SML CSE-342

TRAFFIC FLOW PREDICTION & MANAGEMENT

PROJECT PRESENTATION

Presented By ADITYA UPADHYAY
ARPAN VERMA

14TH MAY 2024

STATISTICAL MACHINE LEARNING(SML) CSE-342

AGENDA

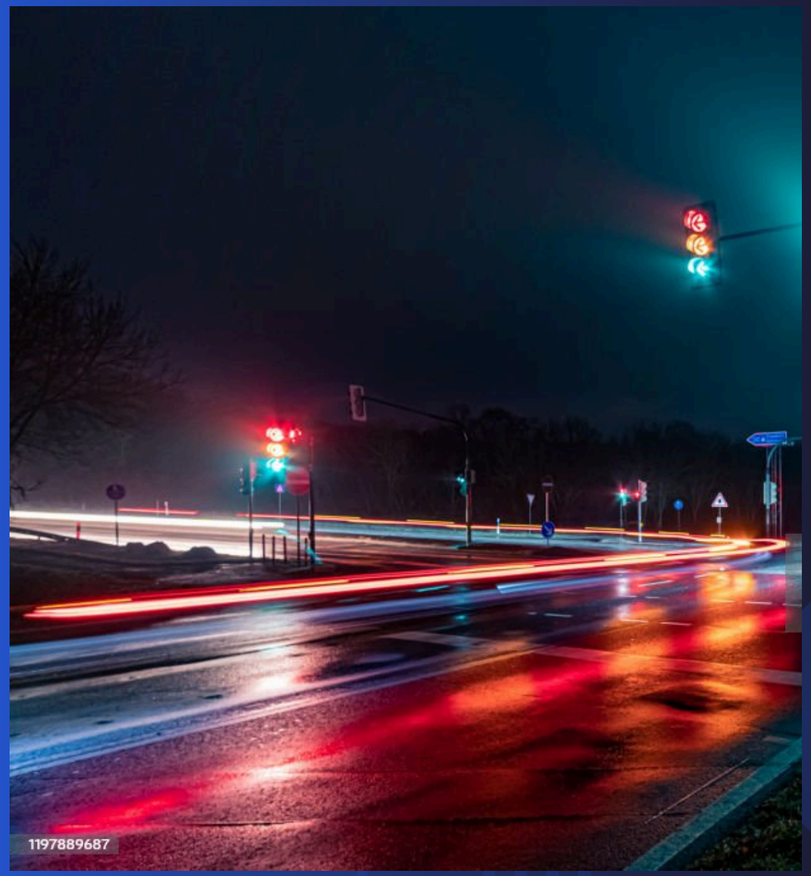
AGENDA

Background of the Study

Traffic congestion is a significant issue in urban areas, leading to wasted time, increased pollution, and economic losses.

Back to Agenda

PROBLEM STATEMENT



Prediction of Traffic on Intersections.

Devising a traffic management plan, by changing the timers of Traffic Signals.

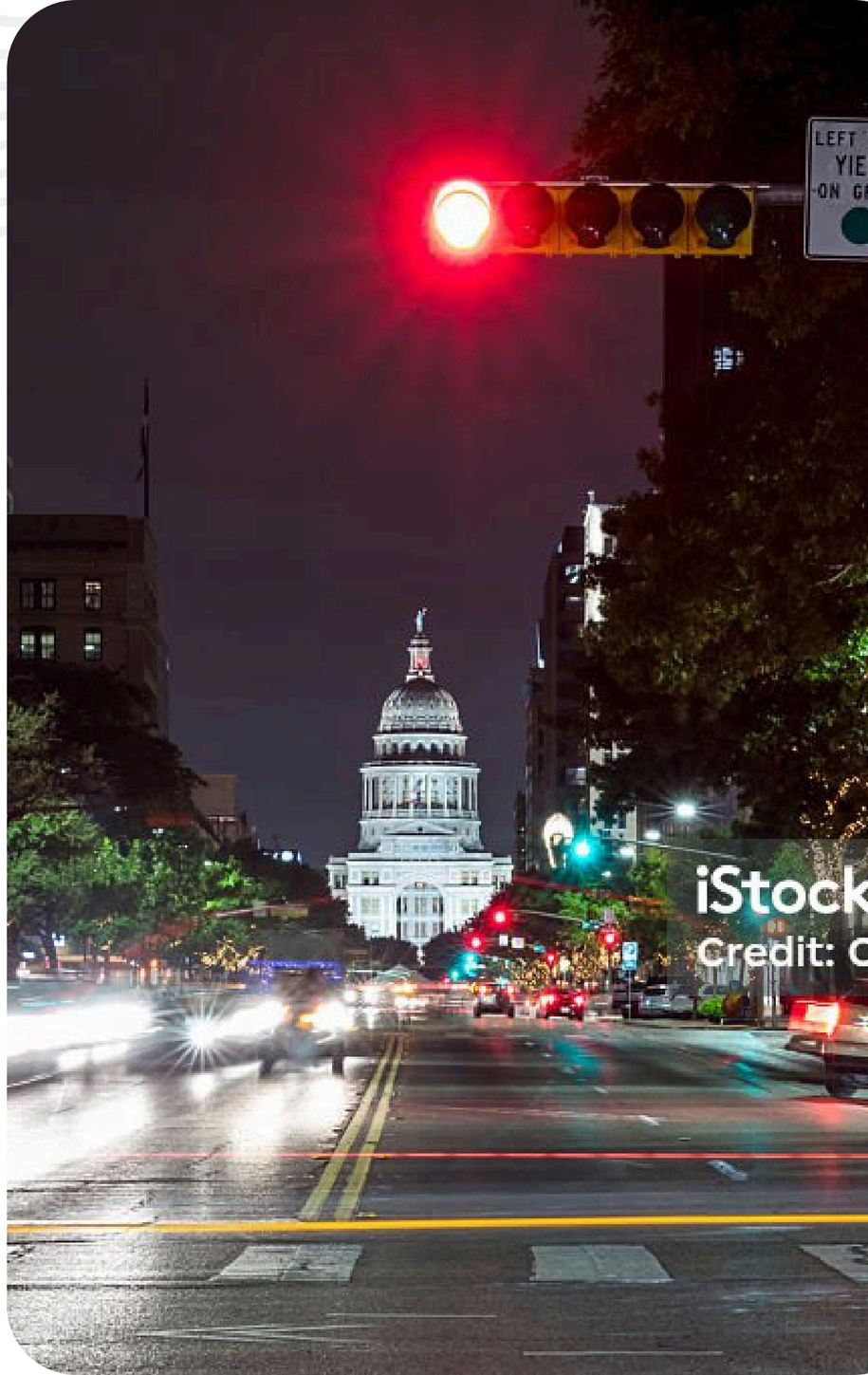
Back to Agenda

Back to Agenda

INTRODUCTION

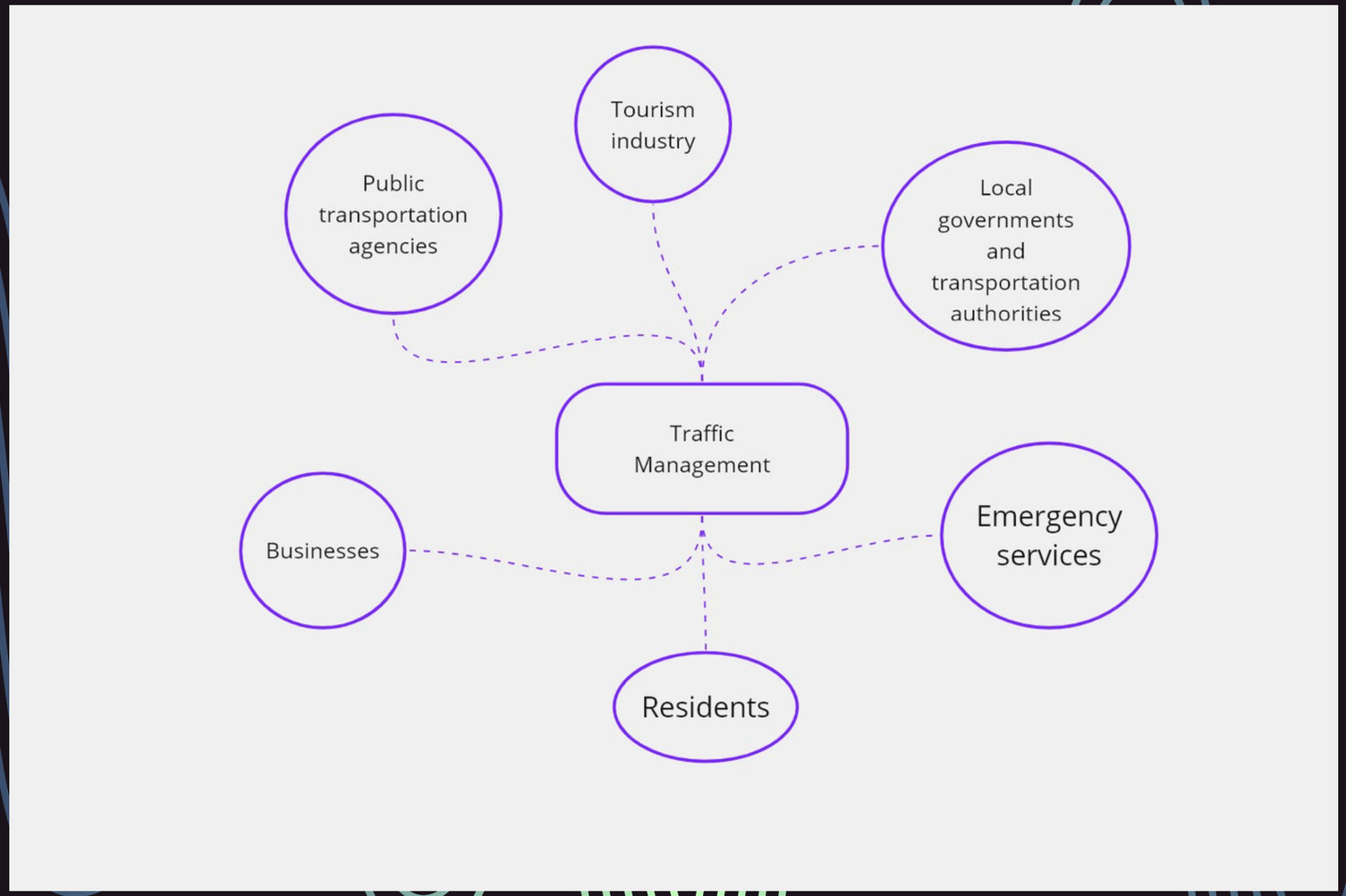
In this project we have utilized and compared various AI/ML models to predict traffic at road intersections.

We then have devised an efficient plan to manage the Signal Timers.



Back to Agenda

STAKEHOLDERS



DATASET

The dataset used for this project has been taken from Kaggle.

This dataset contains 48.1k (48120) observations of the number of vehicles each hour in four different Lanes:
1) DateTime
2) Junction/Lane
3) Vehicles
4) ID

https://www.kaggle.com/datasets/fedesoriano/traffic-prediction-dataset

About this file

Number of Instances: 48120

Number of Attributes: 4

DateTime Datetime in hourly frequency	# Junction Number of the junction	# Vehicles Number of vehicles that hour	ID Unique id
2015-11-01 — 2017-07-01	1 — 4	1 — 180	20.2b — 20.2b
2015-11-01 00:00:00	1	15	20151101001
2015-11-01 01:00:00	1	13	20151101011
2015-11-01 02:00:00	1	10	20151101021
2015-11-01 03:00:00	1	7	20151101031
2015-11-01 04:00:00	1	9	20151101041

Back to Agenda

Back to Agenda

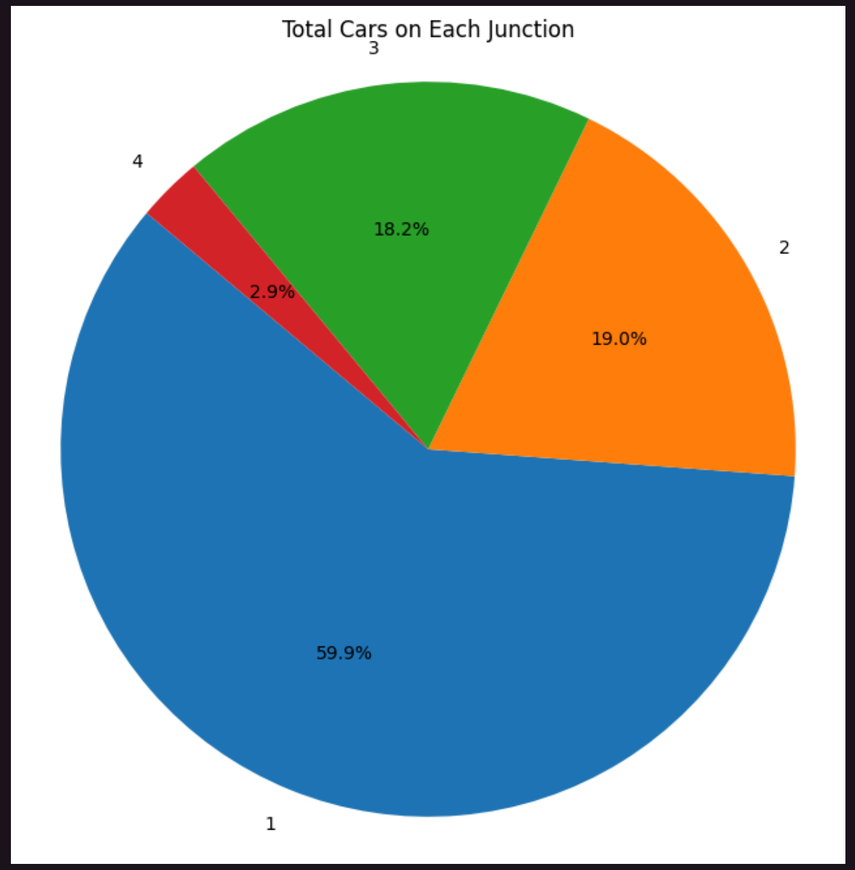
PREPROCESSING

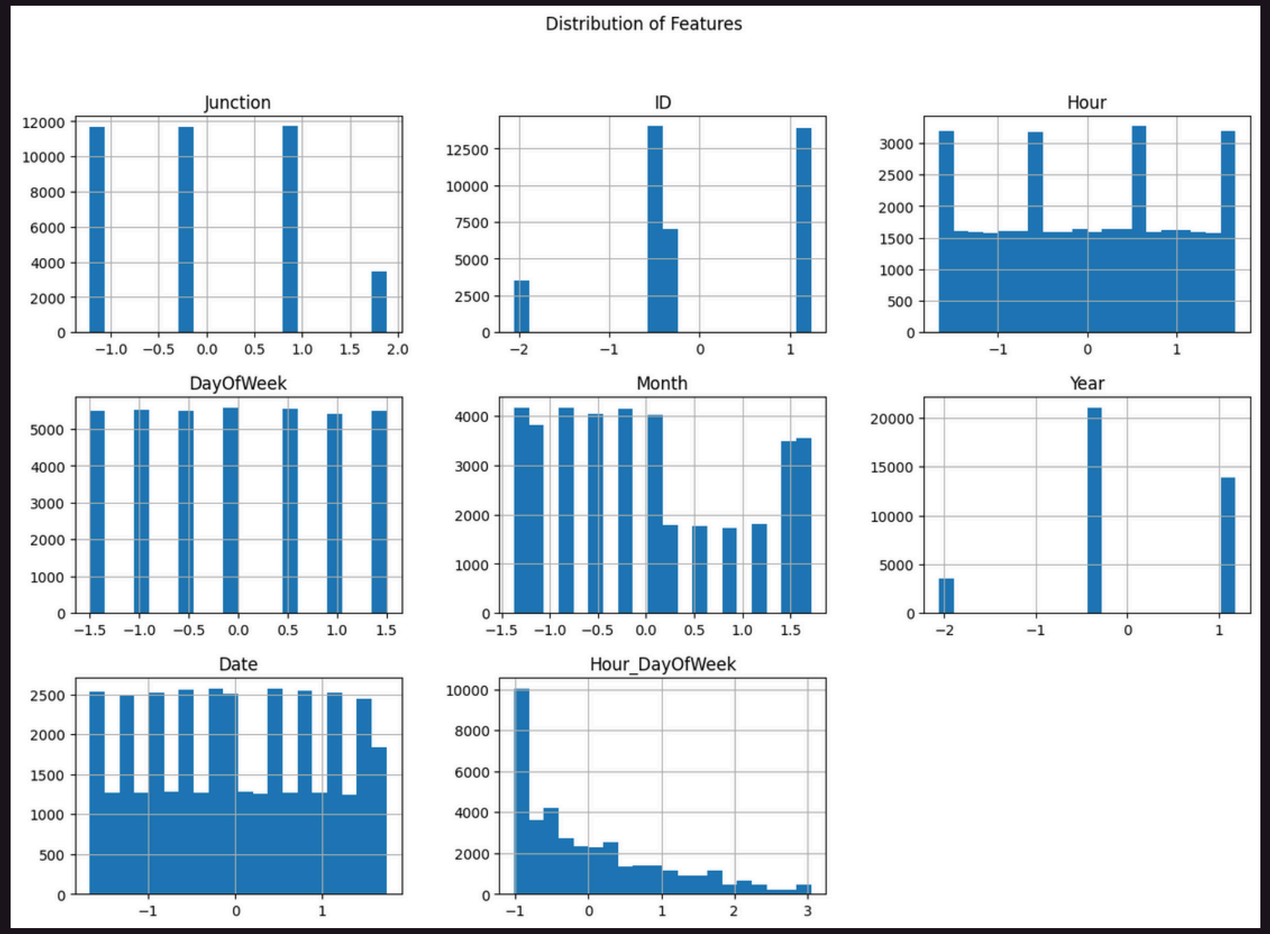
The initial dataset being big enough is broken into two separate datasets D1 and D2. Each with entries- Hence we will have 2 separate datasets.

Normalization by subtracting Mean and dividing by square root of variance.

Plotting the data to see it’s distribution and the need for it to stay as a feature or to be dropped or to be split into multiple features.

DATASET PLOTS:





FEATURE ENGINEERING

It involves extracting relevant information, identifying patterns, and crafting informative representations of the data to enhance the model's ability to capture the underlying relationships and make accurate predictions or classifications.

- Extract DateTime features
- Interaction Feature: data['HourDayOfWeek']
- Junction Feature: data['Overall Traffic']
- Temporal Feature: data['Trend']

Back to Agenda

1) Extract DateTime features:

These steps extract various date and time components from the ’DateTime’ column and create new features such as hour, day of week, month, year, and date.

```
data[’DateTime’] ← pd.todatetime(data[′DateTime′])
data[’Hour’] ← data[′DateTime′].dt.hour
data[’DayOfWeek’] ← data[′DateTime′].dt.dayofweek
data[’Month’] ← data[′DateTime′].dt.month
data[’Year’] ← data[′DateTime′].dt.year
data[”Date”] ← data[′DateTime′].dt.day
```

2) Interaction Feature: “HourDayOfWeek”

Interaction features are variables created by combining two or more existing features in a dataset to capture potential interactions or relationships between them.

data[’HourDayOfWeek’] ← data[’Hour’] × data[’DayOfWeek’]

This step creates an interaction feature by multiplying the hour and day of week features.

3) Junction Feature: “Overall Traffic”

This step aggregates the traffic situation
across all junctions to create a new feature representing overall traffic, which introduces some noise to prevent
overfitting.

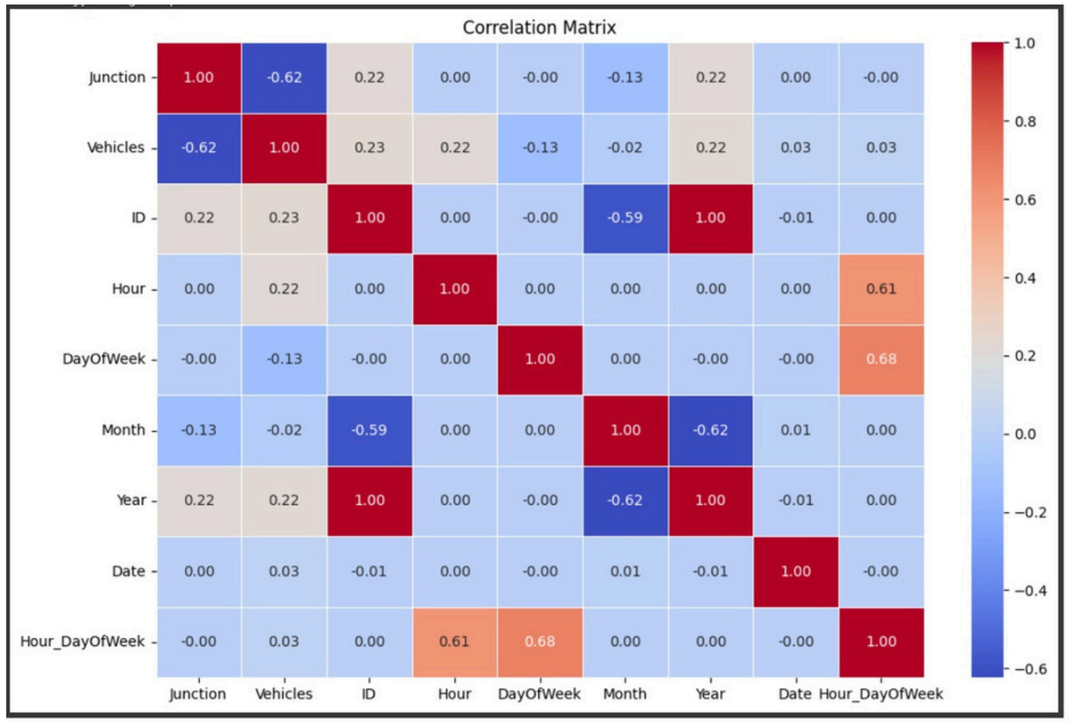
data[[′Junction 1′,′ Junction 2′,′ Junction 3′,′ Junction 4′]].sum(axis = 1)

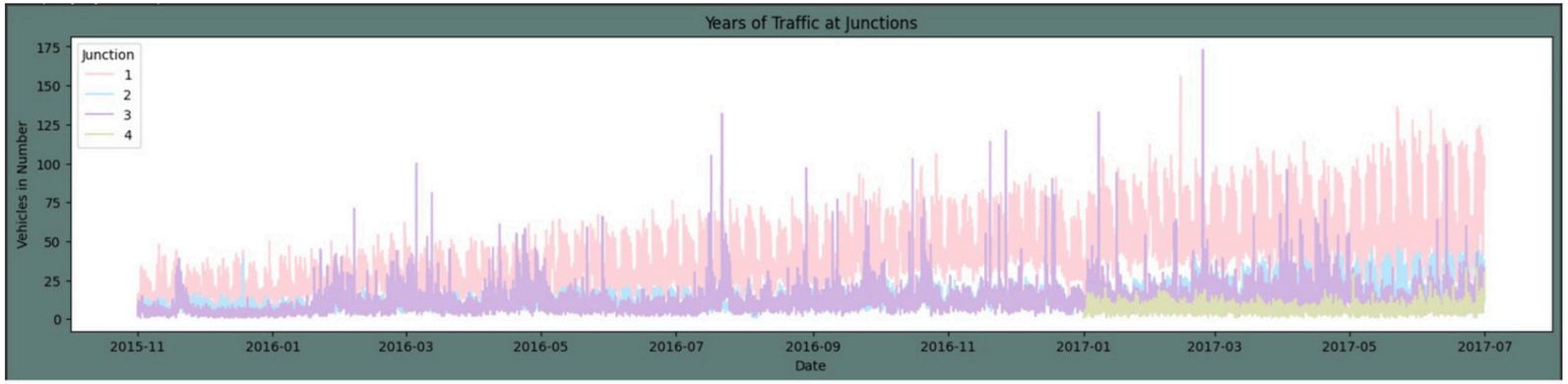
4) Temporal Feature: “TREND”

This step creates a trend feature based on the index of the data.

```
data[[‘Junction 1’,’ Junction 2’,’ Junction 3’,’ Junction 4’]].sum(axis = 1)
```

Plots that helped in Feature Selection:





ALGORITHMS/ METHODOLOGY

1) Linear Regression

2) Random Forests

3) Gradient Boosting

4) XG-Boost

5) LSTM

Back to Agenda

LINEAR REGRESSION

Linear regression, a fundamental technique in machine learning, aims to model the relationship between independent variables and a continuous target variable. The gradient update equations illustrate how the model iteratively adjusts its weights and bias to minimize the difference between predicted and actual values, optimizing the fit of the linear model to the data.

A simple Linear Model was implemented from scratch:

The gradient update equations for the weights (w) and the bias (b) in linear regression are typically written as:

- $wnew=wold−\alpha 1m\sum i=1m(hw,b(x(i))−y(i))x(i)$
- $bnew=bold−\alpha 1m\sum i=1m(hw,b(x(i))−y(i))$

Back to Agenda

RANDOM FOREST

Random Forest is a powerful ensemble learning method used for both classification and regression tasks.
It builds multiple decision trees during training and outputs the average prediction of the individual trees for regression. Random Forest reduces overfitting compared to individual decision trees by averaging multiple predictions.

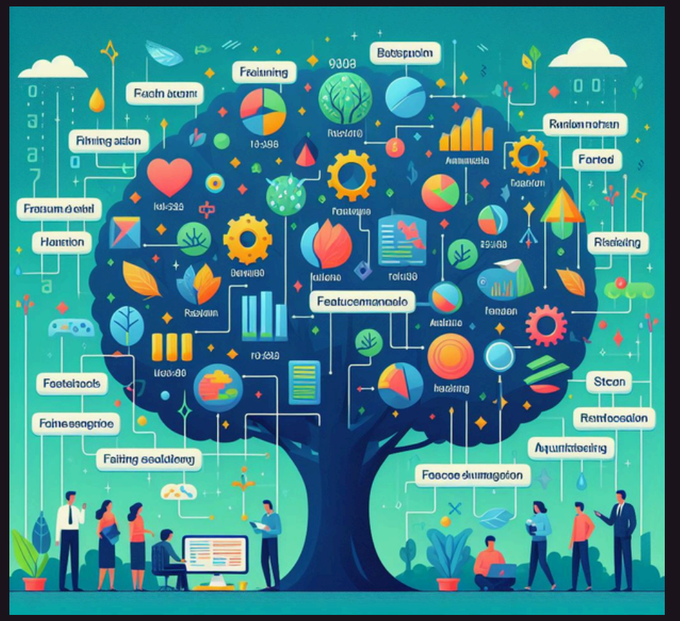
RANDOM FOREST

First the data is normalized and then split into training and testing sets.

Next, columns are selected that could be converted to np.float64 as the features.

A Random Forest Regressor is initialized and trained on the training data with the selected features.

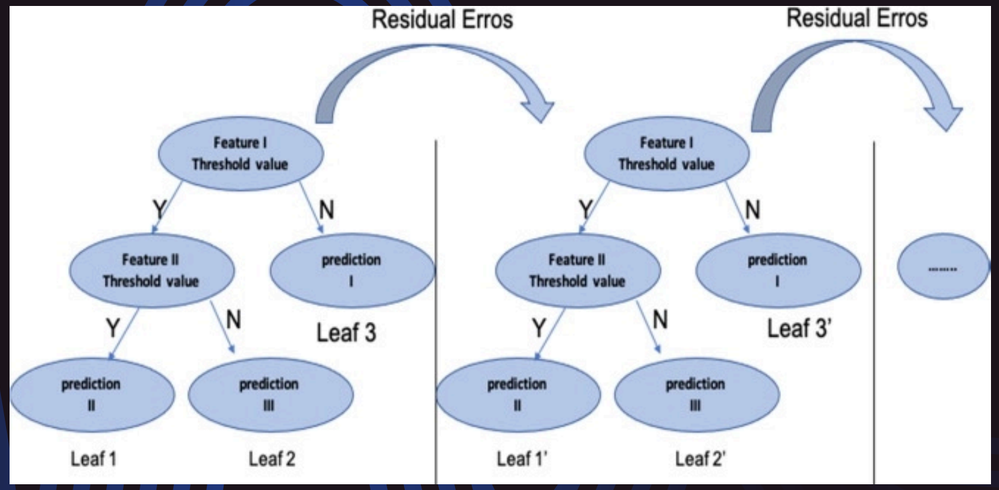
The model achieved an MSE of 0.2403, indicating that, on average, my predictions were about 0.49 off from the actual values.

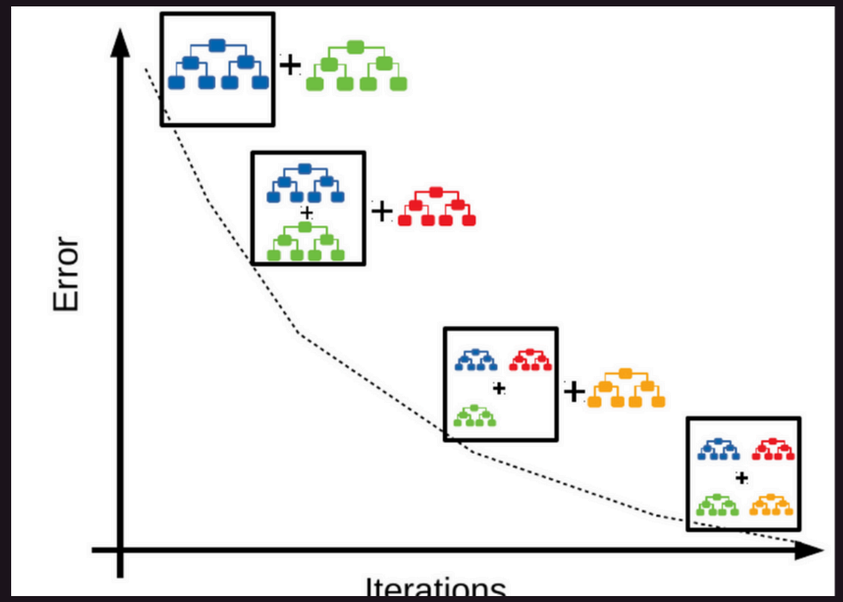


GRADIENT BOOSTING

Gradient Boosting is an ensemble learning technique that builds models sequentially, with each new model correcting errors made by the previous ones. It combines multiple weak learners to create a strong learner.

Gradient Boosting is often used with decision trees as base learners, known as Gradient Boosted Trees. It is a powerful technique for regression and classification tasks, known for its high performance and ability to handle complex interactions in data.





GRADIENT BOOSTING

In the model, after preprocessing the data and selecting the relevant numerical columns, these features are utilized as input for the gradient boosting model.

The model is trained and initialized sequentially, where each new model in the sequence focused on reducing the errors made by the previous models. This iterative process allowed the model to improve its predictive performance gradually.

MSE is chosen as the Evaluation Metric.

XG BOOSTING

XGBoost (Extreme Gradient Boosting) is an optimized and efficient implementation of gradient boosting. It is widely used for supervised learning tasks, especially in structured/tabular data, and has gained popularity for its speed and performance.

XGBoost improves upon traditional gradient boosting by adding regularization techniques and parallel processing, making it highly scalable and effective in handling large datasets.

It also supports various objective functions and evaluation metrics, allowing for fine-tuning to specific problem domains

XG BOOSTING

XGBoost library for regression tasks was used.
First, the dataset was copied, and the features and target variable ('Vehicles') were separated.
The dataset was then split into training and testing sets.

The numerical columns were normalized using MinMaxScaler. Specifically, I normalized the selected features ('Junction', 'Hour', 'DayOfWeek', 'ID') separately for both the training (X_pca) and testing datasets.

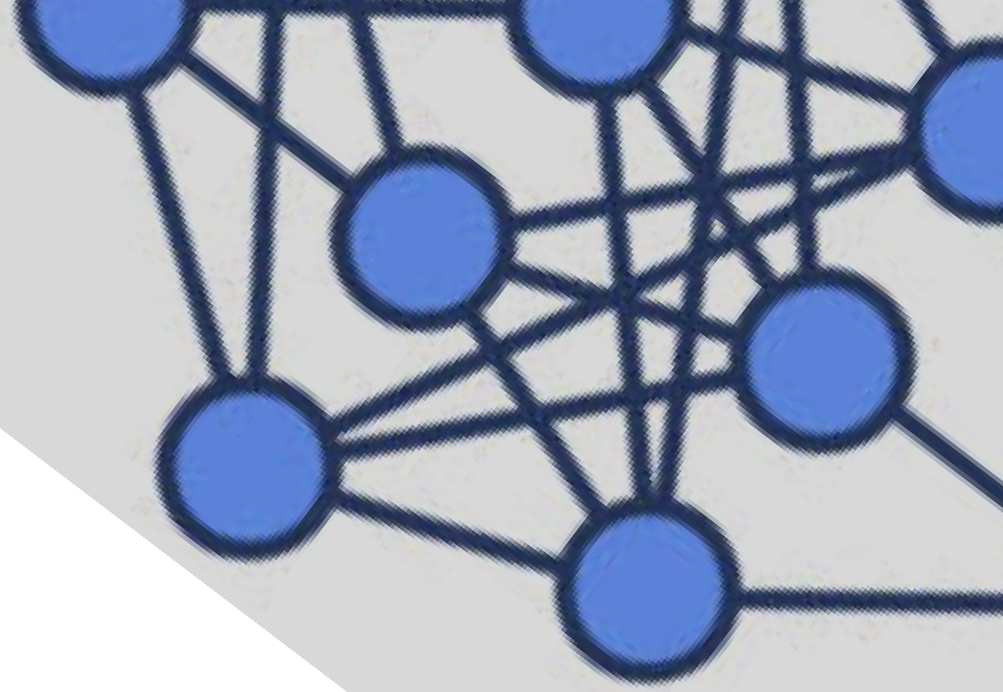
Next, trained the XGBRegressor model on the normalized training data (X_pca, y_train). Then used the trained model to predict the target variable for the normalized testing data (e), storing the predictions in y_pred.
Finally, evaluated the model's performance by calculating the mean squared error (MSE) between the actual target values (y_test) and the predicted values (y_pred).

The trained XGBRegressor model and its corresponding MSE were saved in a dictionary (Model_mse_saver).

SML CSE-342

NEURAL NETWORKS

LSTM



Back to Agenda

LSTM

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture designed to capture long-term dependencies in sequential data. Unlike traditional RNNs, LSTMs have a more complex structure with multiple gates that control the flow of information, allowing them to remember information for long periods.

LSTMs are widely used in tasks such as natural language processing (NLP), speech recognition, and time series prediction, where long-range dependencies are crucial. They excel at capturing patterns in sequential data and are known for their ability to handle vanishing gradient problems, which can occur in traditional RNNs.

In essence, LSTM networks are well-suited for learning from sequences of data and have become a fundamental tool in the field of deep learning for sequential tasks.

LSTM CODE EXP.

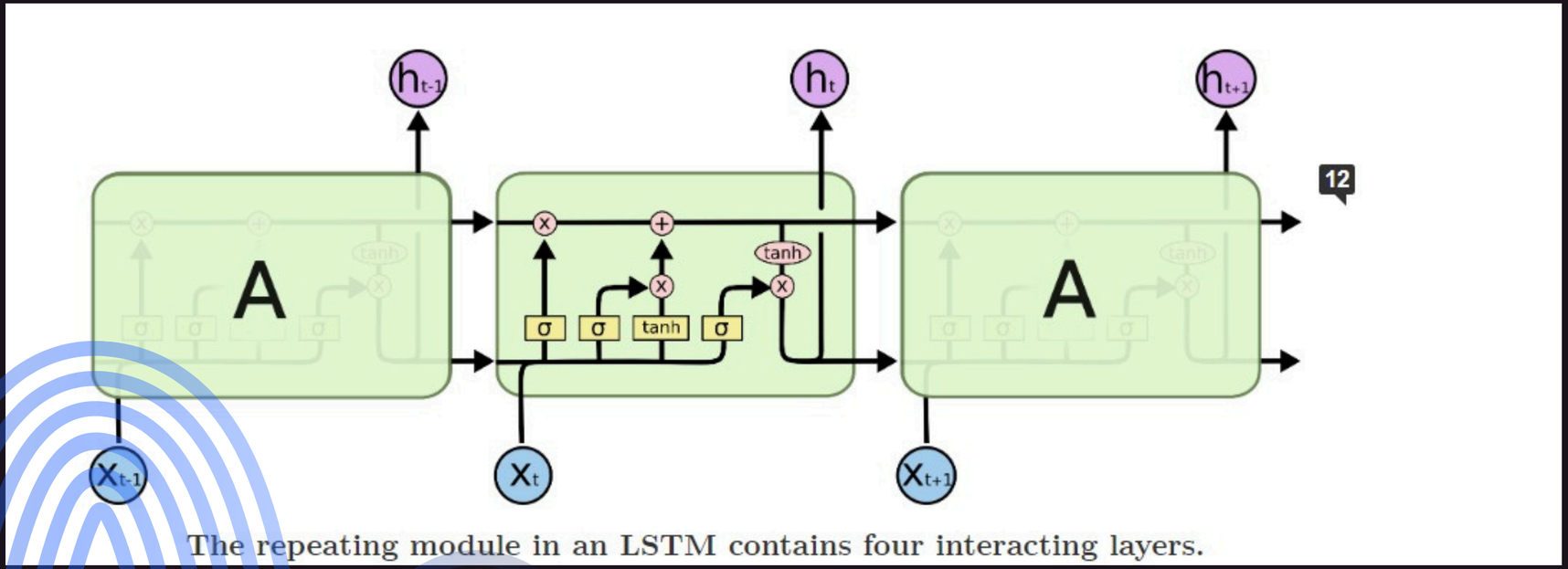
In this code, LSTM (Long Short-Term Memory) neural networks are used for a time series prediction task involving vehicle traffic data. The dataset is preprocessed by dropping unnecessary columns and normalizing the numerical features using MinMaxScaler.

LSTM requires a specific input shape, so the features are reshaped to be suitable for LSTM input. The LSTM model is then built with one LSTM layer and one Dense output layer for regression. The model is compiled using the Adam optimizer and mean squared error loss function.

During training, the model is fit to the training data for a specified number of epochs and batch size. After training, the model is evaluated on the test data, and the mean squared error (MSE) is calculated to measure the model's performance.

The use of LSTM in this code showcases its capability to effectively model and predict patterns in sequential data, making it a powerful tool for time series prediction tasks.

LSTM



The repeating module in an LSTM contains four interacting layers.

LSTM UPDATION EQUATIONS

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

LSTM first forget layer update eqn if sigmod gives 1 it forgets else it gives 0
input layer ki eqn in dono ka point wise multiply yeh decide krega kuch naya add krna hai Long term memory me ya nhi

LSTM ON EACH JUNCTION

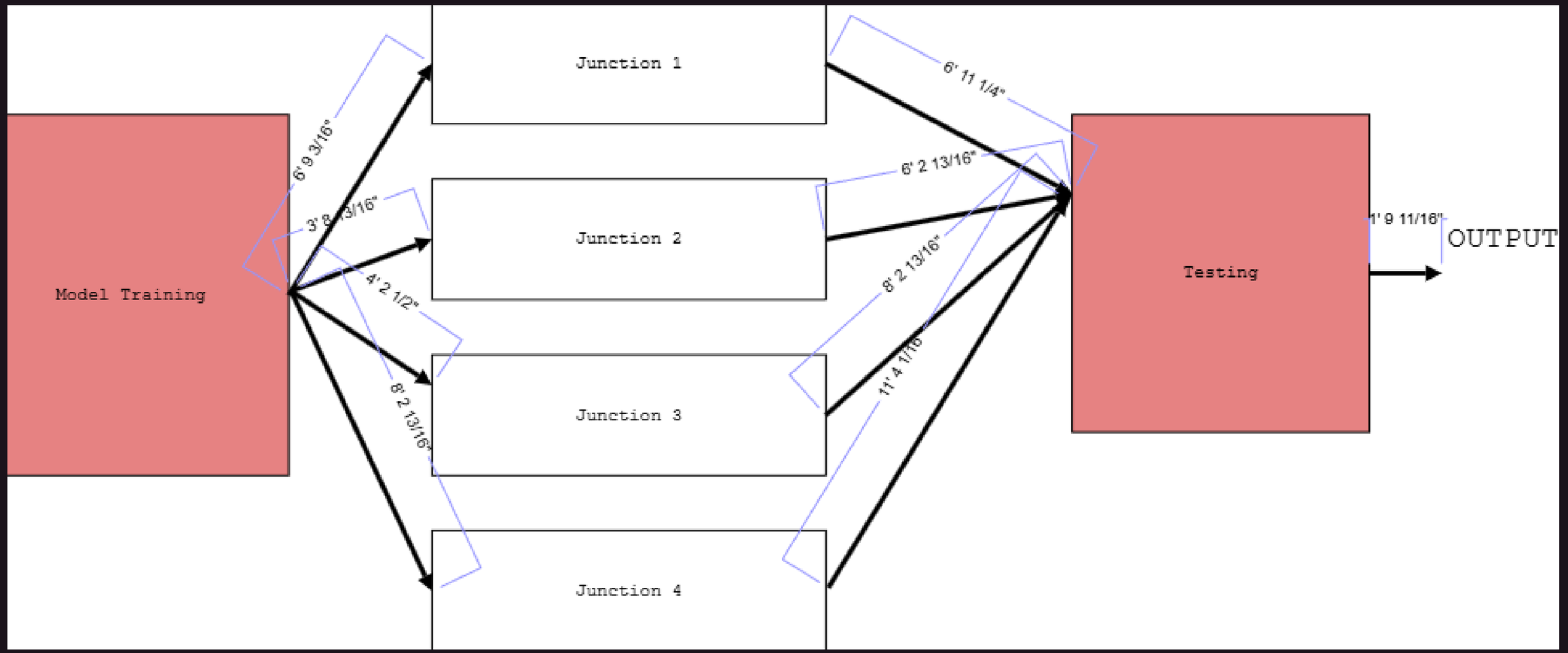
In this code snippet, the dataset is split into features (X) and the target variable (y), and the data is preprocessed and normalized. Then, the code iterates over unique junction values in the test data. For each junction, it filters the training and testing data to include only data points related to that junction.

The LSTM model is then built and trained on the filtered training data for each junction. By training a separate model for each junction, the model can learn the specific patterns and characteristics of traffic flow unique to that junction. This approach can potentially improve prediction accuracy, as each model is specialized to a specific junction's behavior.

By evaluating and predicting on the filtered test data for each junction, the code calculates the mean squared error (MSE) for each junction-specific model. This allows for a more fine-grained evaluation of the model's performance, as it assesses how well the model predicts traffic flow for each individual junction. The MSE values for each junction are printed, showing the performance of each model in predicting traffic flow for its respective junction.

Overall, training a separate model for each junction allows the model to capture the nuances and variations in traffic flow patterns across different junctions, potentially leading to a reduction in MSE compared to using a single model for all junctions.

MODEL DIAGRAM



COMBINING XGBOOST AND RANDOM FOREST RESULTS FOR LSTM INPUT:

Briefly elaborate on how your research analysis will go.

Back to Agenda

LSTM COMBINATION OF OTHER

\

In this code, an ensemble approach is used to improve the prediction performance of an LSTM model. The dataset is preprocessed and normalized. Then, XGBoost and Random Forest models are trained on a subset of features ('Junction', 'Hour', 'DayOfWeek', 'ID') from the dataset.

The predictions from these ensemble models are added as additional features to the original dataset. The LSTM model is then built and trained on this augmented dataset, which now includes the ensemble models' predictions as input features.

By incorporating the predictions from the ensemble models, the LSTM model can learn from the insights provided by these models, potentially capturing patterns and relationships in the data that might have been missed by the individual models alone. This ensemble approach aims to further reduce the mean squared error (MSE) of the final model, and in this case, it succeeded in achieving this goal.

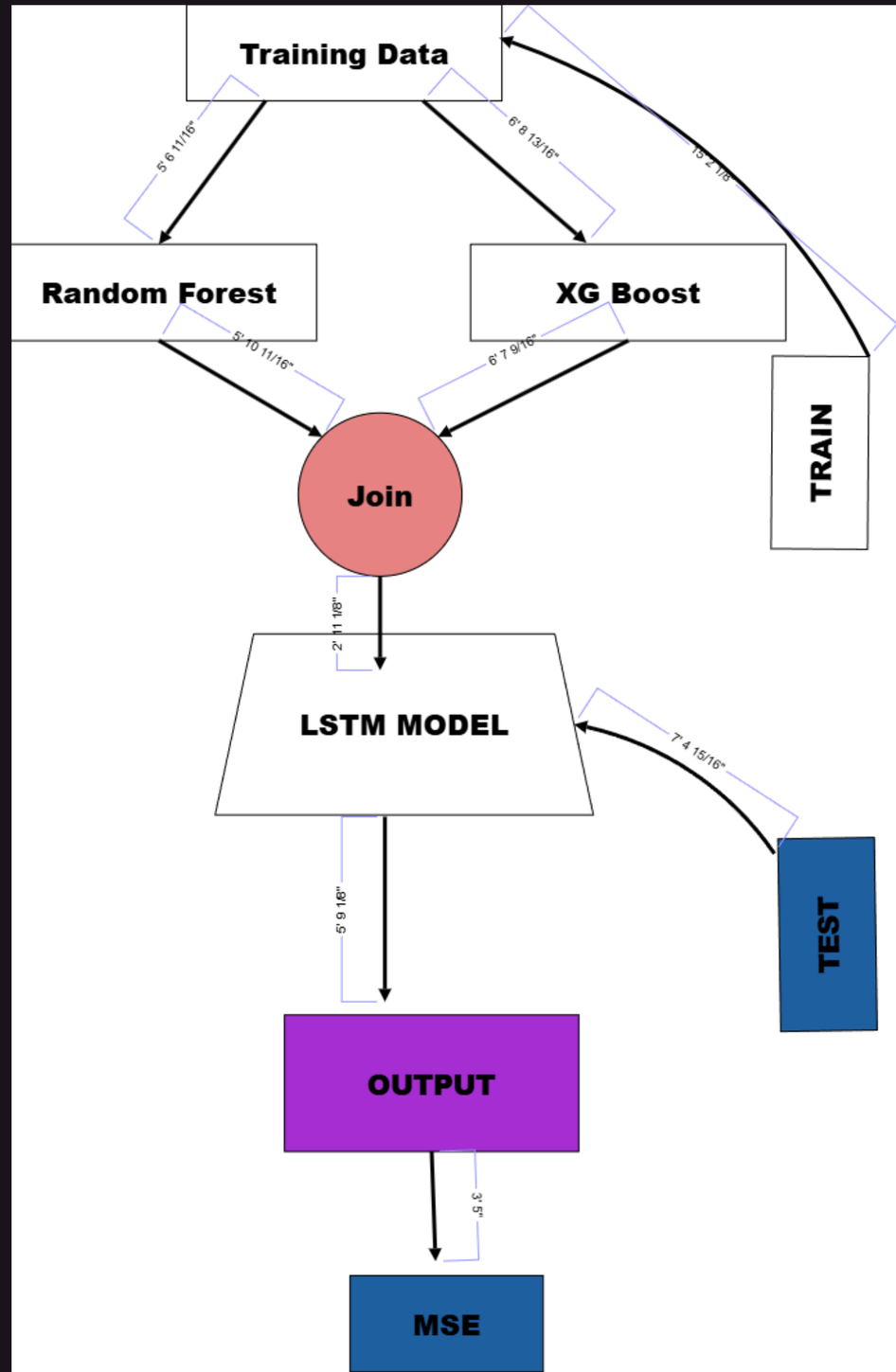
MODEL EXPLAINATION

We can clearly see First we have our training done on Random Forest and then on XG boost .

Hmmm can their predictions be used ???

YUPS,

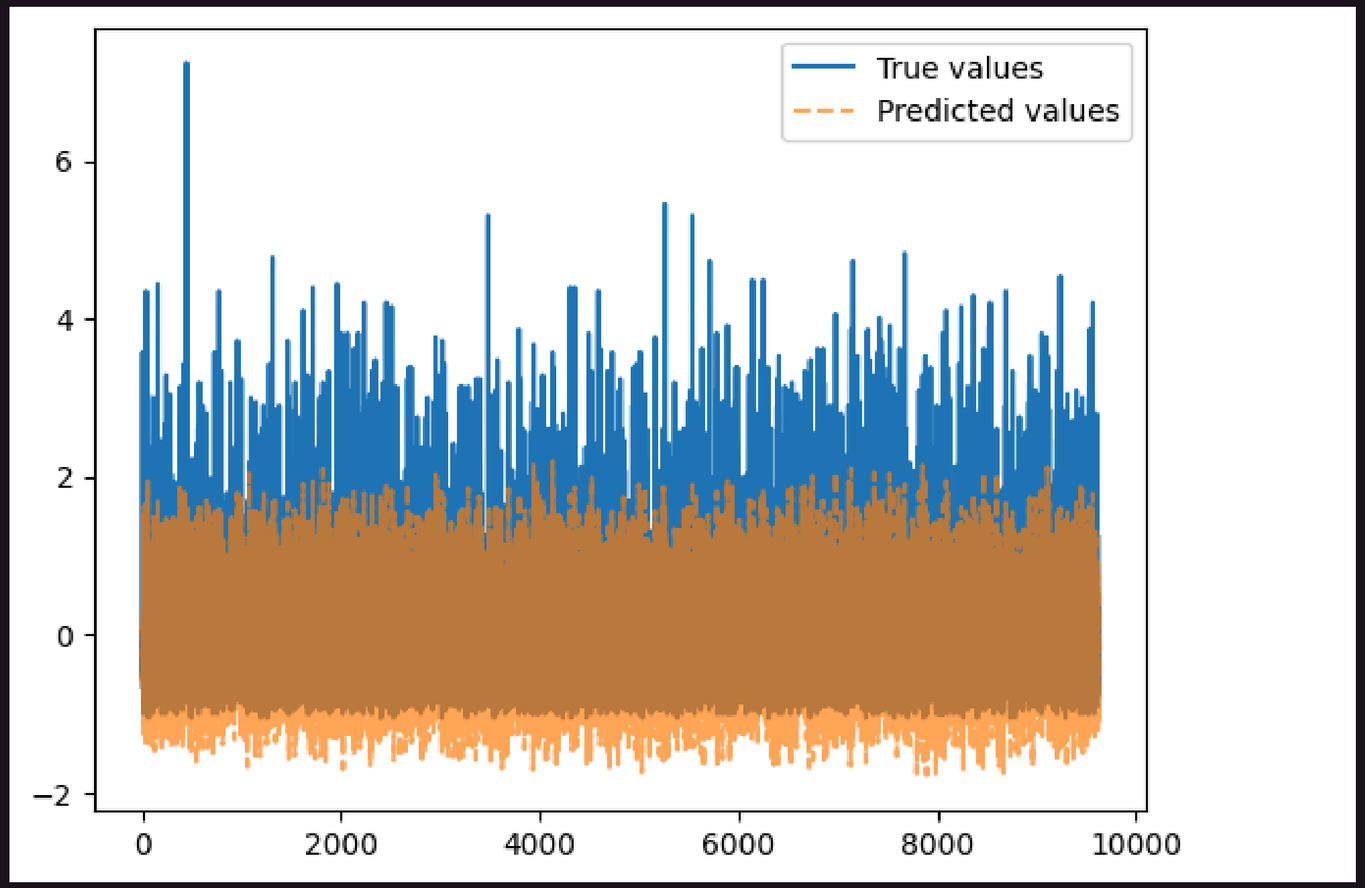
We can provide them as input to Our LSTM to further Reduce Loss It can store these features in its Long Run Memory Rest new data can be added and also Short Term can store rest features and Produce Output



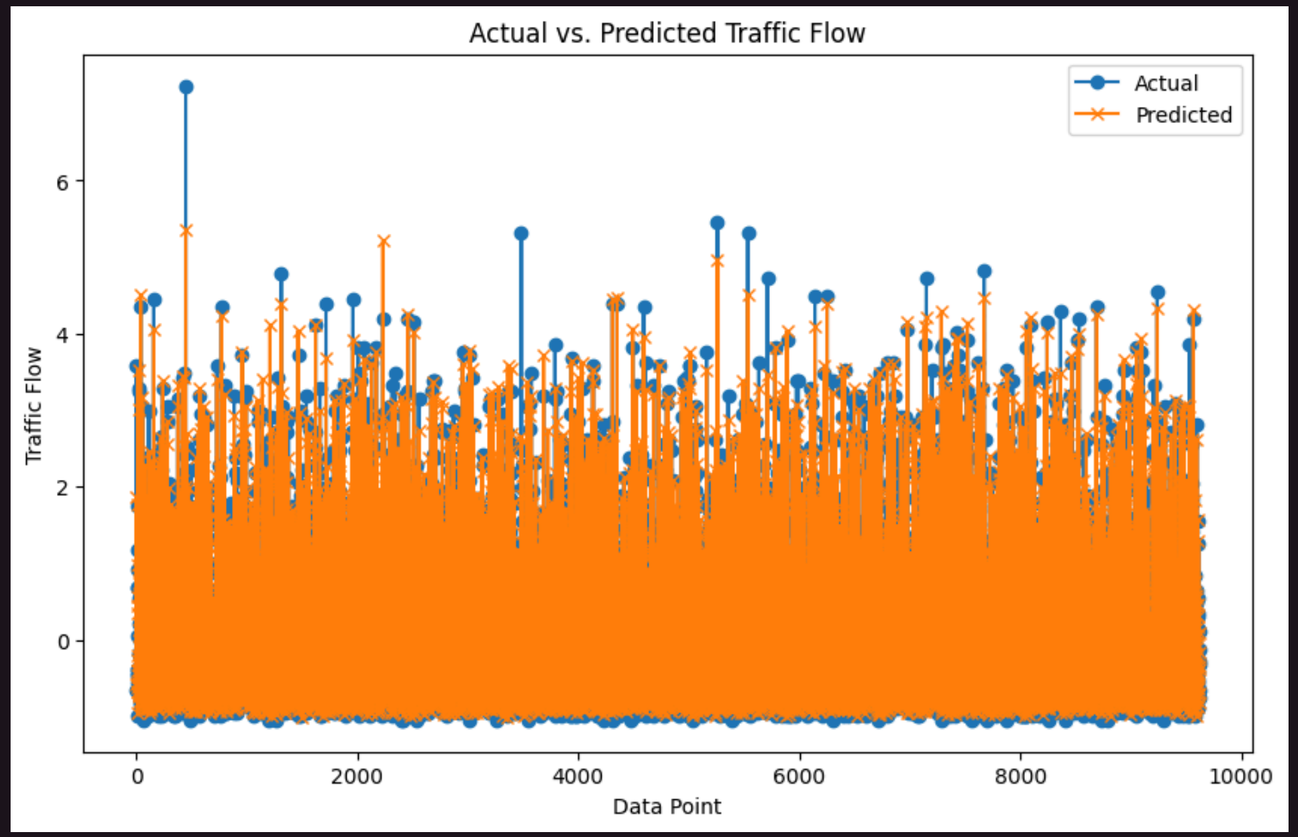
FINDINGS

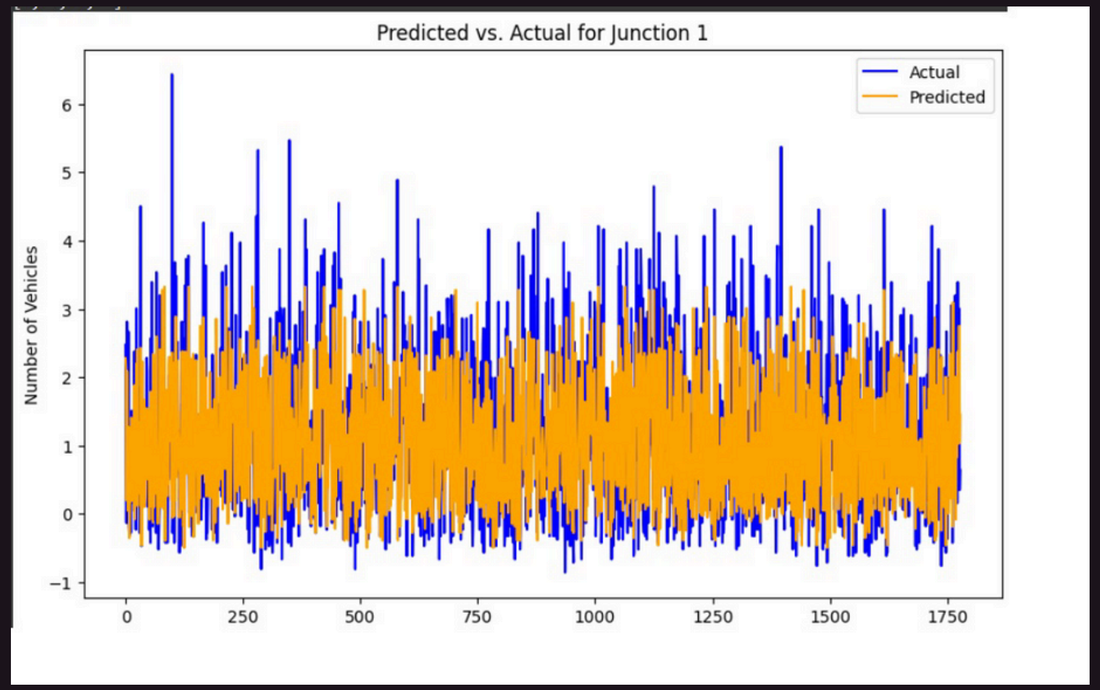
Back to Agenda

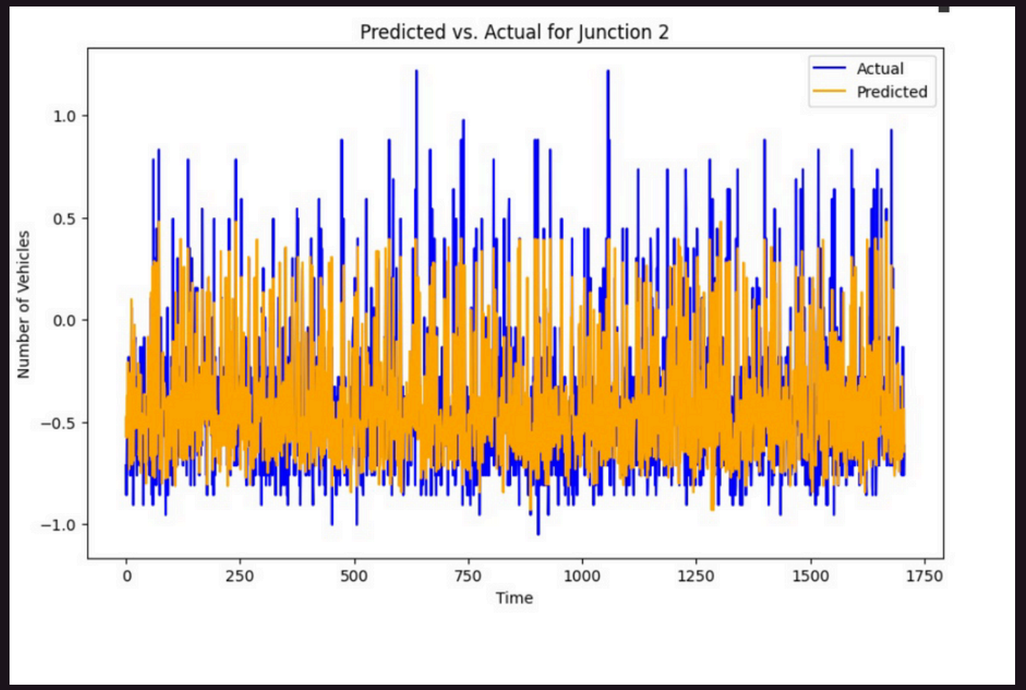
Linear Regression Output:



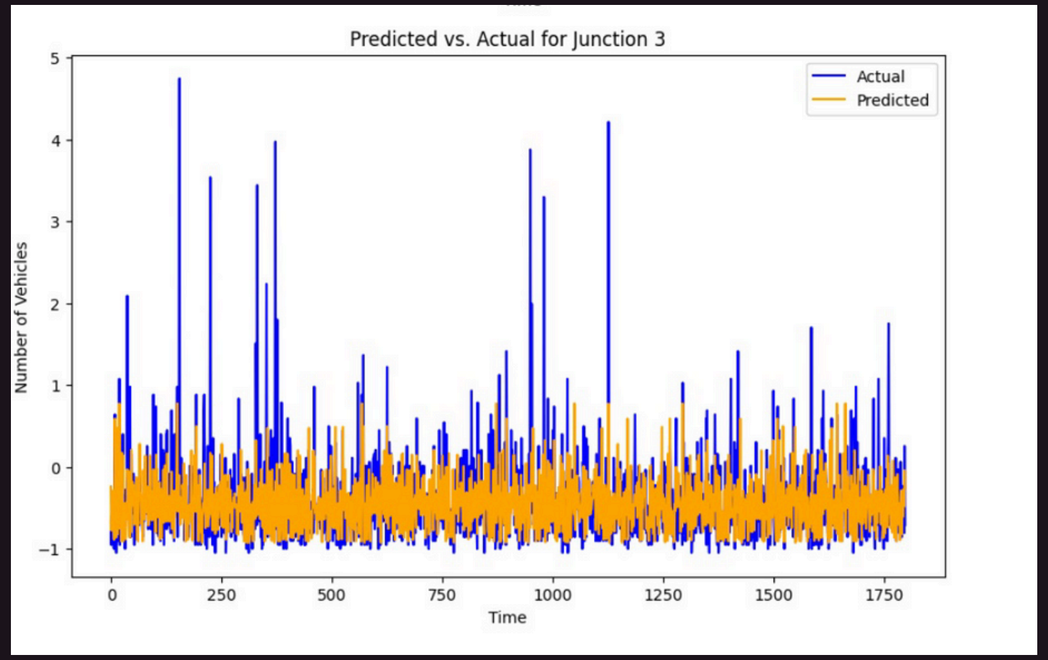
Random Forest Output:

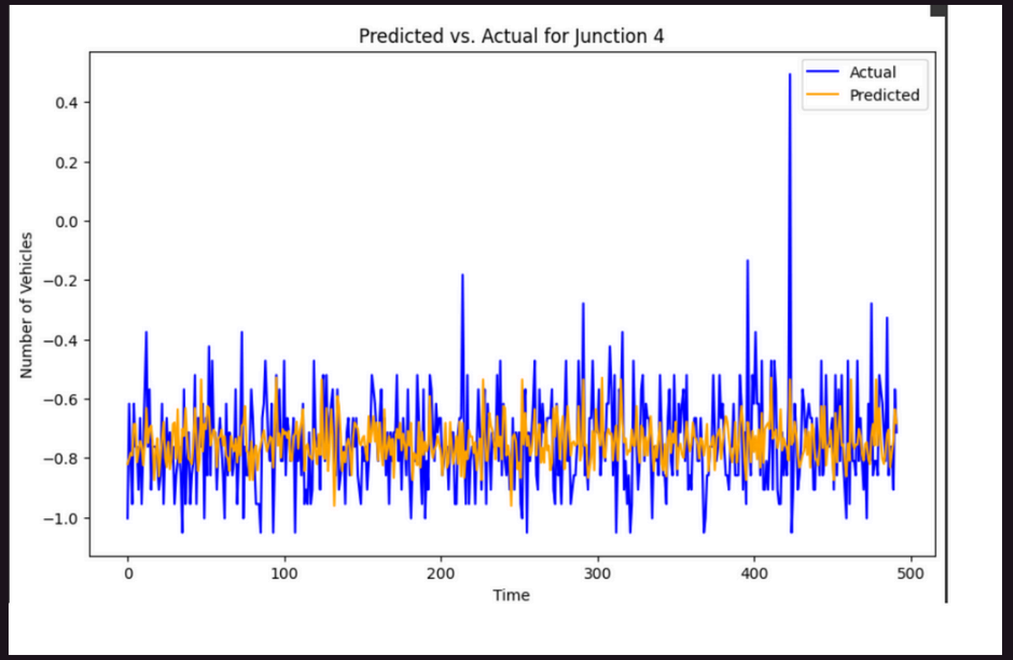






Random Forest Results for Junction 1 (left) and Junction 2 (right).

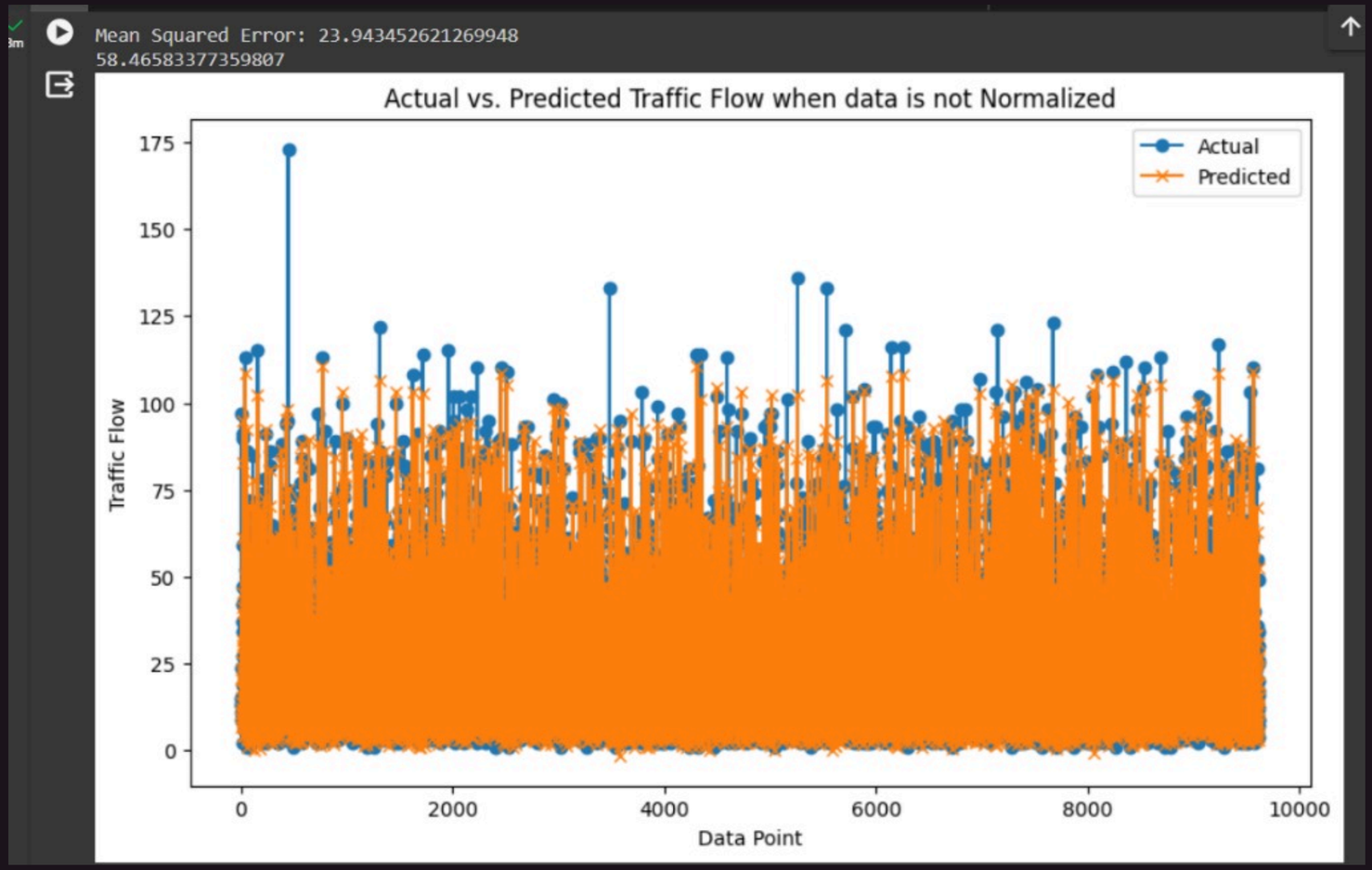




Random Forest Results for Junction 3 (left) and Junction 4 (right).

Gradient Boosting Output:

MSE= 23.94



XG Boost Output:

TABLE I
XGBOOST RESULTS

Model	Mean Squared Error (MSE)
XGBoost	20.826804406348938

Add Company Name

TESTING LSTM ON EACH JUNCTION:

Briefly elaborate on how your research analysis will go.

Back to Agenda

EPOCHS ON JUNCTIONS

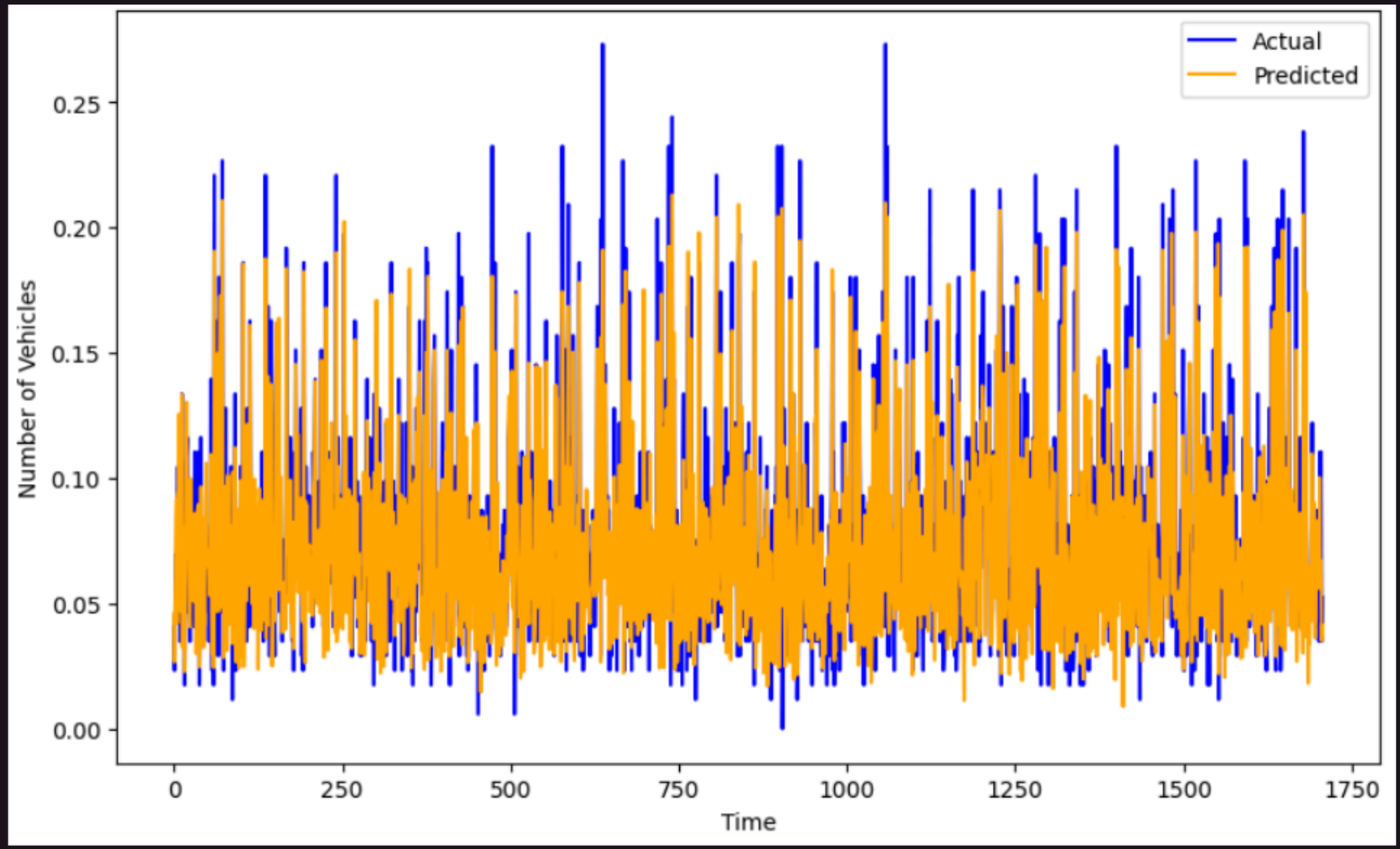
```
Epoch 1/10
554/554 [==============================] - 7s 7ms/step - loss: 6.8679e-04 - val_loss: 4.3262e-04
Epoch 2/10
554/554 [==============================] - 3s 5ms/step - loss: 4.1726e-04 - val_loss: 3.6327e-04
Epoch 3/10
554/554 [==============================] - 3s 5ms/step - loss: 3.5276e-04 - val_loss: 3.5405e-04
Epoch 4/10
554/554 [==============================] - 5s 9ms/step - loss: 3.4012e-04 - val_loss: 5.9366e-04
Epoch 5/10
554/554 [==============================] - 3s 6ms/step - loss: 3.2663e-04 - val_loss: 3.1586e-04
Epoch 6/10
554/554 [==============================] - 3s 6ms/step - loss: 3.2844e-04 - val_loss: 4.9363e-04
Epoch 7/10
554/554 [==============================] - 4s 6ms/step - loss: 3.1238e-04 - val_loss: 3.3640e-04
Epoch 8/10
554/554 [==============================] - 5s 9ms/step - loss: 2.9707e-04 - val_loss: 2.9738e-04
Epoch 9/10
554/554 [==============================] - 3s 6ms/step - loss: 2.9785e-04 - val_loss: 3.6421e-04
Epoch 10/10
554/554 [==============================] - 3s 6ms/step - loss: 2.9269e-04 - val_loss: 3.1818e-04
54/54 [==============================] - 0s 3ms/step
Mean Squared Error (MSE) for Junction 1: 0.0002843516819019646
```

```
Epoch 1/10
565/565 [==============================] - 6s 7ms/step - loss: 0.0027 - val_loss: 0.0022
Epoch 2/10
565/565 [==============================] - 3s 6ms/step - loss: 0.0025 - val_loss: 0.0021
Epoch 3/10
565/565 [==============================] - 4s 6ms/step - loss: 0.0024 - val_loss: 0.0019
Epoch 4/10
565/565 [==============================] - 5s 9ms/step - loss: 0.0024 - val_loss: 0.0020
Epoch 5/10
565/565 [==============================] - 3s 6ms/step - loss: 0.0023 - val_loss: 0.0019
Epoch 6/10
565/565 [==============================] - 3s 6ms/step - loss: 0.0023 - val_loss: 0.0019
Epoch 7/10
565/565 [==============================] - 3s 6ms/step - loss: 0.0023 - val_loss: 0.0019
Epoch 8/10
565/565 [==============================] - 5s 9ms/step - loss: 0.0023 - val_loss: 0.0020
Epoch 9/10
565/565 [==============================] - 3s 6ms/step - loss: 0.0022 - val_loss: 0.0018
Epoch 10/10
565/565 [==============================] - 4s 6ms/step - loss: 0.0022 - val_loss: 0.0019
57/57 [==============================] - 0s 2ms/step
Mean Squared Error (MSE) for Junction 2: 0.00247280544645017
```

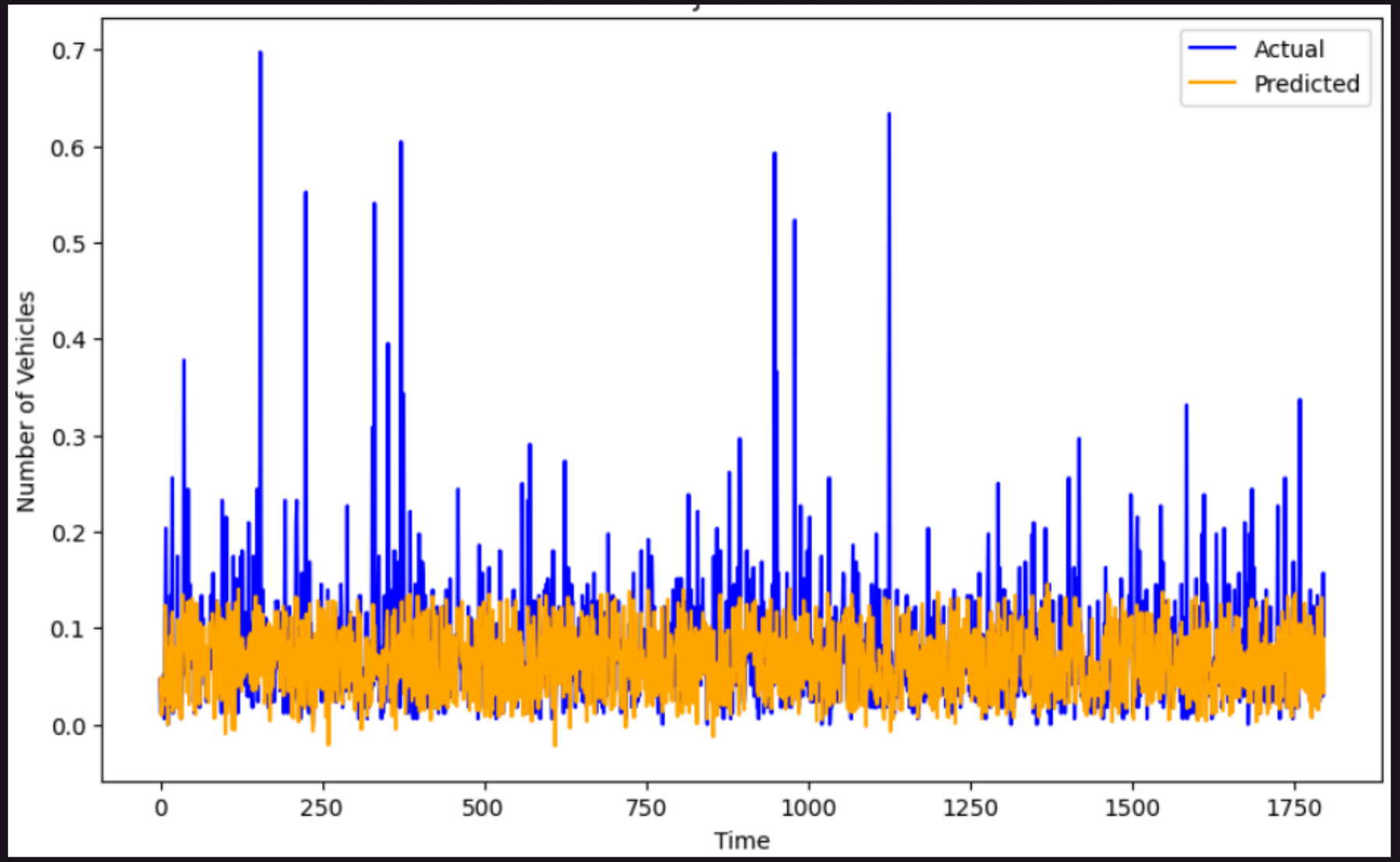
```
Epoch 1/10
561/561 [==============================] - 6s 9ms/step - loss: 0.0067 - val_loss: 0.0046
Epoch 2/10
561/561 [==============================] - 3s 5ms/step - loss: 0.0038 - val_loss: 0.0034
Epoch 3/10
561/561 [==============================] - 3s 6ms/step - loss: 0.0028 - val_loss: 0.0025
Epoch 4/10
561/561 [==============================] - 3s 6ms/step - loss: 0.0023 - val_loss: 0.0024
Epoch 5/10
561/561 [==============================] - 5s 9ms/step - loss: 0.0021 - val_loss: 0.0021
Epoch 6/10
561/561 [==============================] - 3s 6ms/step - loss: 0.0019 - val_loss: 0.0019
Epoch 7/10
561/561 [==============================] - 3s 5ms/step - loss: 0.0019 - val_loss: 0.0019
Epoch 8/10
561/561 [==============================] - 3s 6ms/step - loss: 0.0018 - val_loss: 0.0018
Epoch 9/10
561/561 [==============================] - 5s 9ms/step - loss: 0.0017 - val_loss: 0.0028
Epoch 10/10
561/561 [==============================] - 3s 5ms/step - loss: 0.0017 - val_loss: 0.0018
56/56 [==============================] - 0s 2ms/step
Mean Squared Error (MSE) for Junction 3: 0.0018715477700100207
```

```
Epoch 1/10
169/169 [==============================] - 3s 8ms/step - loss: 4.0130e-04 - val_loss: 3.6592e-04
Epoch 2/10
169/169 [==============================] - 1s 7ms/step - loss: 3.5854e-04 - val_loss: 3.0926e-04
Epoch 3/10
169/169 [==============================] - 2s 10ms/step - loss: 3.3158e-04 - val_loss: 2.8123e-04
Epoch 4/10
169/169 [==============================] - 2s 10ms/step - loss: 3.2819e-04 - val_loss: 2.9761e-04
Epoch 5/10
169/169 [==============================] - 2s 10ms/step - loss: 3.1586e-04 - val_loss: 2.8714e-04
Epoch 6/10
169/169 [==============================] - 2s 10ms/step - loss: 3.1647e-04 - val_loss: 2.4423e-04
Epoch 7/10
169/169 [==============================] - 2s 10ms/step - loss: 2.9084e-04 - val_loss: 2.6640e-04
Epoch 8/10
169/169 [==============================] - 1s 9ms/step - loss: 2.8784e-04 - val_loss: 2.9463e-04
Epoch 9/10
169/169 [==============================] - 1s 6ms/step - loss: 2.8824e-04 - val_loss: 2.2234e-04
Epoch 10/10
169/169 [==============================] - 1s 6ms/step - loss: 2.8659e-04 - val_loss: 2.4441e-04
16/16 [==============================] - 0s 5ms/step
Mean Squared Error (MSE) for Junction 4: 0.00024334803851860373
```

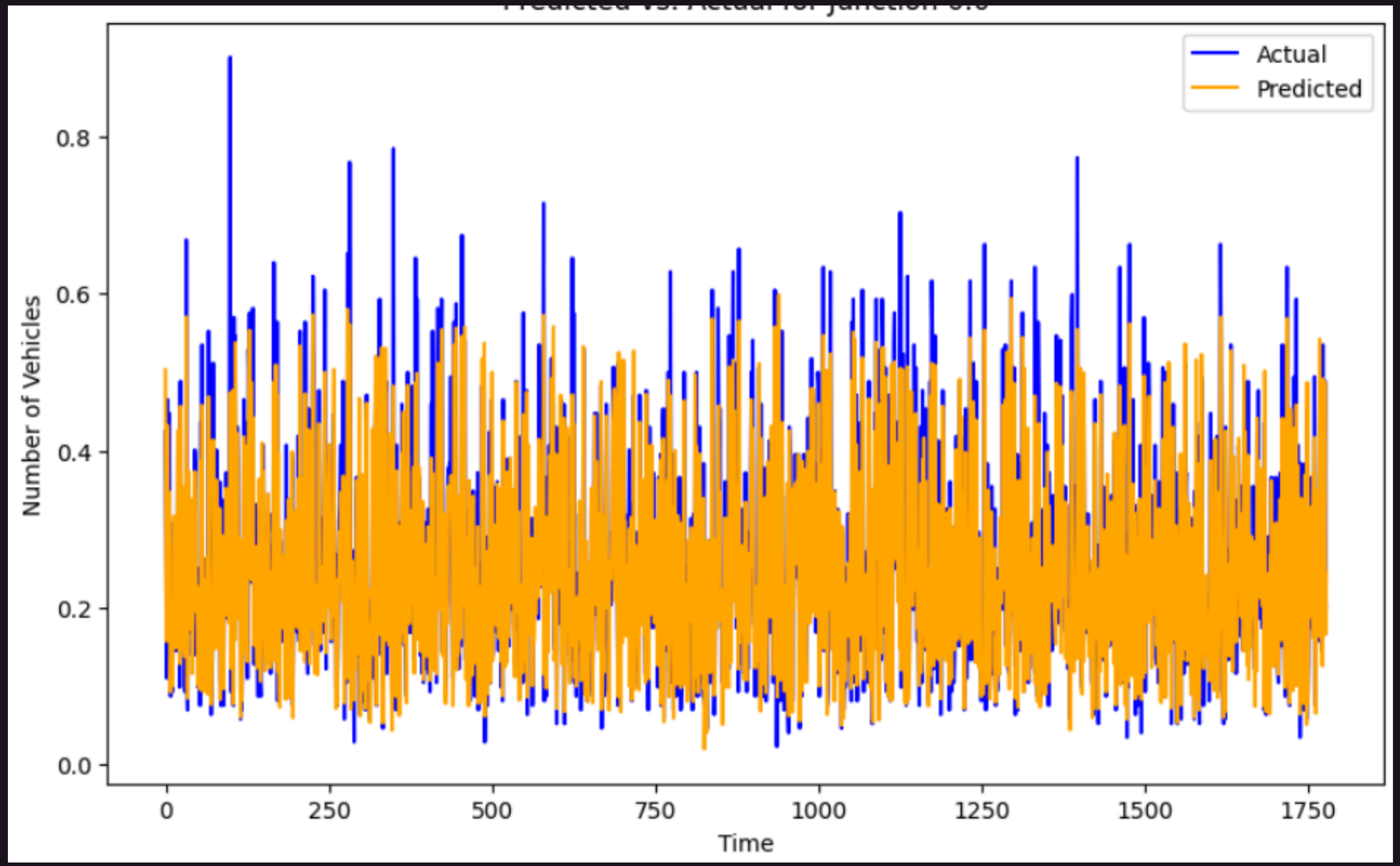
GRAPH JUNCTION 1



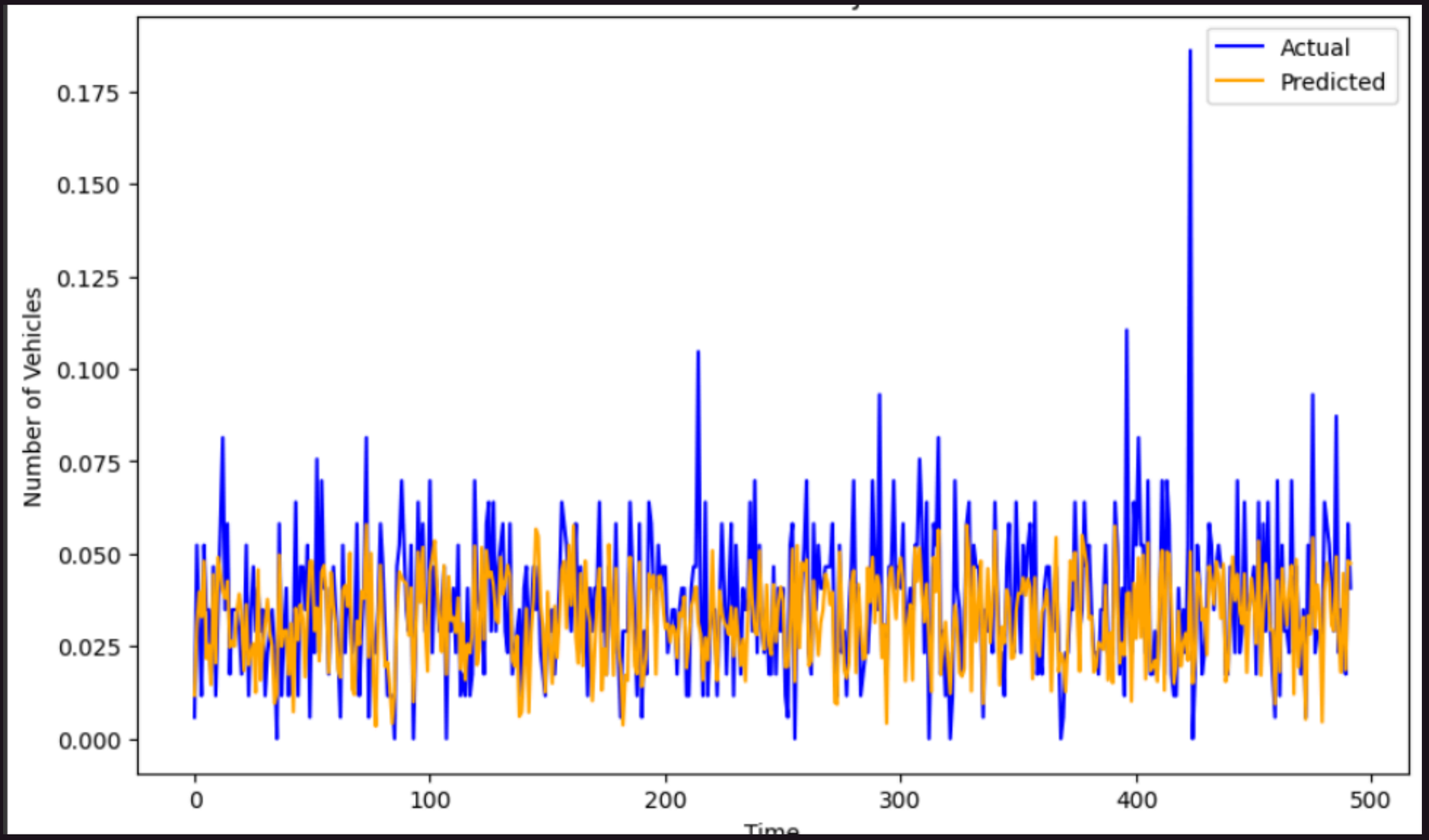
GRAPH JUNCTION 2



GRAPH JUNCTION 3



GRAPH JUNCTION 4



Add Company Name

LSTM RESULT ANALYSIS ON FULLY HIDDEN DATASET:

Briefly elaborate on how your research analysis will go.

Back to Agenda

ANALYSIS

SO ON ANALYSIS OF RESULT ON PURE HIDDEN OTHER DATASET THAT WE HAVE

AS ALREADY SAID WE HAVE 2 DATASETS ON OTHER DATASET WE HAVE TESTED DIRECTLY AS WE THOUGHT IT SHOULD PERFORM IRRESPECTIVE OF DATASET AS WE WANT TO USE IT IN REAL LIFE PREDICTION .

WHAT WE SAW IS SOME MODELS OVERFITTED THE DATA REST PERFORMED GOOD

```
means square error for model Random forest 0.2348358839342759
means square error for model Lstm Layer with Xgboost and random Forest 0.10665118708553166
Index(['Junction', 'ID', 'Hour', 'DayOfWeek', 'Month', 'Year', 'Date',
       'Hour_DayOfWeek'],
      dtype='object')
Index(['Junction', 'ID', 'Hour', 'DayOfWeek', 'Month', 'Year', 'Date',
       'Hour_DayOfWeek'],
      dtype='object')
[1, 2, 3, 4]
182/182 [==============================] - 1s 3ms/step
means square error for model Lstm Layer 1.4134001359334992e+16
187/187 [==============================] - 1s 3ms/step
means square error for model Lstm Layer 3.3023747177703464e+16
180/180 [==============================] - 1s 2ms/step
means square error for model Lstm Layer 2.3273988163054 85e+19
55/55 [==============================] - 0s 2ms/step
means square error for model Lstm Layer 3.1045118752720474e+17
```

INFERENCE

WHAT WE INFER FOR NEURAL NETWORKS IS THAT WITHOUT ANY PREVIOUS INFO FROM OTHER SUORCES OR MODELS IT HAD OVERFITTED THE DATA .

BUT OTHER MODEL WORKED REASONABLY WELL ON IT WE CAN SAY WITH A HIGH CONFIDENCE IT WORKED SUPER WELL ALSO RANDOM FOREST WORKED GREAT.

WE HAVE PROOF THAT RANDOM FOREST NEVERS OVERFITS DATA.

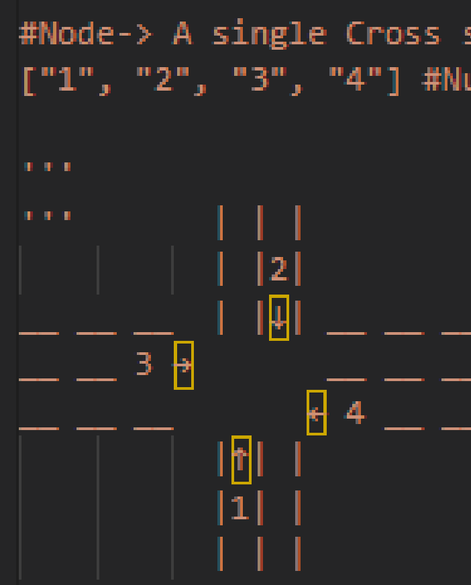
TRAFFIC MANAGEMENT

Extending our traffic prediction
model to a real-life scenario where we aim to predict the timing for each traffic light to optimize traffic flow.

Back to Agenda

```
'''
#Node-> A single Cross section/Junction
["1", "2", "3", "4"] #Number of Cars in Lane 1, 2, 3 and 4 respectively

'''
'''
```




```
'''
# t-> Global Time Taken to be 40 seconds.
# K-> Right Turn Factor
#t/K-> Time given for right Turn

#Current Lane-> The lane which is being considered at the moment.
#Conjugate Lane-> The lane opposite to the current lane.
```

Assumptions:

1) One car can cross the junction in 1 second.

2) Three lights are present for each lane indicating Middle, Left, and Right, changing colors as Green, Yellow, and Red.

3) Wait time ranges from 120 seconds for four critical phases to 180 seconds. One full cycle duration of traffic lights is not to exceed 120 seconds [Gorodokin and Kudryavtseva (2015)]. In our case, we shall be a bit flexible and let it range till 180 seconds (if need be), i.e., instead of 40 seconds per lane, 60 seconds are given to each lane.

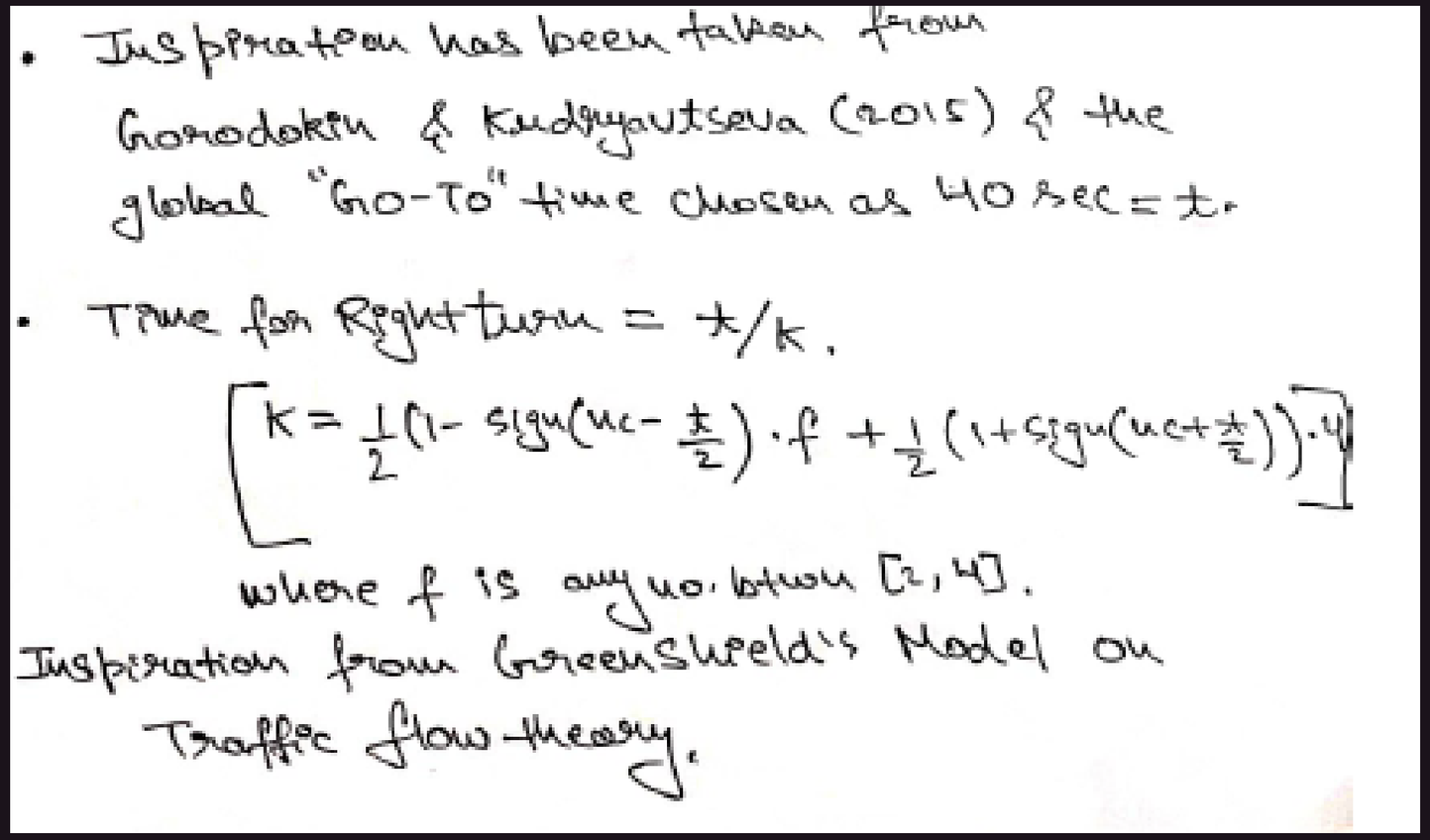
Assumptions:

4) All roads are connected, i.e., no dead ends are present. There are always 4 lanes at a cross section/ junction.

5) Emergency vehicles have utmost priority.

6) One-fourth of vehicles take a right turn, while the rest go straight or left.

Inspiration for the Algorithm:



Algorithm:

This algorithm calculates the green time for the right lane signal.

Algorithm 1 Function calc_K(node, lane)

procedure CALC_K(node, lane)
    $f \leftarrow 0$
    $nc \leftarrow Junctions[node][conjugate(lane)]$
    if $nc < mid/2$ **then**
        $f \leftarrow 2$
    else
        $f \leftarrow 2.666$
    end if
    $k\_fac \leftarrow (1/2) \times (1 - signum(nc - mid)) \times f + (1/2) \times (1 + signum(nc - mid)) \times 4$
    return k_fac
end procedure

Algorithm:

The algorithm in the Colab Notebook tries to calculates the minimum time for the green light to be on, such that all the traffic flows out.

Algorithm 2 Function remove_conjugate_cars(node, lane, remaining_time)

```
procedure REMOVE_CONJUGATE_CARS(node, lane, remaining_time)
    global TIME_COUNTER
    conj_lane ← conjugate(lane)
    print "Lane: ", lane, " Left : Straight - Green ", " Lane: ", conj_lane, " Left : Straight - Green"
    s_time ← 0
    r_time ← 0
    if Junctions[node][conj_lane] ≥ t then
        K_fac ← calc_K(node, conj_lane)
        r_time ← t/K_fac
        s_time ← (3/4) × t
    else if Junctions[node][conj_lane] ≥ t/2 then
        K_fac ← calc_K(node, conj_lane)
        r_time ← t/K_fac
        s_time ← max(r_time, Junctions[node][conj_la
    else
        r_time ← Junctions[node][conj_lane] × (1/4)
        s_time ← Junctions[node][conj_lane] × (3/4)
        r_time ← int(r_time)
        s_time ← int(s_time)
    end if
    print r_time, s_time
    if s_time > remaining_time then
        for i from remaining_time to 0 decreasing by 1 do
            print i
        end for
        TIME_COUNTER += remaining_time
        Junctions[node][lane] −= remaining_time
        Junctions[node][conj_lane] −= remaining_time
        print "TIME: ", TIME_COUNTER
        print "Now Lane: ", lane, " is RED"
        print Junctions
        print "Lane: ", conj_lane, " is Green-¿ Left : Straight : Right"
        if r_time > s_time − remaining_time then
            conj_time_left ← r_time
        else
            conj_time_left ← s_time − remaining_time
        end if
        conj_time_left ← int(conj_time_left)
        for i from conj_time_left to −1 decreasing by 1 do
            print i
        end for
        TIME_COUNTER += conj_time_left
        Junctions[node][conj_lane] −= conj_time_left
        print "Both Lanes 1 and 2 have finished their iteration"
        print Junctions
        print "Now changing to other batch of lanes"
    elseSimilar logic as above for the else case
    end if
end procedure
```

CONCLUSION

Back to Agenda

WHAT WE INFERED HERE IS VERY CRUCIAL IS THAT OUR MODELS WORKED GOOD FOR TRAIN DATA FROM ONE DATASET AND TEST DATA FROM OTHER DATASET.

WE SEE BEST PREDICTIONS ARE GIVEN BY LSTM AND RANDOM FOREST .

THEY DIDNT OVERFIT THE DATA.

NOW, WE CAN USE THEIR INDIVIDUAL CONFIDENCE TO PREDICT AND FINALLY TAKE MEANS.

THESE CONFIDENCES ARE NOT DIRECTLY FOUND THEY ARE HYPERPARAMETER CALCULATED USING CLEAR CONSIDERATION ON MSE’S.

Add Company Name

NEXT-STEPS

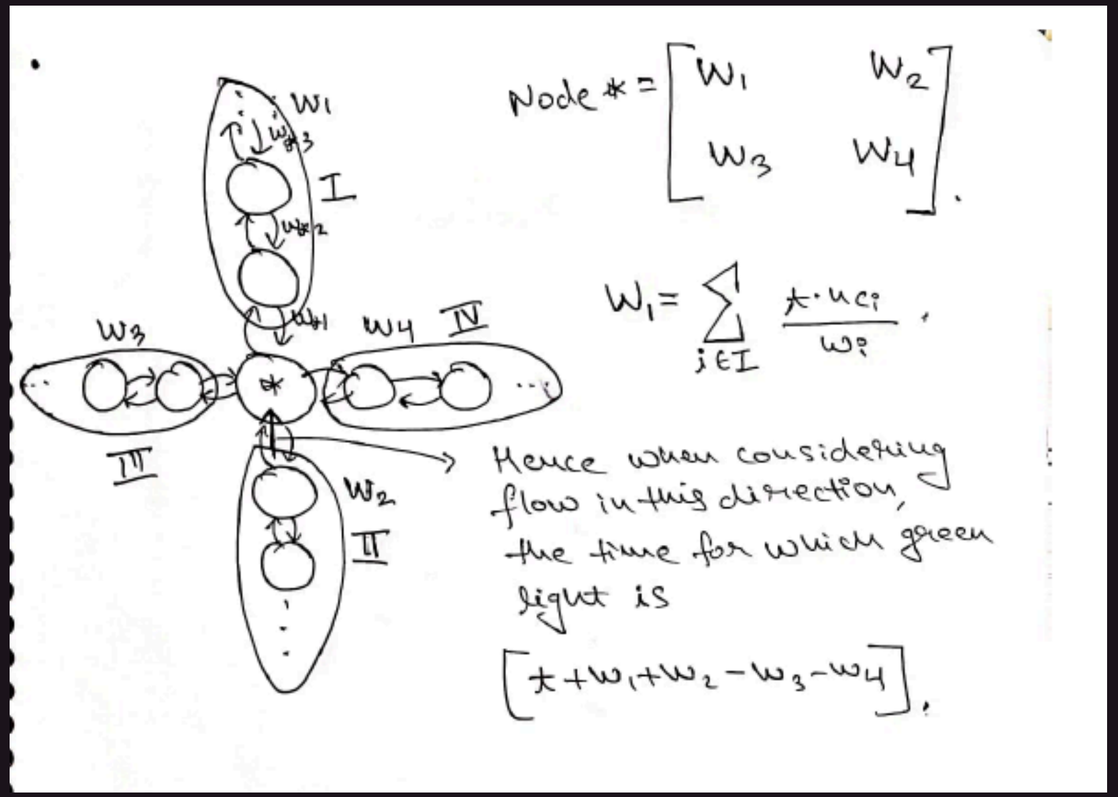
Work that can be continued in future.

Back to Agenda

Implementation on a Directed weighted Graph:

A similar approach as in the figure can be done in which each node/junction has a confidence with which it will change the time of a particular junction.

This confidence will be inversely proportional to the weights of the graph and directly proportional to the number of cars.



REFERENCES

Back to Agenda

LINKS:

https://www.kaggle.com/code/puneetgupta24/ traffic-prediction

https://www.geeksforgeeks.org/ linear-regression-implementation-fromscratch-using-python • https://www.overleaf.com

https://ops.fhwa.dot.gov/publications/signal\ timing/03.htm#: :text=Phase%20Time%20%3D%203.8% 20%2B%202.1%20

Relevant Literature/Case Studies:

1) Probabilistic Model for Signalized Intersection Capacity with a Short Right-Turn Lane:

https://tinyurl.com/yct23w4r

2) Green Shield’s Model:
https://tinyurl.com/ybmn6aj9

3) Traffic flow prediction models:
https://tinyurl.com/2bf856zt