widely used for time series forecasting in various fields like economics, finance, and meteorology. They are particularly useful when dealing with univariate data (a single time series) and when trends or seasonality need to be addressed.

4.2.1 ARIMA results for Junction 1:

```
#Apply auto_arima to find best order
model = pm.auto_arima(X_train_1, seasonal=False, m=12,trace = True)

#Create a model for junction 1
ar1 = pm.arima.ARIMA((2,1,4))
ar1.fit(X_train_1)
```

```
#Predict a model
pred1 = ar1.predict(X_test_1.shape[0])
```

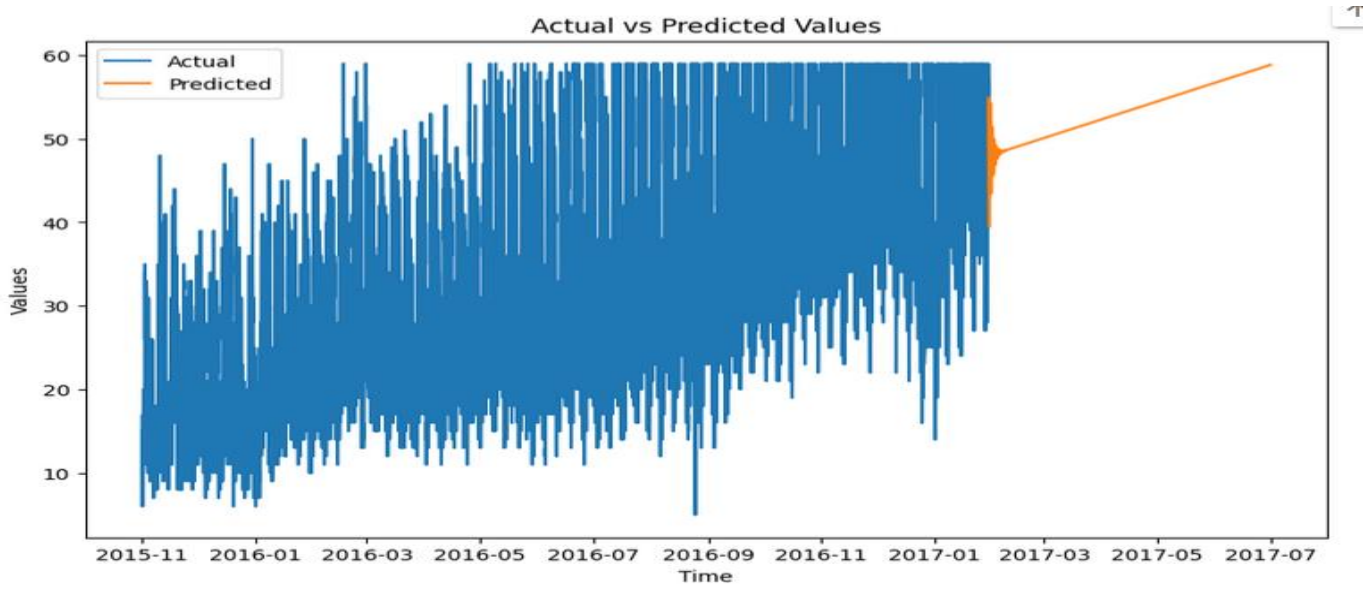

Figure 2 ARIMA Predictor vs Actual plot for Junction 1

From the graph we can say that at Junction 1 traffic is going to increase with time.

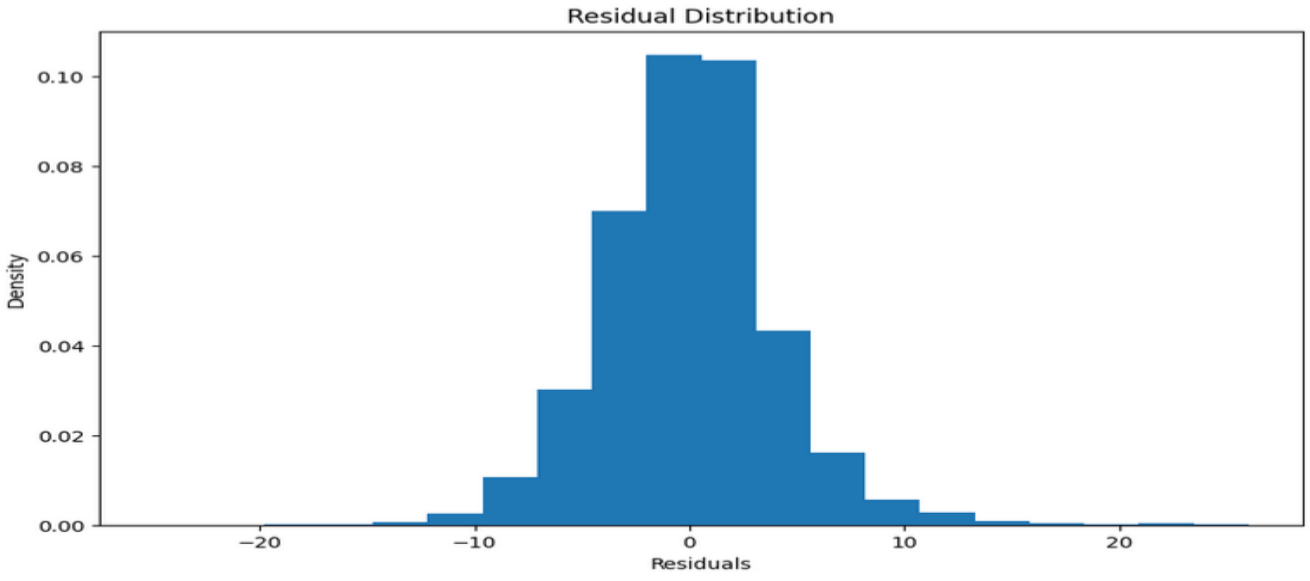

Figure 3 ARIMA Residual distribution for Junction 1

From the graph we can say residuals are normally distributed at Junction 1.

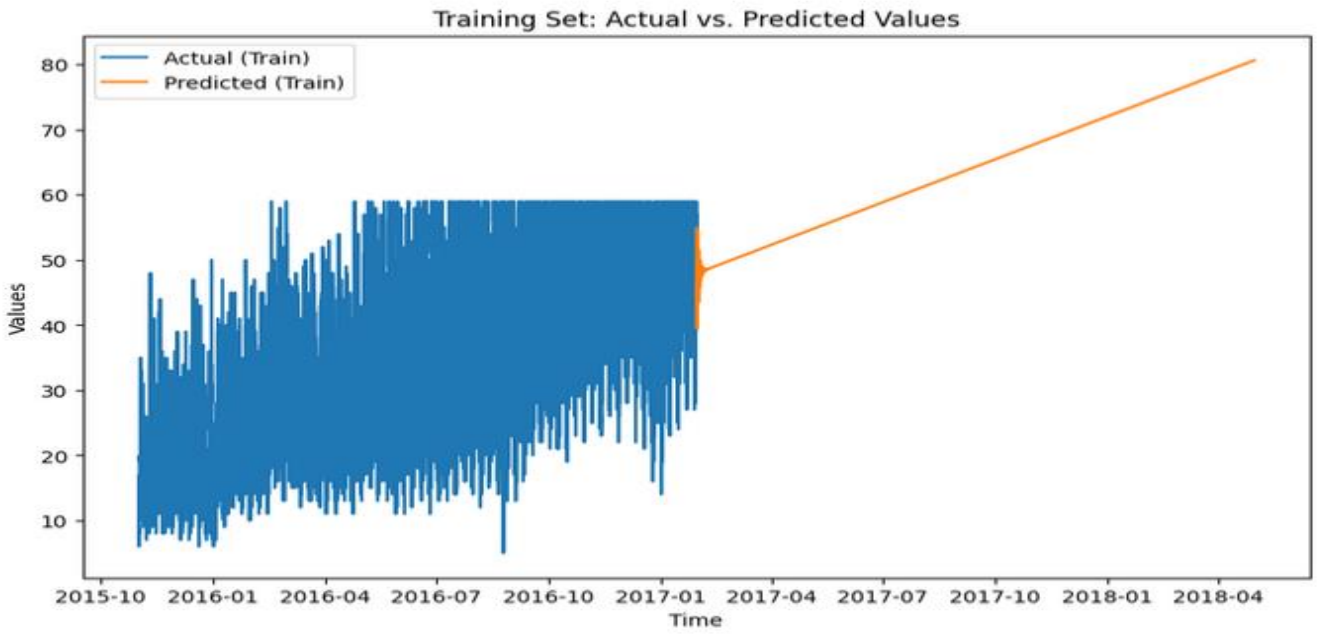

Figure 4 ARIMA Underfit/Overfit plot for Junction 1

From the graph we can say that at Junction 1 traffic is going to increase with time.

4.2.2 ARIMA results for Junction 2:

```
#Apply auto_arima to find best order
model = pm.auto_arima(X_train_2, seasonal=False, m=12,trace = True)

#Create a model for junction 2
ar2 = pm.arima.ARIMA((2,1,0))
ar2.fit(X_train_2)

#Predict a model
pred2 = ar2.predict(X_test_2.shape[0])
```

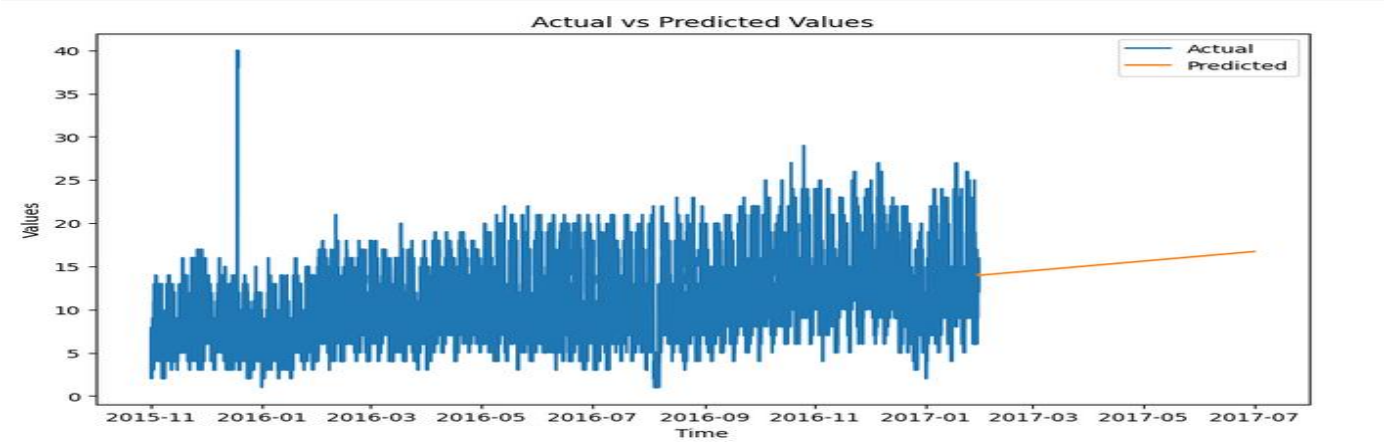

Figure 5 ARIMA Predictor vs Actual plot for Junction 2

From the graph we can say that it is hard to accurately predict whether traffic is going to increase or decrease at Junction 2 with time.

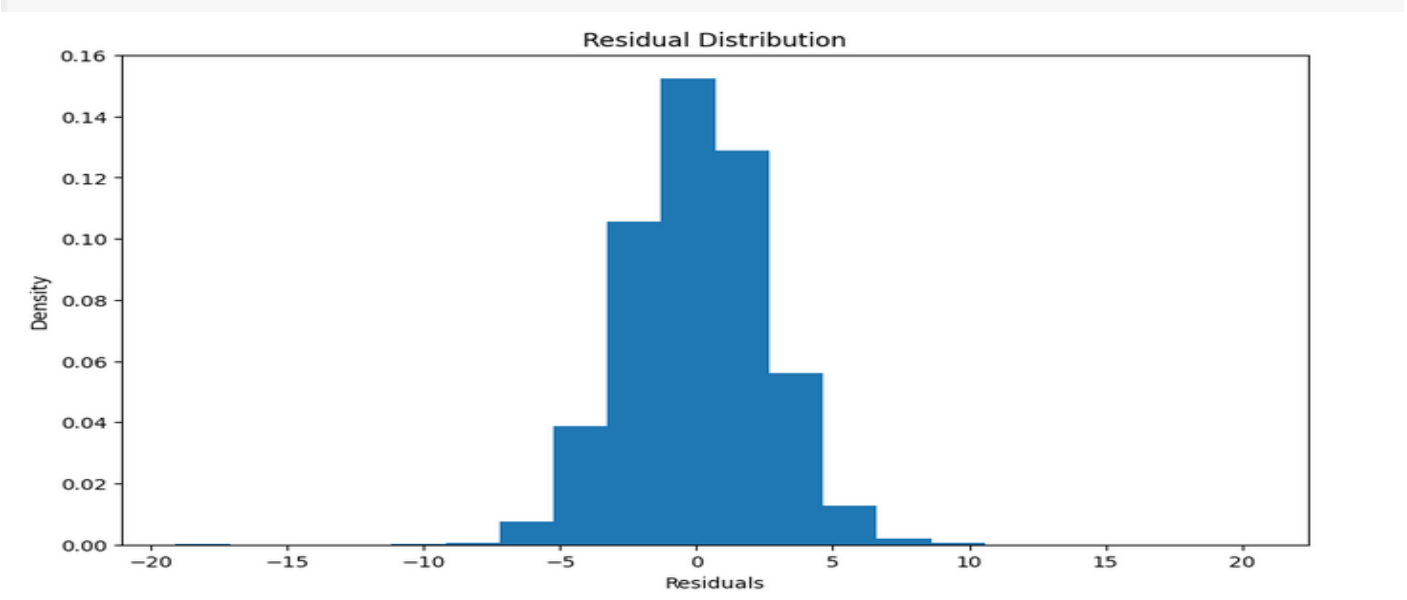

Figure 6 ARIMA Residual distribution for Junction 2

From the graph we can say residuals are normally distributed at Junction 2

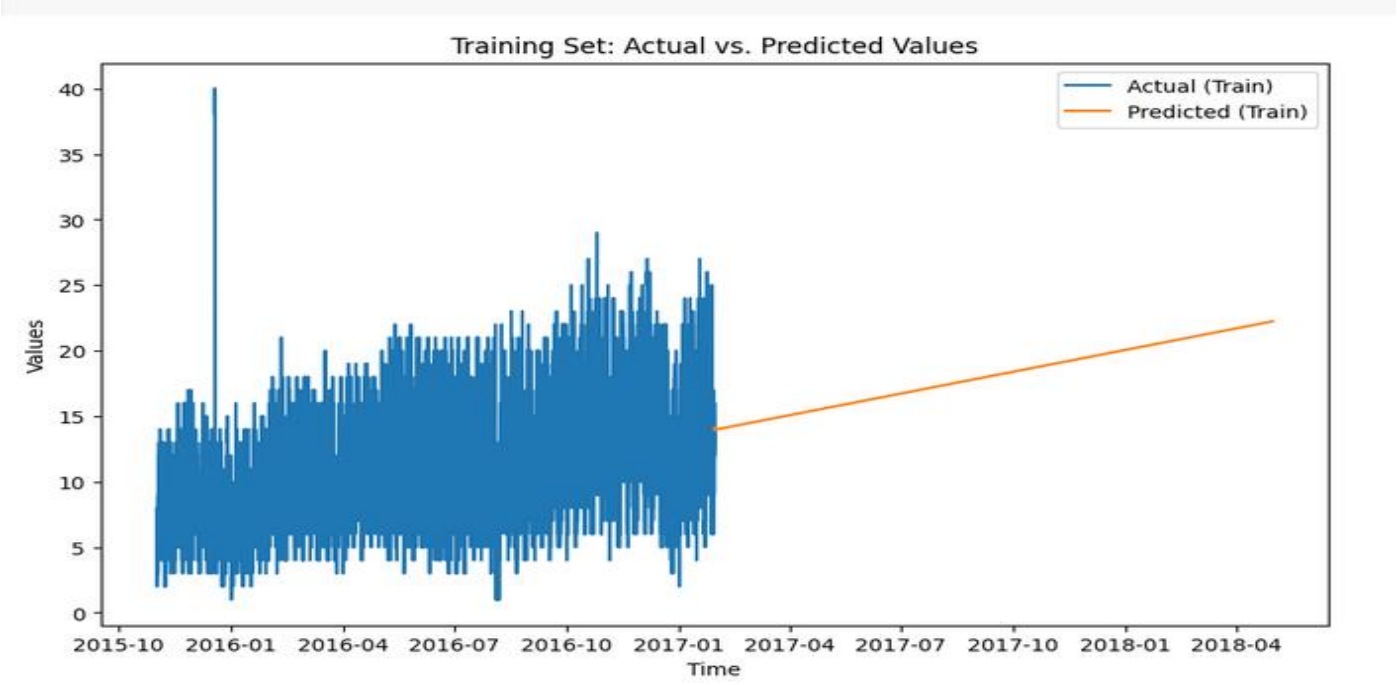

Figure 7 ARIMA Underfit/Overfit plot for Junction 2

From the graph we can say that it is hard to accurately predict whether traffic is going to increase or decrease at Junction 2 with time.

4.2.3 ARIMA results for Junction 3:

```
#Apply auto_arima to find best order
model = pm.auto_arima(X_train_3, seasonal=False, m=12,trace = True)

#Create a model for junction 3
ar3 = pm.arima.ARIMA((4,1,2))
ar3.fit(X_train_3)

#Predict a model
Pred3 = ar3.predict(X_test_3.shape[0])
```

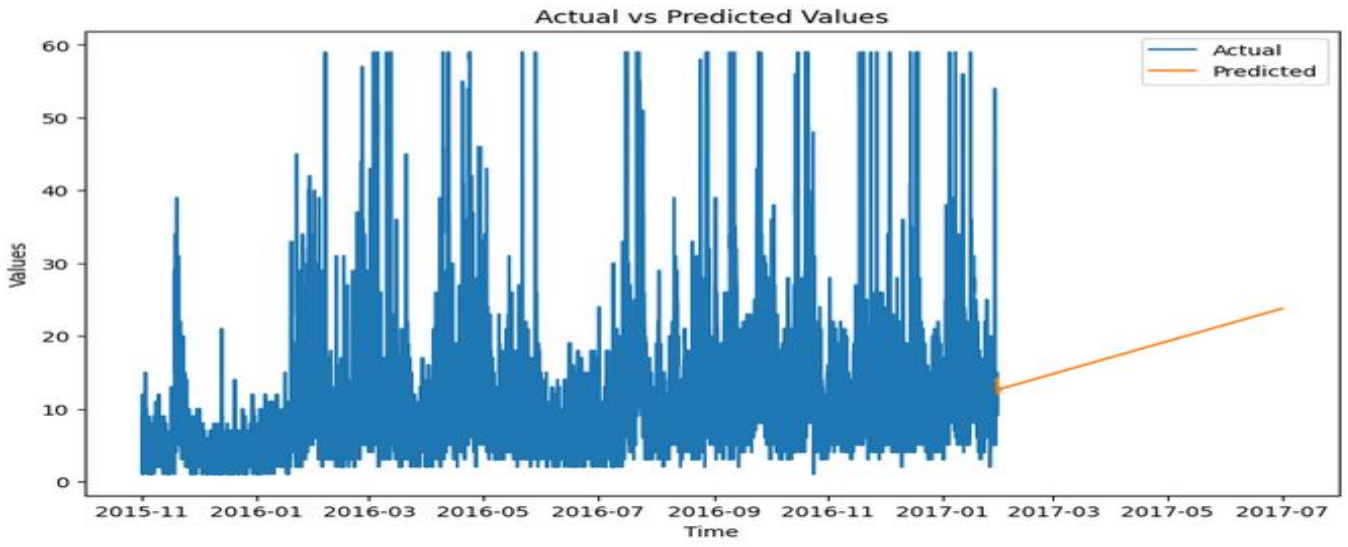

Figure 8 ARIMA Predictor vs Actual plot for Junction 3

From the graph we can say that at Junction 3 traffic is going to increase with time.

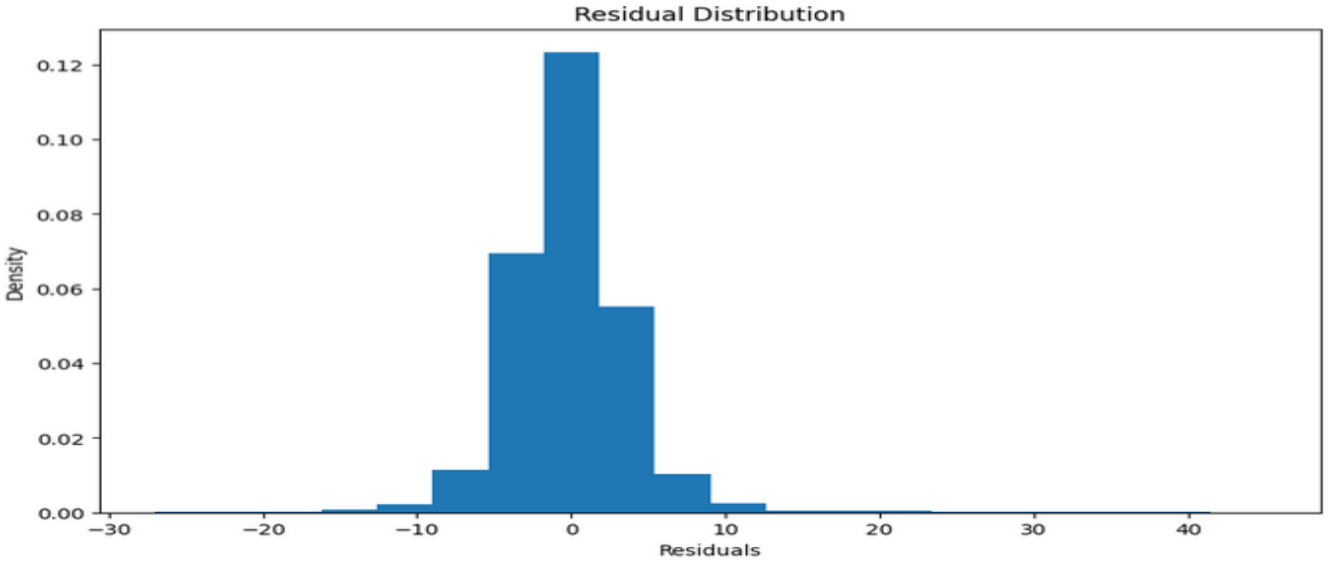

Figure 9 ARIMA Residual distribution for Junction 3

From the graph we can say residuals are normally distributed at Junction 3

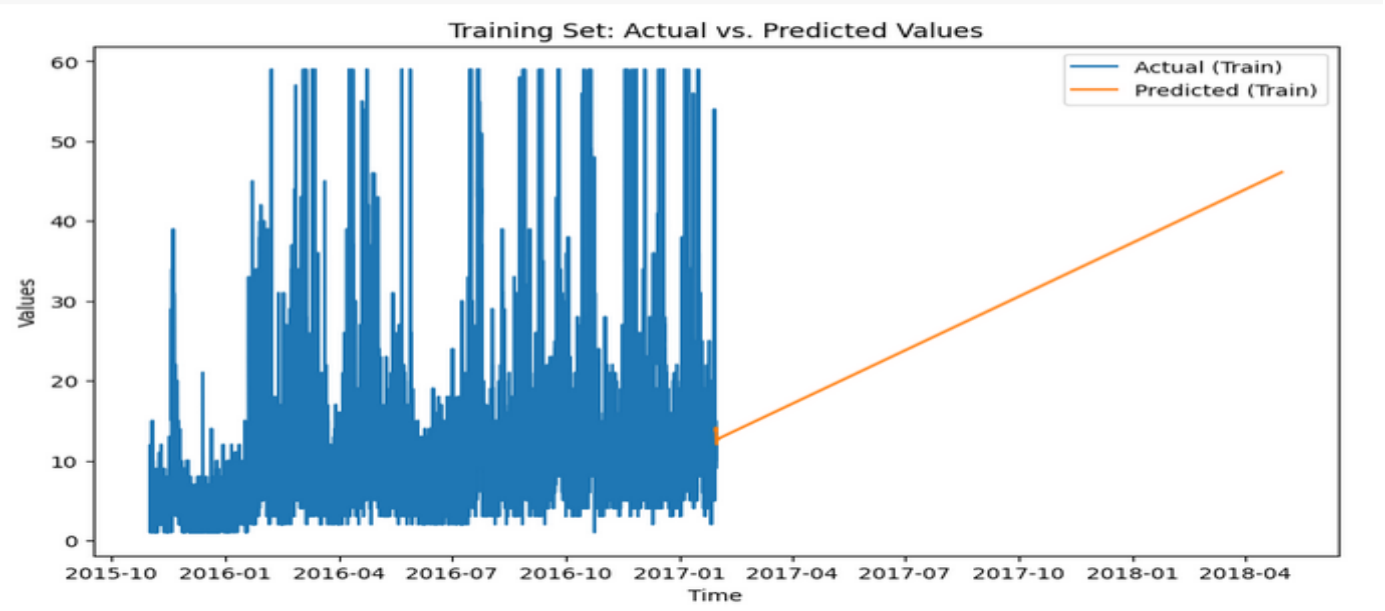

Figure 10 ARIMA Underfit/Overfit plot for Junction 3

From the graph we can say that at Junction 3 traffic is going to increase with time.

4.2.4 ARIMA results for Junction 4:

```
#Apply auto_arima to find best order
model = pm.auto_arima(X_train_4, seasonal=False, m=12,trace = True)

#Create a model for junction 3
ar4 = pm.arima.ARIMA((4,1,2))
ar4.fit(X_train_4)

#Predict a model
Pred4 = ar4.predict(X_test_4.shape[0])
```

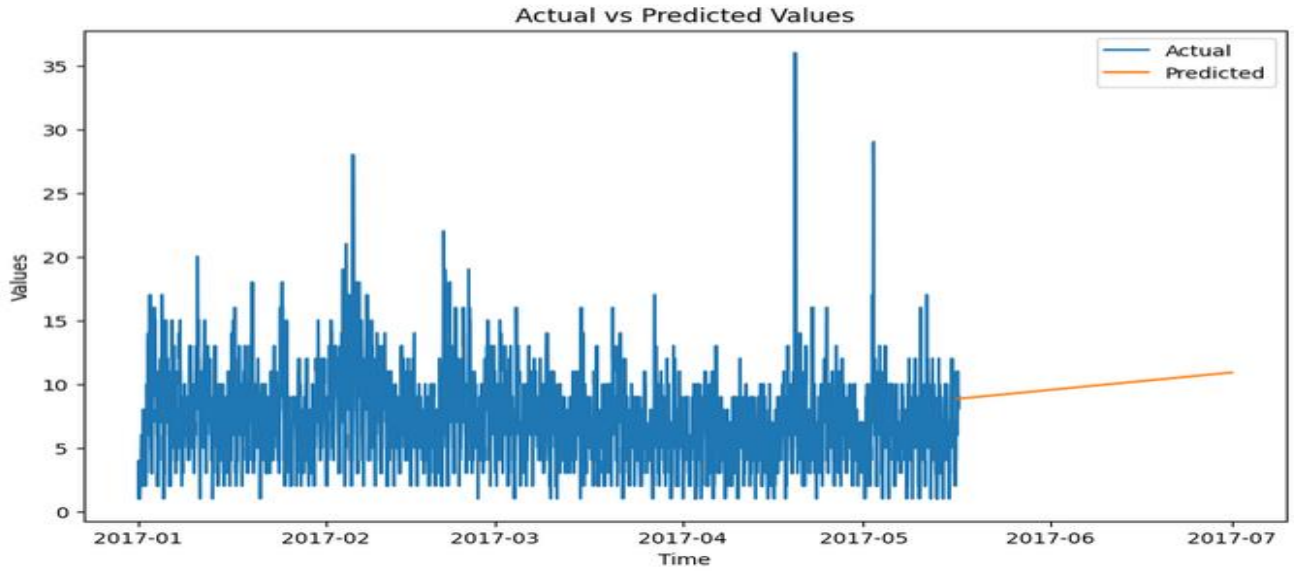

Figure 11 ARIMA Predictor vs Actual plot for Junction 4

From the graph we can say that it is hard to accurately predict whether traffic is going to increase or decrease at Junction 4 with time.

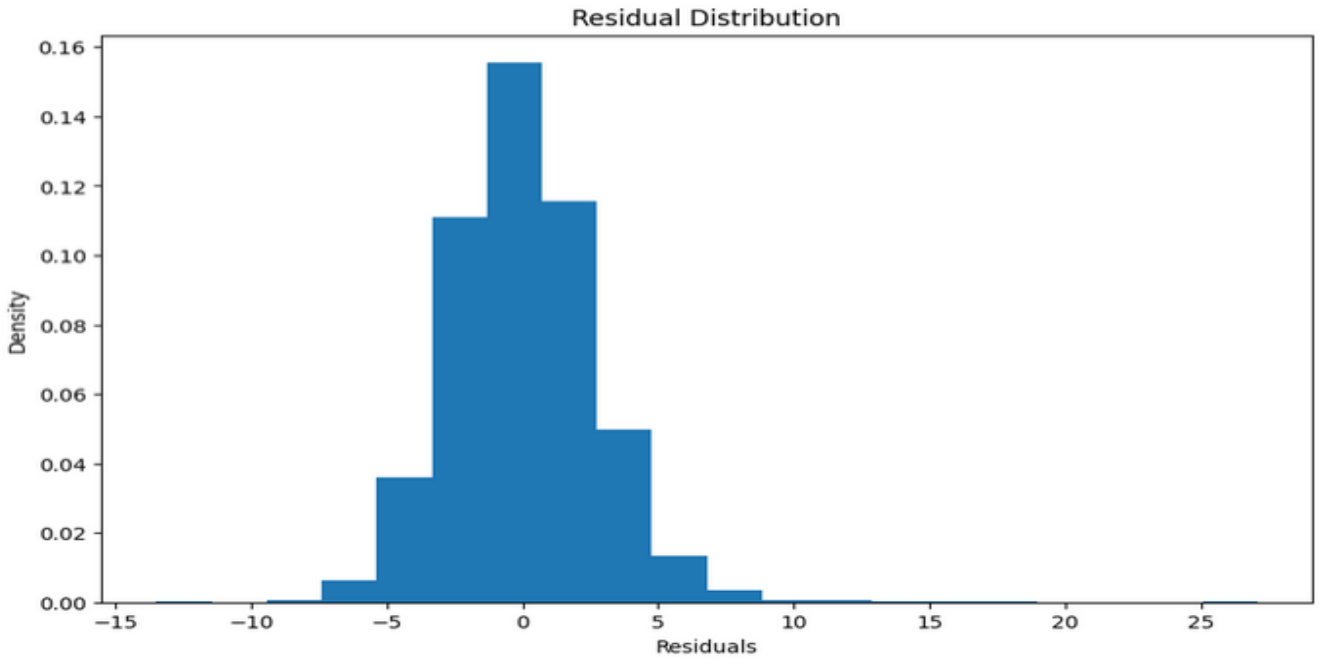

Figure 12 ARIMA Residual distribution for Junction 4

From the graph we can say residuals are normally distributed at Junction 4

4.3 XGBOOST

XGBoost, which stands for eXtreme Gradient Boosting, is a highly efficient and scalable machine learning library designed for supervised learning tasks. It's particularly well-suited for regression, classification, and ranking problems. XGBoost has gained popularity due to its performance and accuracy in various machine learning competitions and real-world applications.

Here's an overview of its key features and components:

1. **Gradient Boosting Framework:** XGBoost is based on the gradient boosting technique, which builds models in an additive fashion. It sequentially adds new models to correct errors made by previous models. Each new model is trained to minimize the residual errors of the ensemble, thereby improving overall performance.
2. **Regularization:** XGBoost incorporates L1 (Lasso) and L2 (Ridge) regularization techniques to prevent overfitting and improve model generalization. This is a notable feature compared to other gradient boosting methods.
3. **Tree-Based Learning:** XGBoost primarily uses decision trees as base learners. It builds an ensemble of trees where each tree tries to correct the errors of its predecessors.
4. **Parallelization and Scalability:** XGBoost supports parallel processing, making it faster and more scalable compared to traditional gradient boosting methods. This feature is especially useful for handling large datasets and complex models.
5. **Handling Missing Values:** XGBoost has built-in mechanisms to handle missing values in the dataset, which allows it to perform well even with incomplete data.
6. **Feature Importance:** The library provides tools to analyze the importance of different features in the model, which can help in understanding the model and improving feature selection.
7. **Flexibility:** XGBoost offers a range of hyperparameters that can be tuned to optimize model performance, giving users flexibility in configuring the model to suit their specific needs.

Overall, XGBoost is valued for its high performance, flexibility, and efficiency, making it a popular choice among data scientists and machine learning practitioners for a variety of tasks.

```
# Define the parameter grid
param_grid = {
    'n_estimators': [10,20,30,40,50],
```

```
    'learning_rate': [0.02,0.05,0.08,0.1,0.12],
    'max_depth': [3,4,5,6,7],
}
xgb_model = xgb.XGBRegressor()
#Apply Grid Search
grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid,
                    scoring='neg_mean_squared_error',cv=5, verbose=2,return_train_score=True)

#Train a model
grid_search.fit(X_train, Y_train)
best_params = grid_search.best_params_
best_xgb_model = grid_search.best_estimator_
xgc=best_xgb_model
xgc.fit(X_train, Y_train)

#Predict a model
Y_pred = xgc.predict(X_test)
```

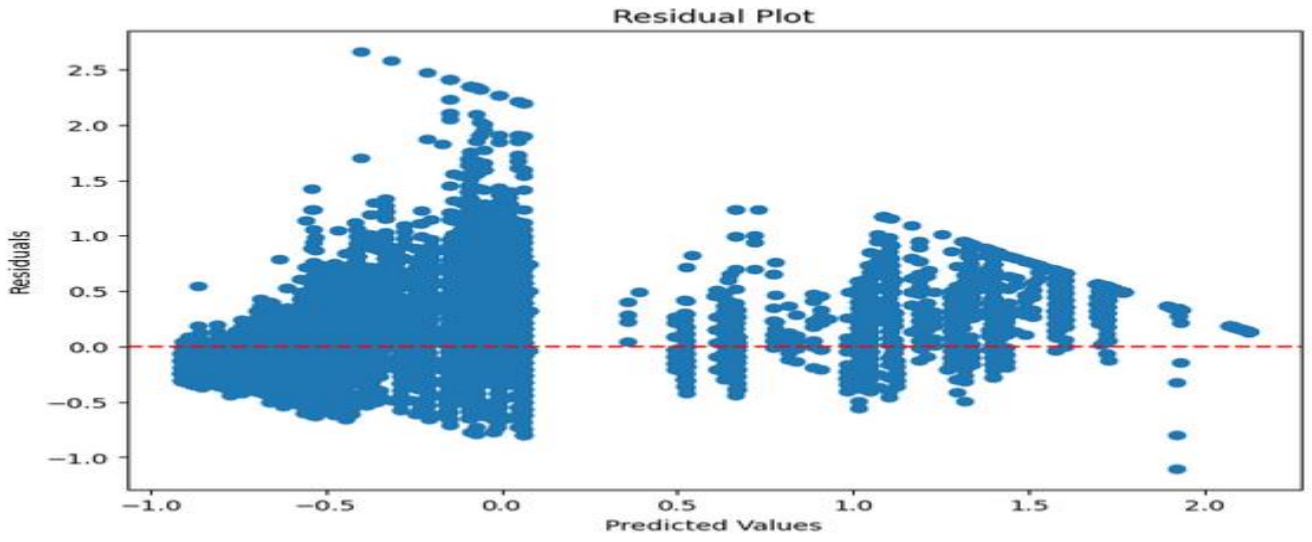

Figure 13 XGBOOST Residuals plot for all Junctions

Graph is Funnel Shape so there is Heteroscedasticity in residuals. Which indicates that the model's errors are not consistent across the range of predictions. XGBoost, like many tree-based models, is generally robust to heteroscedasticity.

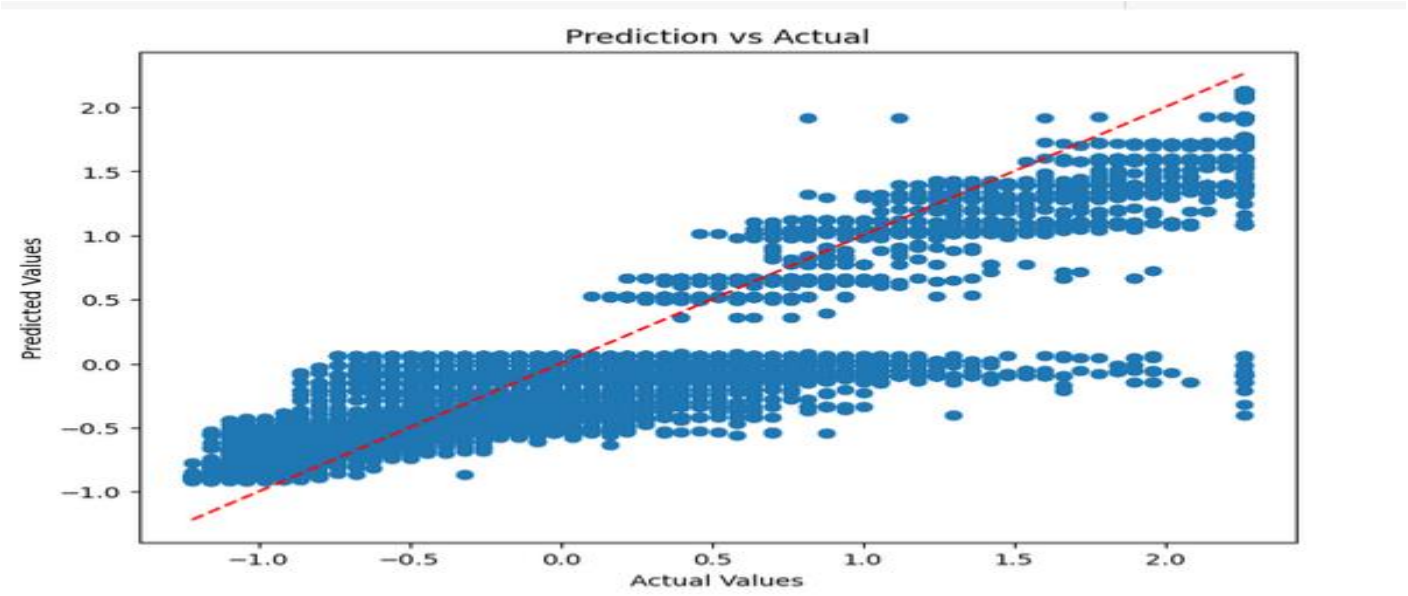

Figure 14 XGBOOST Predictor vs Actual plot for all Junction

Almost 70% points clustered tightly around the diagonal line, indicating that the predictions are very close to the actual values with minimal deviations.

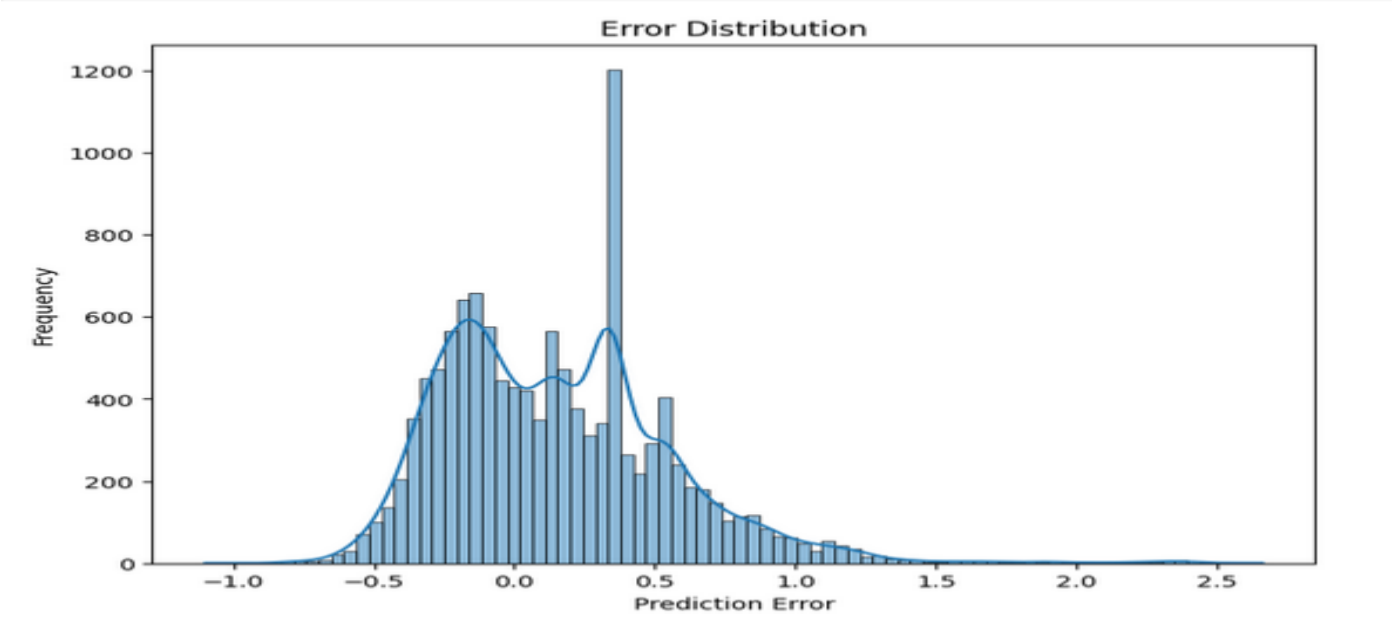

Figure 15 XGBOOST Error distribution for all Junction

Distribution centered around zero with a relatively small spread, indicating that model's errors are small and unbiased.

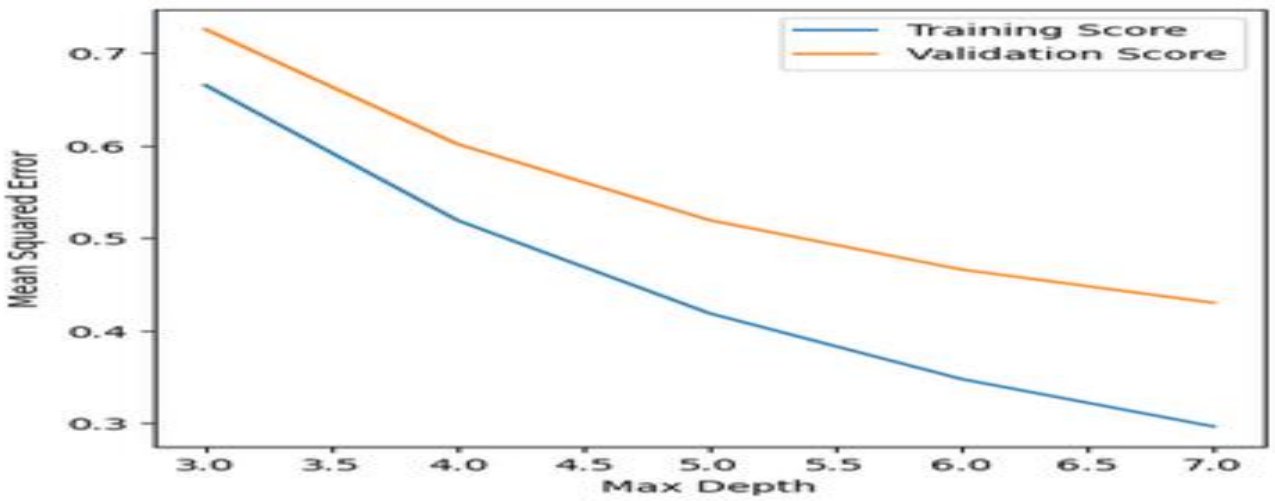

Figure 16 XGBOOST Underfit/Overfit plot for all Junction

Here both training and validation scores are decreasing with increase in depth. means optimum fit. (Overfitting: If the training score consistently decreases. While the validation score plateaus or increases with increasing complexity.)

4.4 Long Short Term Memory (LSTM)

Long Short Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture designed to address some of the limitations of traditional RNNs, particularly the problems of long-term dependencies and vanishing/exploding gradients. LSTMs are particularly well-suited for tasks involving sequences and time-series data, such as natural language processing, speech recognition, and financial forecasting.

Key Components of LSTM:

1. **Cell State**: This is the memory of the LSTM unit, which carries information across long sequences. The cell state allows the network to remember important information for long periods.
2. **Gates**: LSTMs use gates to control the flow of information into and out of the cell state. These gates are:
    - **Forget Gate**: Decides which information from the cell state should be discarded. It uses a sigmoid activation function to output a value between 0 (completely forget) and 1 (completely retain).
    - **Input Gate**: Determines which new information should be added to the cell state. It involves a sigmoid activation function for the gate and a tanh activation function to create candidate values for the new information.

- **Output Gate**: Controls the output from the cell state to the next layer in the network. It uses a sigmoid activation function to decide which parts of the cell state should be output and a tanh function to scale the output.

3. **Cell State Update**: The cell state is updated based on the output of the forget and input gates. This allows the LSTM to keep or discard information selectively and effectively.

4.4.1 LSTM for Junction 1

```
#Define the model

model1 = Sequential()

model1.add(LSTM(2, activation='relu',return_sequences=True, input_shape=(12,1)))

#First LSTM layer with 4 units, ReLU activation, and returning sequences for the next layer

model1.add(LSTM(1, activation='relu'))

#Second LSTM layer with 2 units and ReLU activation, no need to return sequences as this is the last LSTM layer

model1.add(Dense(1))

history1 = model1.fit(X_train, Y_train, validation_split=0.1, epochs =100, batch_size =10)

#Predict on the test set

Y_pred= model1.predict(X_test_rnn)
```

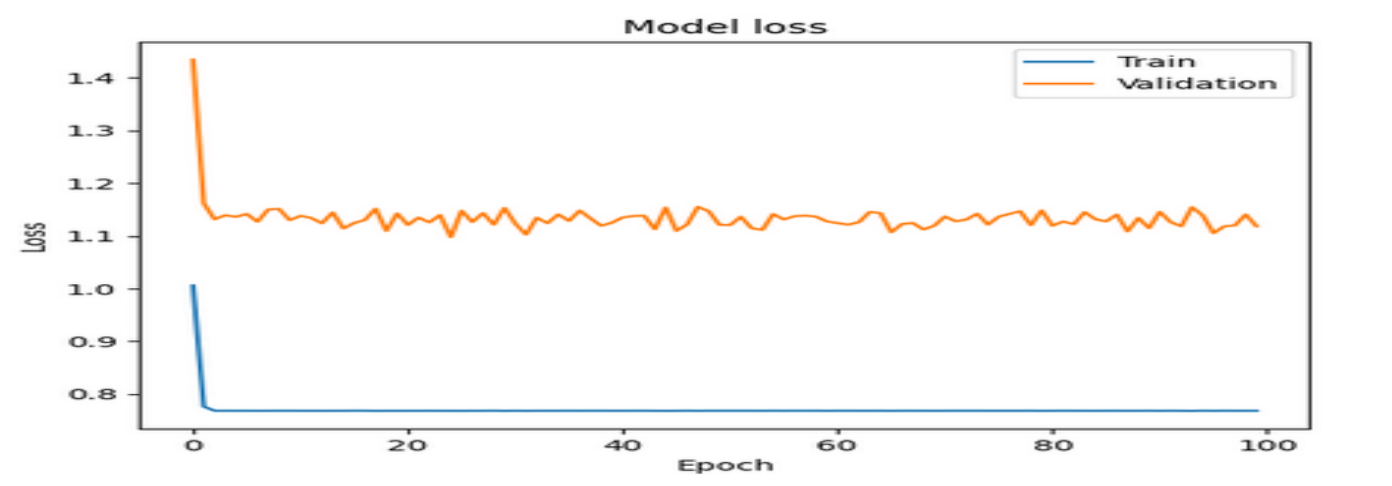

Figure 17 LSTM Overfit/Underfit plot for Junction 1

Optimum fitting: If the training score & validation score consistently decreases with increasing complexity.

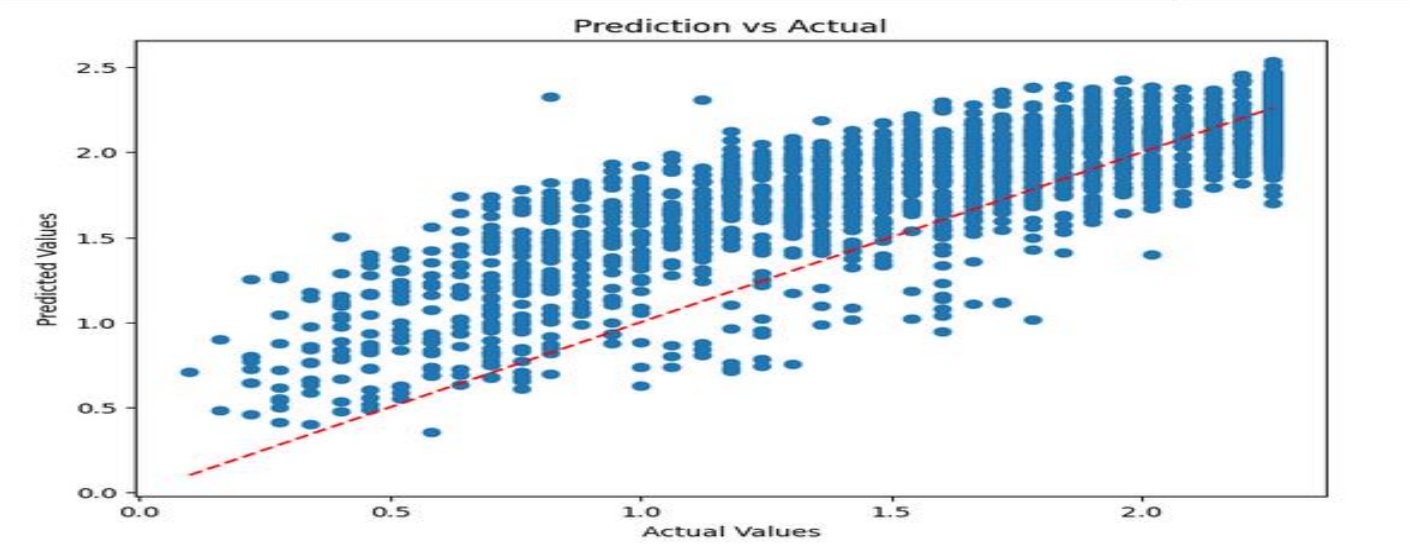

Figure 18 LSTM Predictor vs Actual plot for Junction 1

Almost 60% points clustered tightly around the diagonal line, indicating that the predictions are very close to the actual values with minimal deviations.

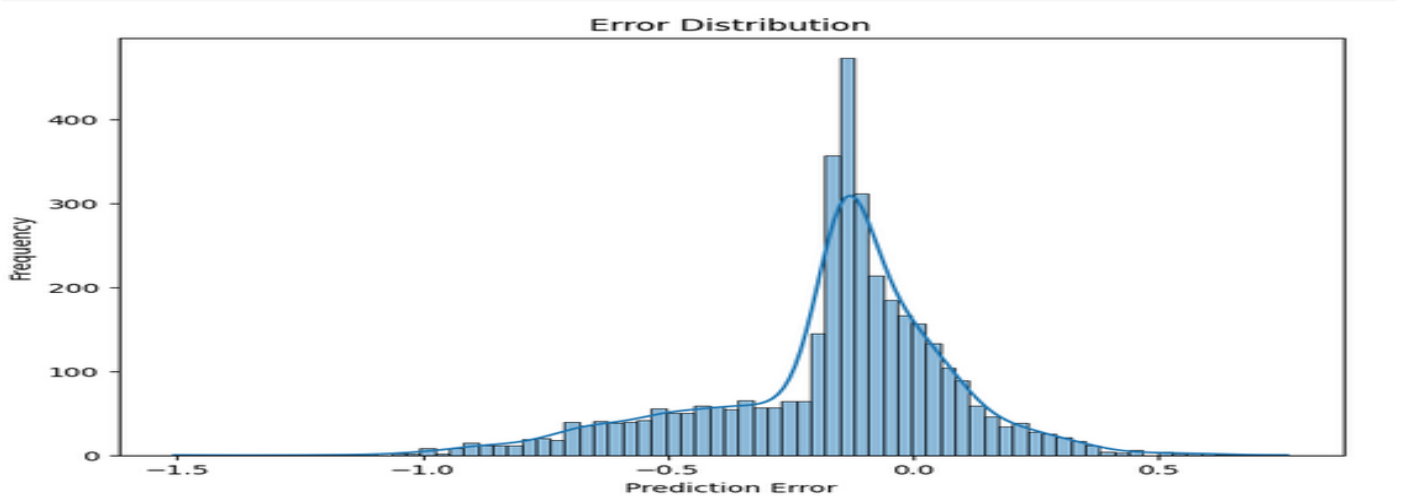

Figure 19 LSTM Error distribution plot for Junction 1

In junction 1, error distribution is right skewed and model has negative biased.

4.4.2 LSTM for Junction 2

Define the model

```
model2 = Sequential()

model2.add(LSTM(2, activation='relu',return_sequences=True, input_shape=(12,1)))

# First LSTM layer with 4 units, ReLU activation, and returning sequences for the next layer

model2.add(LSTM(1, activation='relu'))

# Second LSTM layer with 2 units and ReLU activation, no need to return sequences as this is the last LSTM layer

model2.add(Dense(1))

# Output layer with a single neuron for regression

model2.compile(optimizer='adam', loss='mse')

# Compile the model with Adam optimizer and mean squared error loss

history2 = model2.fit(X_train, Y_train,validation_split=0.1, epochs =100, batch_size =10)

# Predict on the test set

Y_pred= model2.predict(X_test_rnn)

Y_pred = Y_pred.flatten()
```

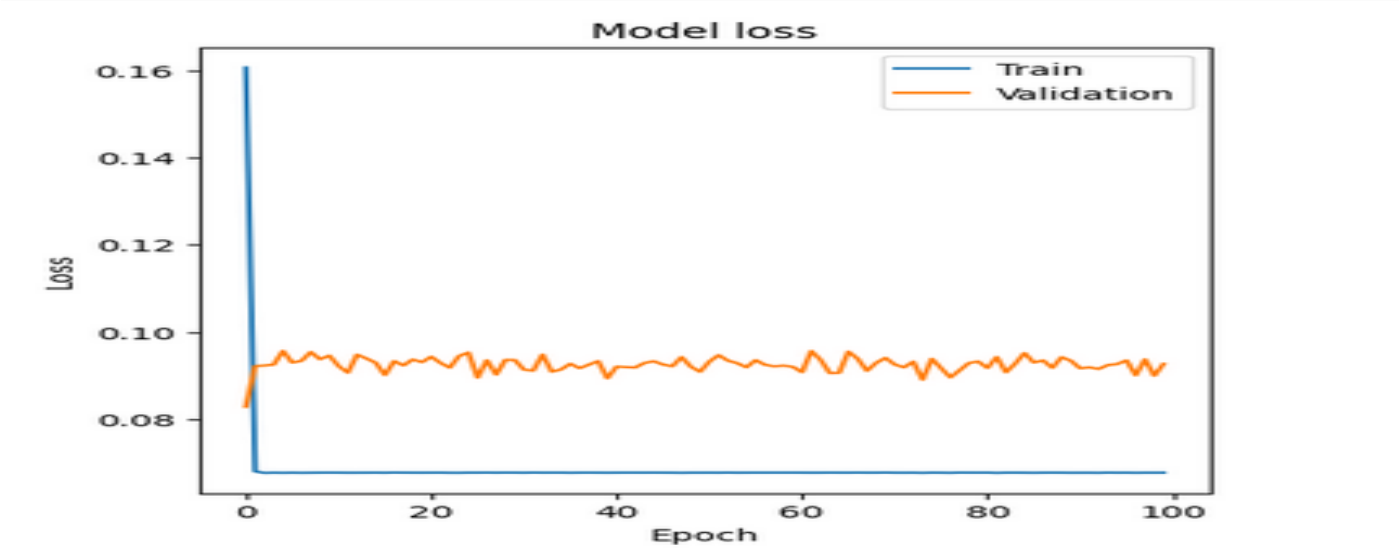

Figure 20 LSTM Overfit/Underfit plot for Junction 2

Optimum fitting: If the training score & validation score consistently decreases with increasing complexity.

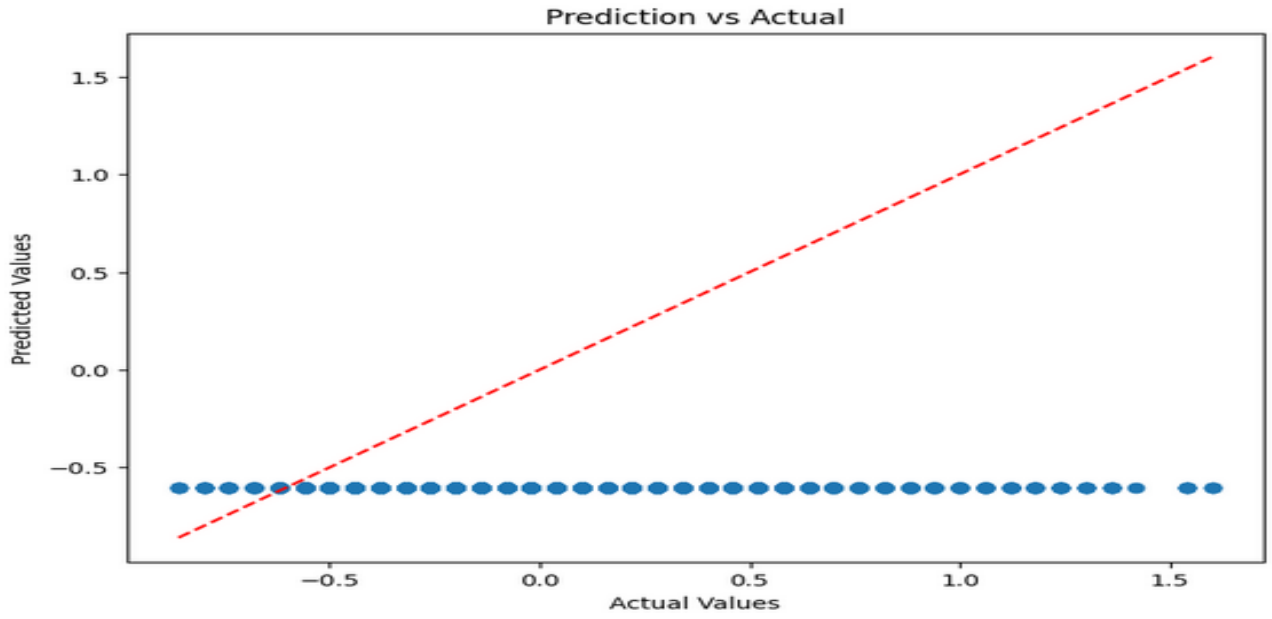

Figure 21 LSTM Predictor vs Actual plot for Junction 2

Almost 60% points clustered tightly around the diagonal line, indicating that the predictions are very close to the actual values with minimal deviations.

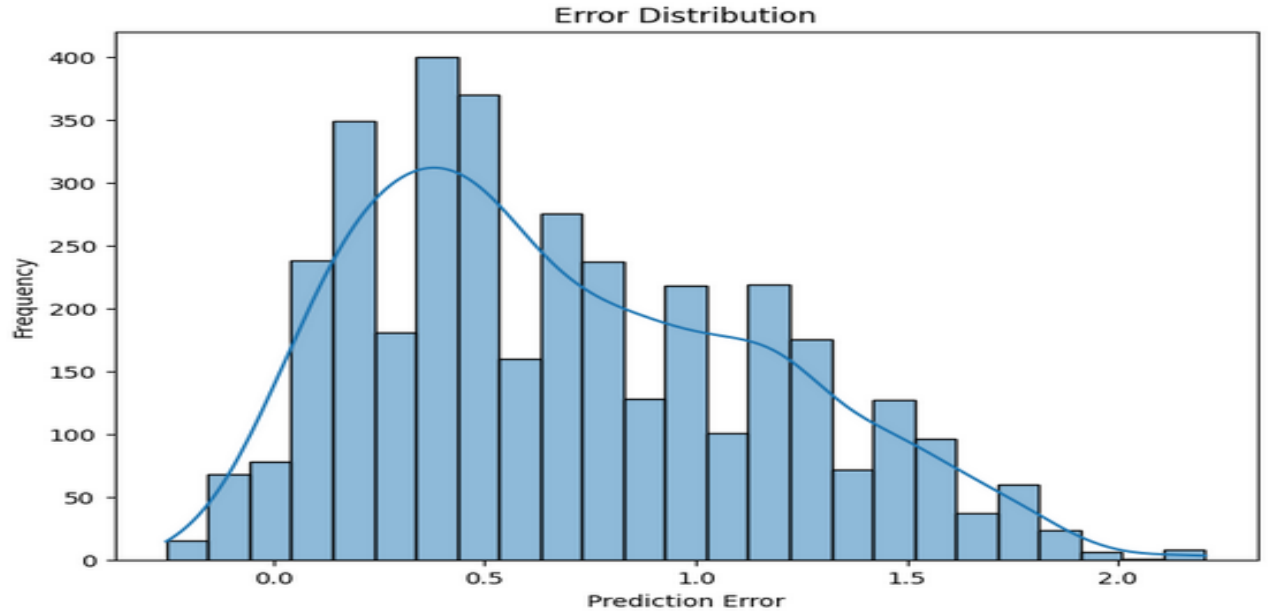

Figure 22 LSTM Error distribution plot for Junction 2

In junction 2, error distribution is left skewed and model has positive biased.

4.4.3 LSTM for Junction 3

```
# Define the model

model3 = Sequential()

model3.add(LSTM(2, activation='relu',return_sequences=True, input_shape=(12,1)))

# First LSTM layer with 4 units, ReLU activation, and returning sequences for the next layer

model3.add(LSTM(1, activation='relu'))

# Second LSTM layer with 2 units and ReLU activation, no need to return sequences as this is the last LSTM layer

model3.add(Dense(1))

# Output layer with a single neuron for regression

model3.compile(optimizer='adam', loss='mse')

# Compile the model with Adam optimizer and mean squared error loss

history3 = model3.fit(X_train, Y_train, validation_split=0.1, epochs =100, batch_size =10)

# Predict on the test set

Y_pred= model3.predict(X_test_rnn)

Y_pred = Y_pred.flatten()
```

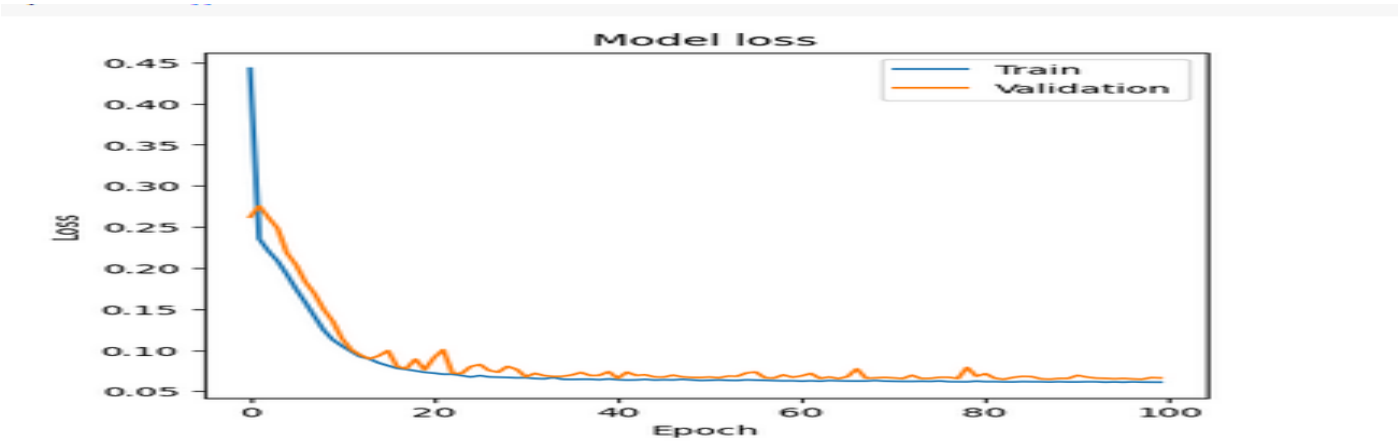

Figure 23 LSTM Overfit/Underfit plot for Junction 3

Optimum fitting: If the training score & validation score consistently decreases with increasing complexity.

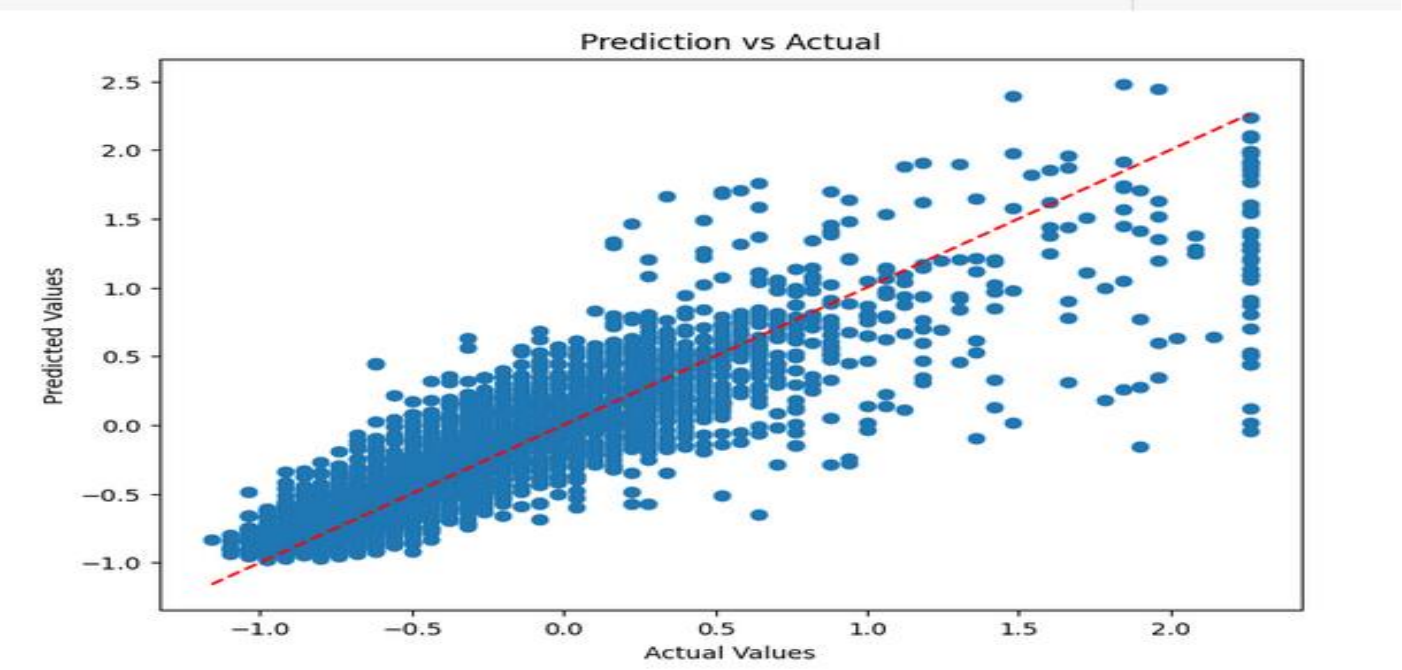

Figure 24 LSTM Predictor vs Actual plot for Junction 3

Almost 70% points clustered tightly around the diagonal line, indicating that the predictions are very close to the actual values with minimal deviations.

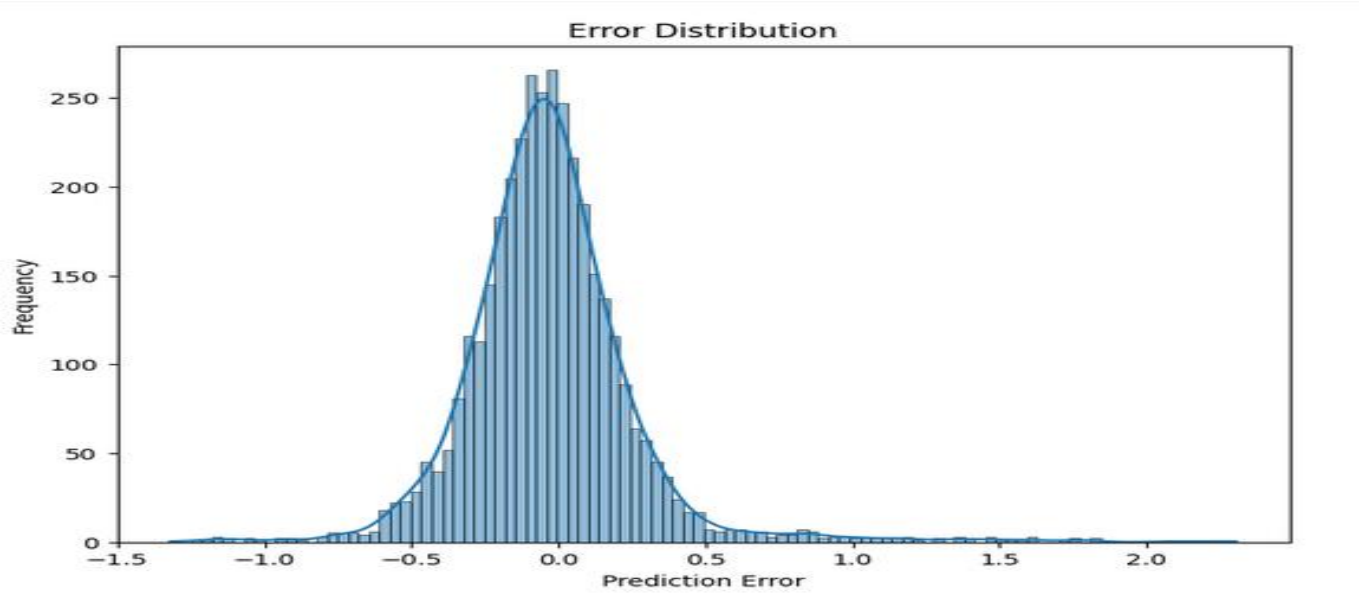

Figure 25 LSTM Error distribution plot for Junction 3

In junction 3, error distribution is normal and model is not biased.

4.4.4 LSTM for Junction 4

```
# Define the model

model4 = Sequential()

model4.add(LSTM(2, activation='relu',return_sequences=True, input_shape=(12,1)))

# First LSTM layer with 4 units, ReLU activation, and returning sequences for the next layer

model4.add(LSTM(1, activation='relu'))

# Second LSTM layer with 2 units and ReLU activation, no need to return sequences as this is the last LSTM layer

model4.add(Dense(1))

# Output layer with a single neuron for regression

model4.compile(optimizer='adam', loss='mse')

# Compile the model with Adam optimizer and mean squared error loss

history4 = model4.fit(X_train, Y_train, validation_split=0.1,epochs =100, batch_size =10)

# Predict on the test set

Y_pred= model4.predict(X_test_rnn)

Y_pred = Y_pred.flatten()
```

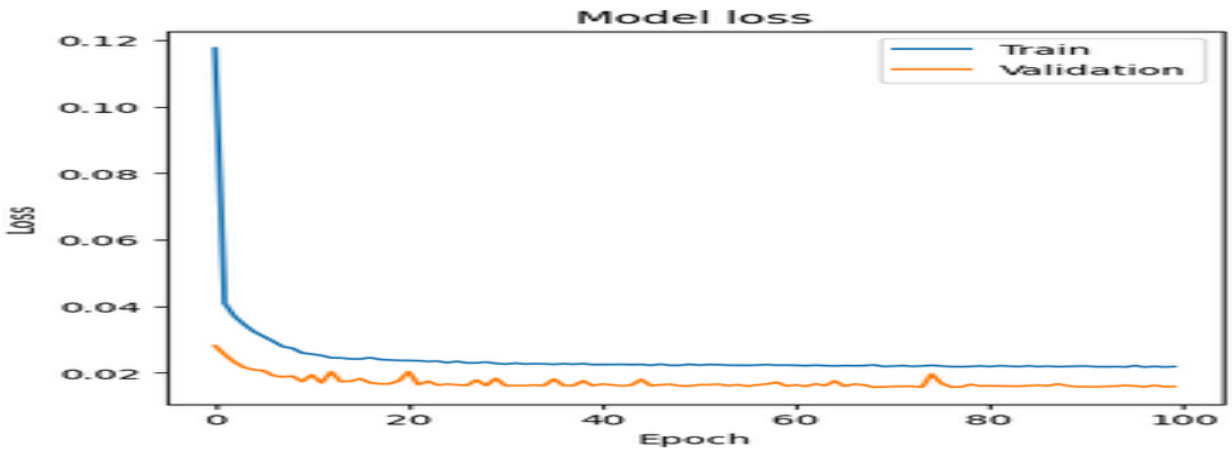

Figure 26 LSTM Overfit/Underfit plot for Junction 4

Optimum fitting: If the training score & validation score consistently decreases with increasing complexity.

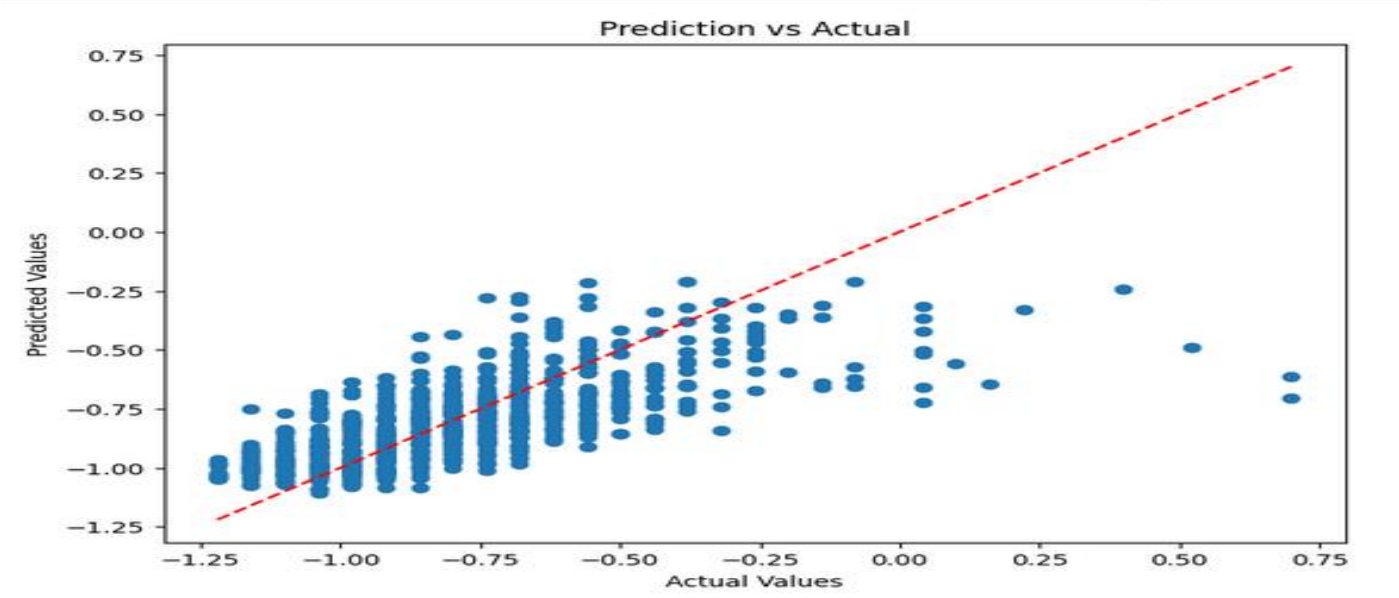

Figure 27 LSTM Predictor vs Actual plot for Junction 4

Almost 60% points clustered tightly around the diagonal line, indicating that the predictions are very close to the actual values with minimal deviations.

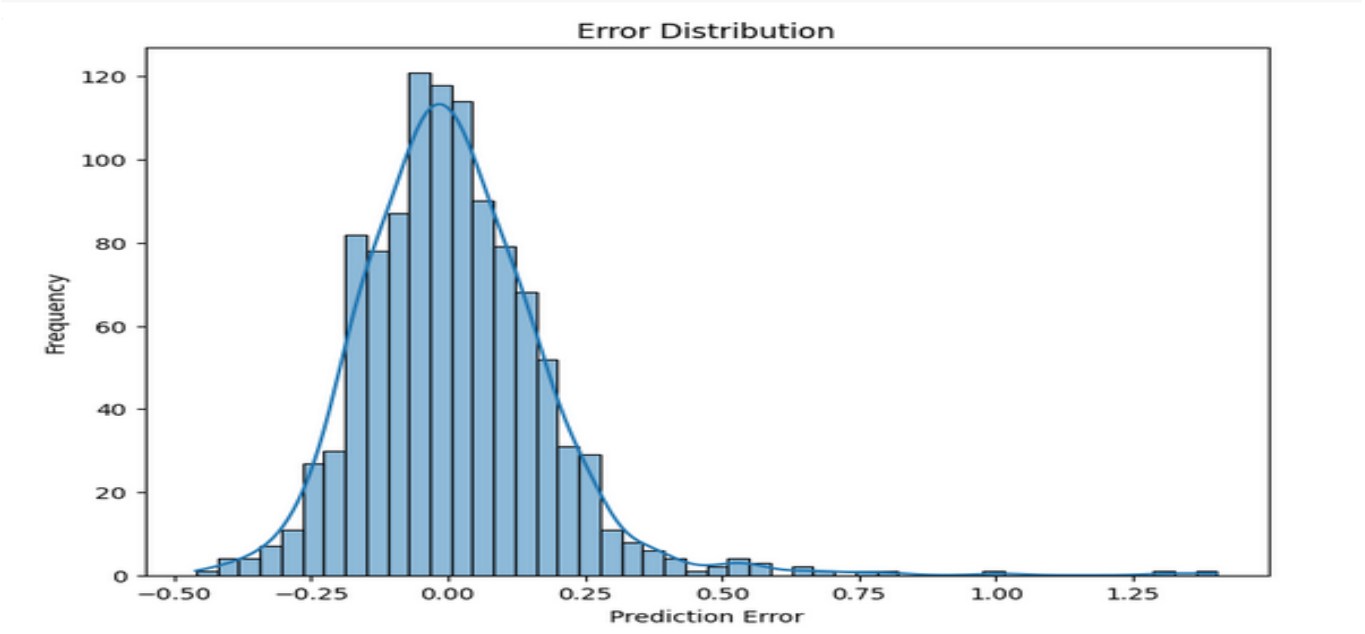

Figure 28 LSTM Error distribution plot for Junction 4

In junction 4, error distribution is normal and model is not biased.

5. Model Comparison

	ARIMA	**XGBOOST**	**LSTM**
Junction 1 **(**Root Mean Square Error**)**	9.310842254787437	0.4295610689498854	1.18353000324633
Junction 2 **(**Root Mean Square Error**)**	10.77006572554436	0.4295610689498854	0.8431044580636126
Junction 3 **(**Root Mean Square Error**)**	10.60344037673732	0.4295610689498854	0.2917749323522624
Junction 4 **(**Root Mean Square Error**)**	4.445092648047436	0.4295610689498854	0.17124771522392784

From the above table we can clearly see root mean square error is minimum in XGBOOST model, so we can say that for our analysis XGBOOST is the best model.

6. Conclusion

To accurately forecast traffic at various junctions we have applied ARIMA, XGBOOST and LSTM at various junctions. At junction 4 and junction 3 LSTM has given the efficient result. At junction 2 and junction 1 XGBOOST has given the efficient result. From ARIMA we can predict that at junction 1 traffic is going to increase with time. From LSTM we can predict that at junction 2 almost 60% points clustered tightly around the diagonal line, indicating that the predictions are very close to the actual values with minimal deviations. From LSTM we can predict at junction 3 almost 70% points clustered tightly around the diagonal line, indicating that the predictions are very close to the actual values with minimal deviations. From LSTM we can predict at junction 4 almost 60% points clustered tightly around the diagonal line, indicating that the predictions are very close to the actual values with minimal deviations. We have found that overall XGBOOST has given the most efficient result. Further we have found that at all junctions traffic achieve its peak at 19th, 20th and 21st hour after that traffic starts declining. From our analyses we have found that traffic is increasing yearly at all junctions per se but at junction 1 it is increased at very fast rate with respect to other junctions.

7. References

1. https://www.kaggle.com/
2. https://www.geeksforgeeks.org/
3. Machine Learning using Python by Manaranjan Pradhan and U Dinesh Kumar
4. https://stackoverflow.com/
5. https://en.wikipedia.org/wiki/Long_short-term_memor

IACSD

Hourly forecast of traffic at various junctions