Industrial Internship Report on

”Smart City Traffic Patterns”

Prepared by

Urvashi

Executive Summary

This report provides details of the Industrial Internship provided by upskill Campus and The IoT Academy in collaboration with Industrial Partner UniConverge Technologies Pvt Ltd (UCT).

This internship was focused on a project/problem statement provided by UCT. We had to finish the project including the report in 6 weeks’ time.

My project was ”Smart city traffic prediction using Machine Learning”. I had to test out various models to find the one neural networks that could work with my data efficiently. The government wants to implement a robust traffic system for the city by being prepared for traffic peaks. They want to understand the traffic patterns of the four junctions of the city. Traffic patterns on holidays, as well as on various other occasions during the year, differ from normal working days. This is important to take into account for your forecasting.

This internship gave me a very good opportunity to get exposure to Industrial problems and design/implement solution for that. It was an overall great experience to have this internship.

TABLE OF CONTENTS

1 Preface

I started working on finding ways to tackle traffic prediction through various models. I did the predictive analysis using various Python libraries. I tried using the GRU model to implement the model but wanted to find better solutions because of how slow it was.

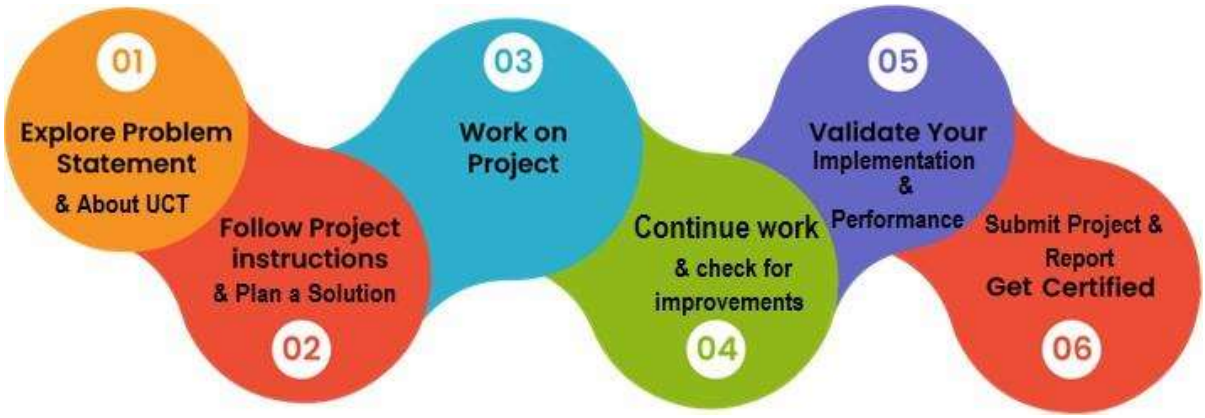
I then tried Neural networks using RNN, Keras, and Tensorflow and surprisingly it worked at a smooth flow and generated plots to visualize the predictions.

Finding the right internship is very difficult for guiding our career in the right direction. It is thanks to Upskill campus program that I was able to achieve the right path, this project served as an accountability factor in my career.

The government wants to implement a robust traffic system for the city by being prepared for traffic peaks. They want to understand the traffic patterns of the four junctions of the city. Traffic patterns on holidays, as well as on various other occasions during the year, differ from normal working days. This is important to take into account for your forecasting. I had to build an ML model to implement forecasting and prediction in this project.

USC/UCT has given me an excellent opportunity to enhance my skill sets and make me industry-level ready.

How Program was planned



Time series models used for forecasting include

- decomposition models,
- exponential smoothing models,

- and ARIMA models, etc

I have decided to use the GRU model which is a part of the RNN deep-learning model.

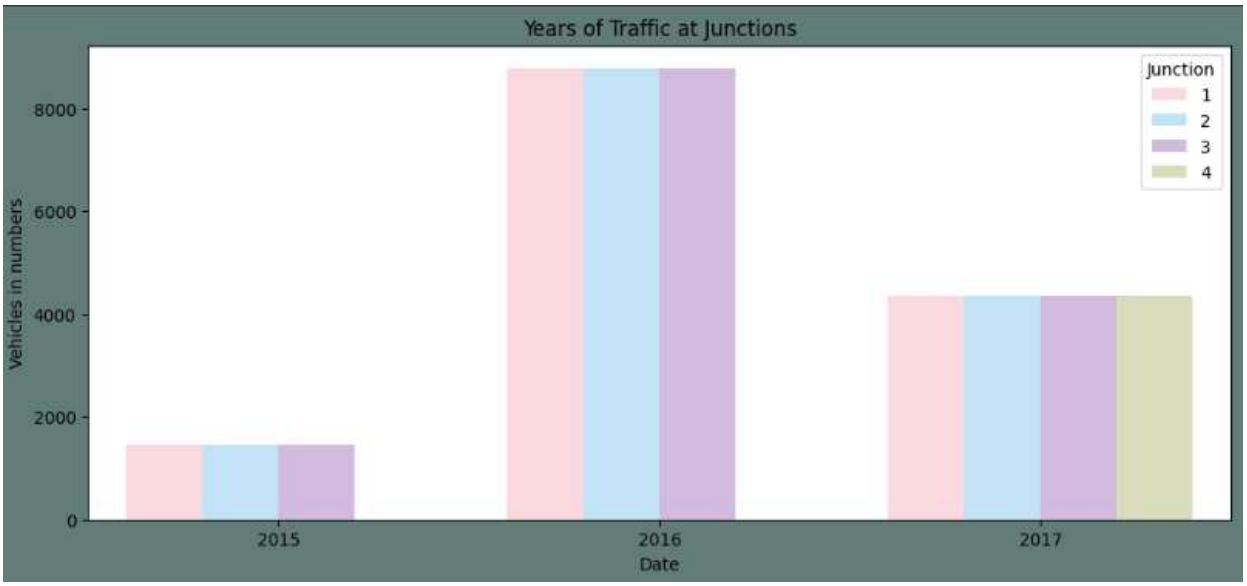
At first, I tried to visualize the data more clearly with the help of matplotlib plots and seaborn scatter plots.

1. Under feature engineering:

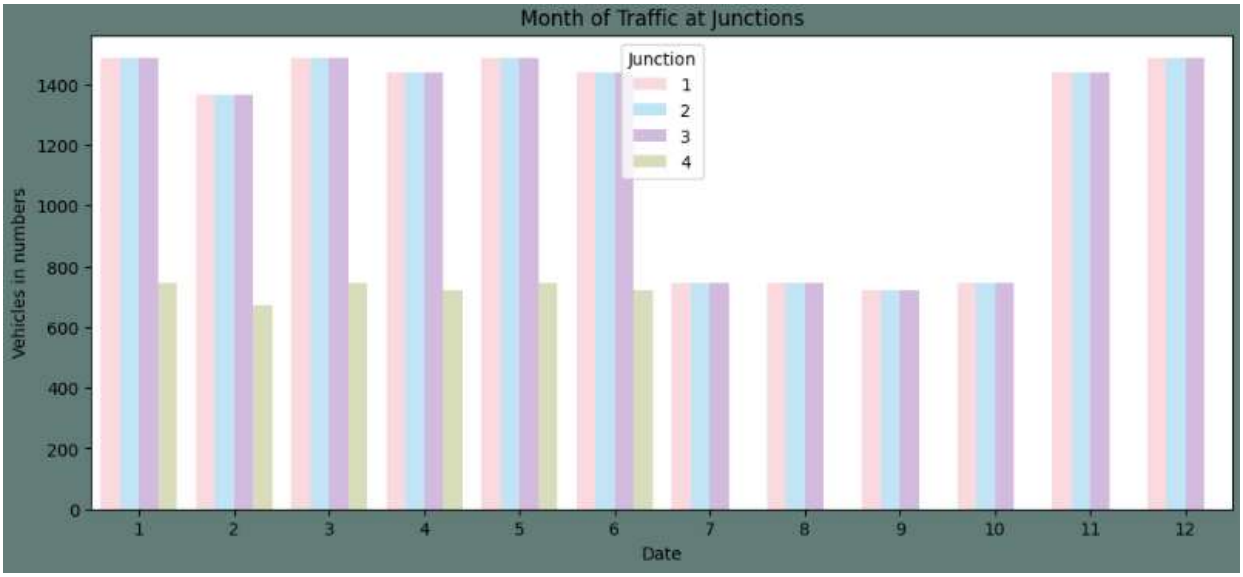
The year, month, date, hour, and day have been segregated on all four junctions to see the data.

2. Exploratory Data analysis

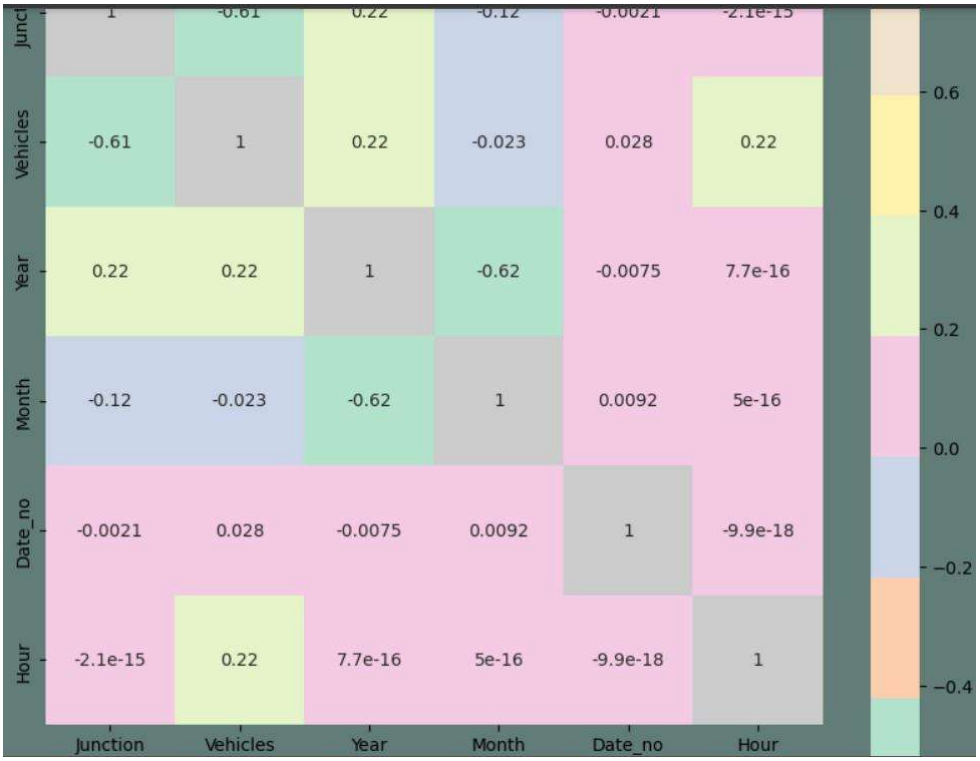
```
new_features = [ "Year", "Month", "Date_no", "Hour", "Day"]


for i in new_features:
  plt.figure(figsize=(10,2),facecolor="#627D78")
  ax=sns.lineplot(x=df[i],y="Vehicles",data=df, hue="Junction", palette=colors )
  plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```



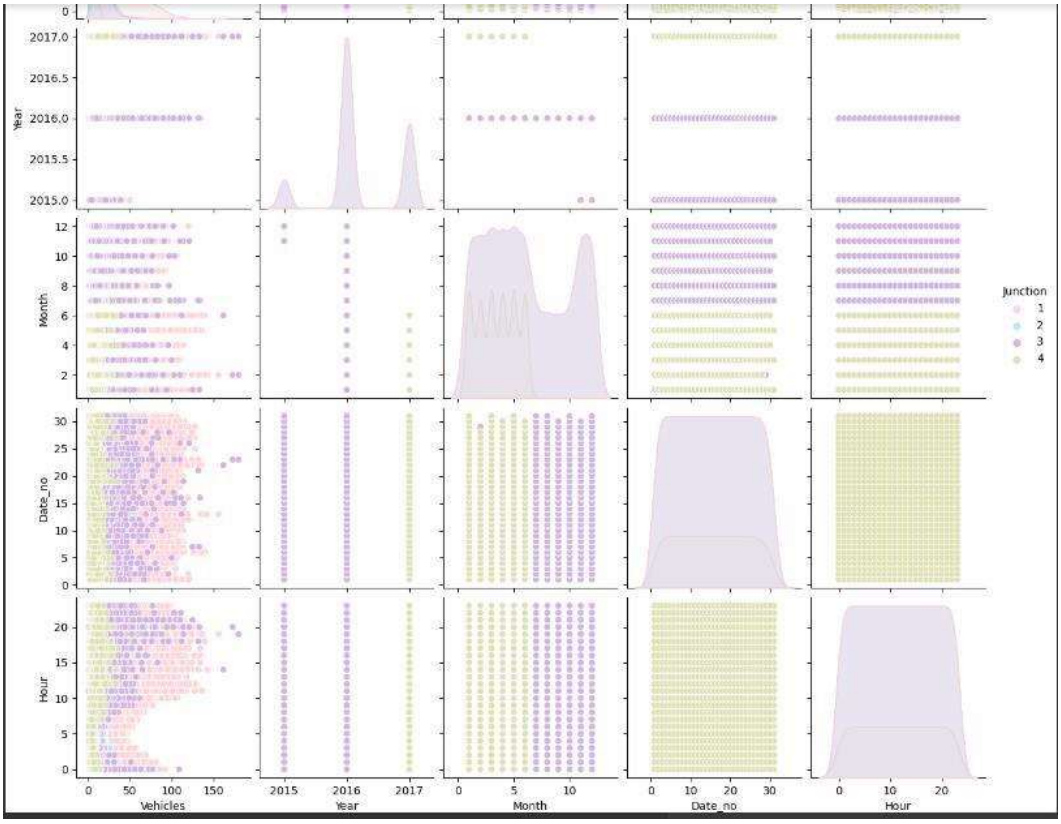
```
plt.figure(figsize=(12,5),facecolor="#627D78")
count = sns.countplot(data=df, x =df["Month"], hue="Junction", palette=colors)
count.set_title("Month of Traffic at Junctions")
count.set_ylabel("Vehicles in numbers")
count.set_xlabel("Date")
```



3. I used a **correlation matrix** to know the strength and direction of the linear (straight-line) association between two quantitative variables. Denoted by r, it takes values between -1 and +1. A positive value for r indicates a positive association and a negative value for r indicates a negative association. The closer r is to 1 the closer the data points fall to a straight line, thus, the linear association is stronger.



4. SNSPlot: A count plot can be thought of as a histogram across a categorical, instead of quantitative, variable.



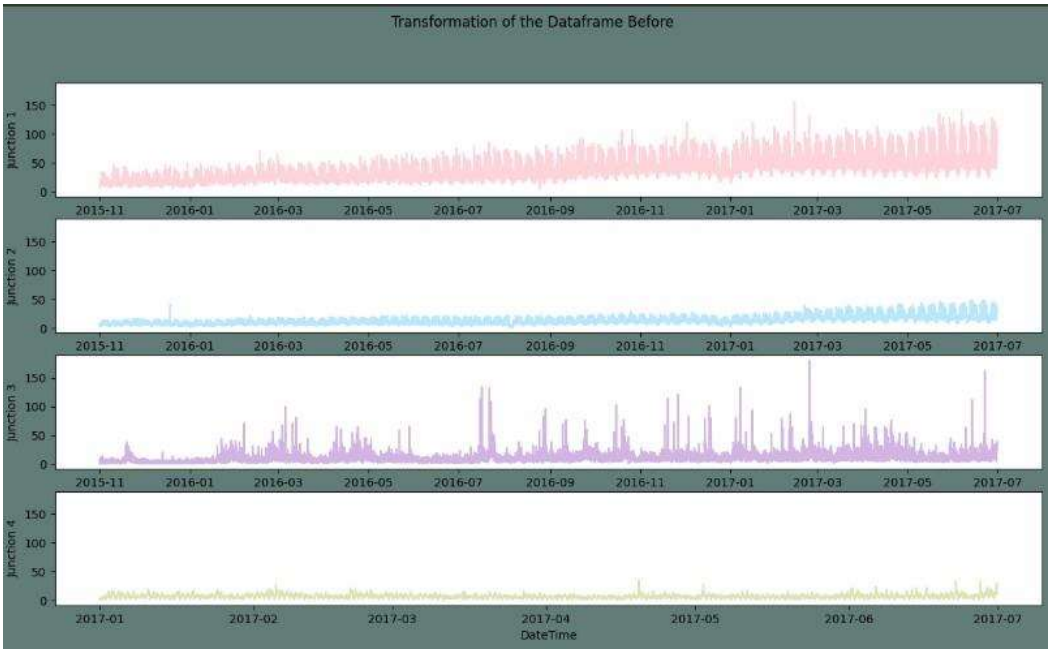
5. Data Transformation and Preprocessing

We shall proceed in the following order for this step:

- At each junction, make unique frames and chart them
- Plotting the series and changing it
- Using the Augmented Dickey-Fuller test to determine if converted series are seasonal
- Making training and test sets.

```
dataframe_1 = dataframe_junction[[('Vehicles', 1)]]

dataframe_2 = dataframe_junction[[('Vehicles', 2)]]

dataframe_3 = dataframe_junction[[('Vehicles', 3)]]

dataframe_4 = dataframe_junction[[('Vehicles', 4)]]

dataframe_4 = dataframe_4.dropna() #For only a few months, Junction 4 has only had minimal data.
```

```
 # As DFS's data frame contains many indices, its index is lowering level one.
list_dfs = [dataframe_1, dataframe_2, dataframe_3, dataframe_4]
for i in list_dfs:
   i.columns= i.columns.droplevel(level=1)
def Sub_Plots4(dataframe_1, dataframe_2,dataframe_3,dataframe_4,title):
   fig, axes = plt.subplots(4, 1, figsize=(15, 8),facecolor="#627D78", sharey=True)
   fig.suptitle(title)
   #J1
   pl_1=sns.lineplot(ax=axes[0],data=dataframe_1,color=colors[0])
   #pl_1=plt.ylabel()
   axes[0].set(ylabel ="Junction 1")
   #J2
   pl_2=sns.lineplot(ax=axes[1],data=dataframe_2,color=colors[1])
   axes[1].set(ylabel ="Junction 2")
#J3
   pl_3=sns.lineplot(ax=axes[2],data=dataframe_3,color=colors[2])
   axes[2].set(ylabel ="Junction 3")
   #J4
   pl_4=sns.lineplot(ax=axes[3],data=dataframe_4,color=colors[3])
   axes[3].set(ylabel ="Junction 4")
Sub_Plots4(dataframe_1.Vehicles,
dataframe_2.Vehicles,dataframe_3.Vehicles,dataframe_4.Vehicles,"Transformation of the
Dataframe Before")
```



6. I **normalized and differenced** the data of the four junctions so that they can be compared on a common ground:

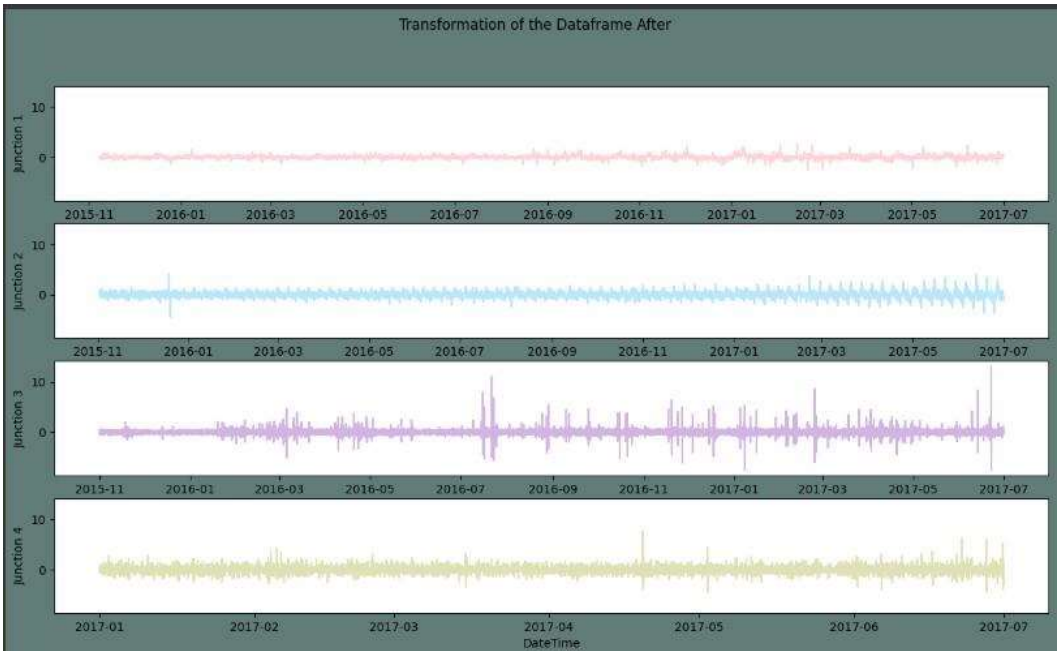
Junction1: on a daily basis

Junction2: on a daily basis

Junction3:on an hourly basis

Junction4: on an hourly basis

Sub_Plots4(dataframe_N1.Diff, dataframe_N2.Diff,dataframe_N3.Diff,dataframe_N4.Diff,"Transformation of the Dataframe After")



This was before I decided to switch my model. The graph above are implemented on training dataset.

Thanks to all the staff members at upskill and UCT for bringing this opportunity and my fellow peers in this internship without whom this project would have not been possible. My university library gives me a space so that I can code till dusk. all these people and things have helped me and been a part of my journey directly or indirectly.

My advice to my juniors is whatever you have in mind is a thing you should believe in and persevere hard for. UCT gave me this opportunity that I recommend all of you to take part in and take advantage of.

2 Introduction

2.1 About UniConverge Technologies Pvt Ltd

A company established in 2013 and working in Digital Transformation domain and providing Industrial solutions with prime focus on sustainability and RoI.

For developing its products and solutions it is leveraging various **Cutting Edge Technologies e.g. Internet of Things (IoT), Cyber Security, Cloud computing (AWS, Azure), Machine Learning, Communication Technologies (4G/5G/LoRaWAN), Java Full Stack, Python, Front end** etc.

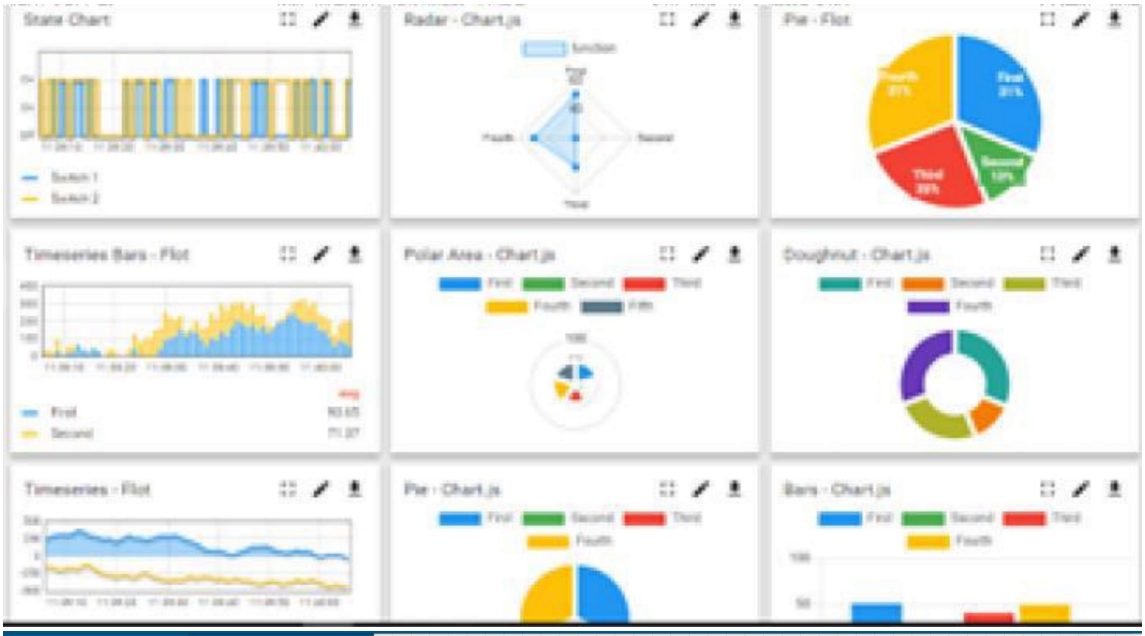
i. UCT IoT Platform (UCT Insight)

UCT Insight is an IOT platform designed for quick deployment of IOT applications on the same time providing valuable "insight" for your process/business. It has been built in Java for backend and ReactJS for Front end. It has support for MySQL and various NoSql Databases.

- It enables device connectivity via industry standard IoT protocols - MQTT, CoAP, HTTP, Modbus TCP, OPC UA
- It supports both cloud and on-premises deployments.

It has features to

- Build Your own dashboard
- Analytics and Reporting
- Alert and Notification
- Integration with third party application(Power BI, SAP, ERP)
- Rule Engine



ii. Smart Factory Platform (FACTORY WATCH)

Factory watch is a platform for smart factory needs.

It provides Users/ Factory

- with a scalable solution for their Production and asset monitoring
- OEE and predictive maintenance solution scaling up to digital twin for your assets.
- to unleased the true potential of the data that their machines are generating and helps to identify the KPIs and also improve them.
- A modular architecture that allows users to choose the service that they what to start and then can scale to more complex solutions as per their demands.

Its unique SaaS model helps users to save time, cost and money.

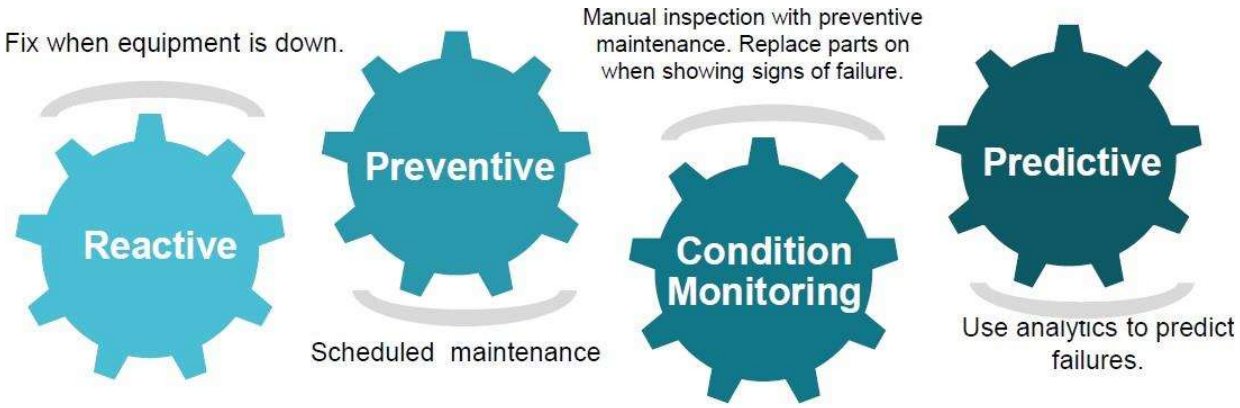
Machine	Operator	Work Order ID	Job ID	Job Performance	Job Progress		Output		Rejection	Time (mins)				Job Status	End Customer
					Start Time	End Time	Planned	Actual		Setup	Pred	Downtime	Idle		
CNC_S7_81	Operator 1	WO0405200001	4168	58%	10:30 AM		55	41	0	80	215	0	45	In Progress	i
CNC_S7_81	Operator 1	WO0405200001	4168	58%	10:30 AM		55	41	0	80	215	0	45	In Progress	i

iii. LoRaWAN based Solution

UCT is one of the early adopters of LoRAWAN teschnology and providing solution in Agritech, Smart cities, Industrial Monitoring, Smart Street Light, Smart Water/ Gas/ Electricity metering solutions etc.

iv. Predictive Maintenance

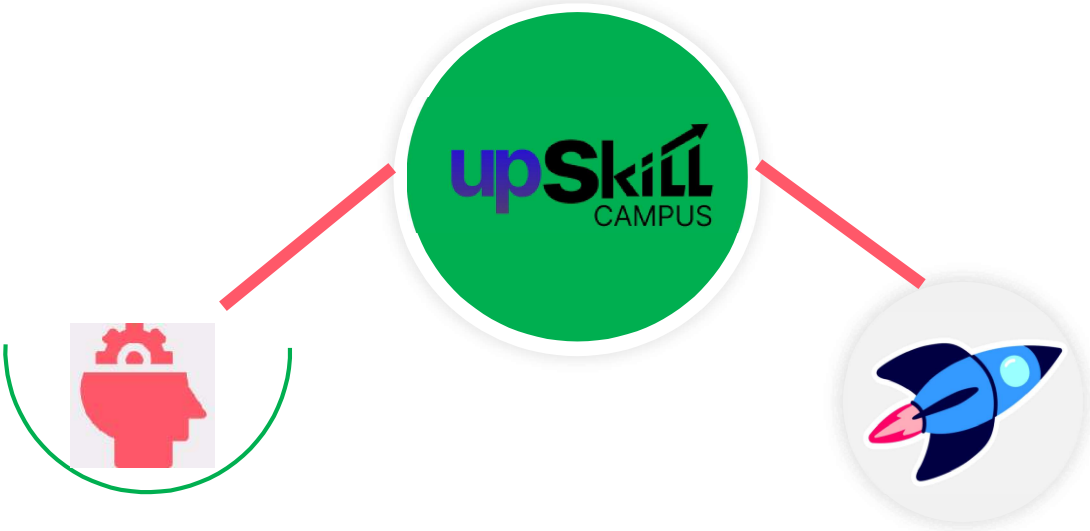
UCT is providing Industrial Machine health monitoring and Predictive maintenance solution leveraging Embedded system, Industrial IoT and Machine Learning Technologies by finding Remaining useful life time of various Machines used in production process.



2.2 About upskill Campus (USC)

upskill Campus along with The IoT Academy and in association with Uniconverge technologies has facilitated the smooth execution of the complete internship process.

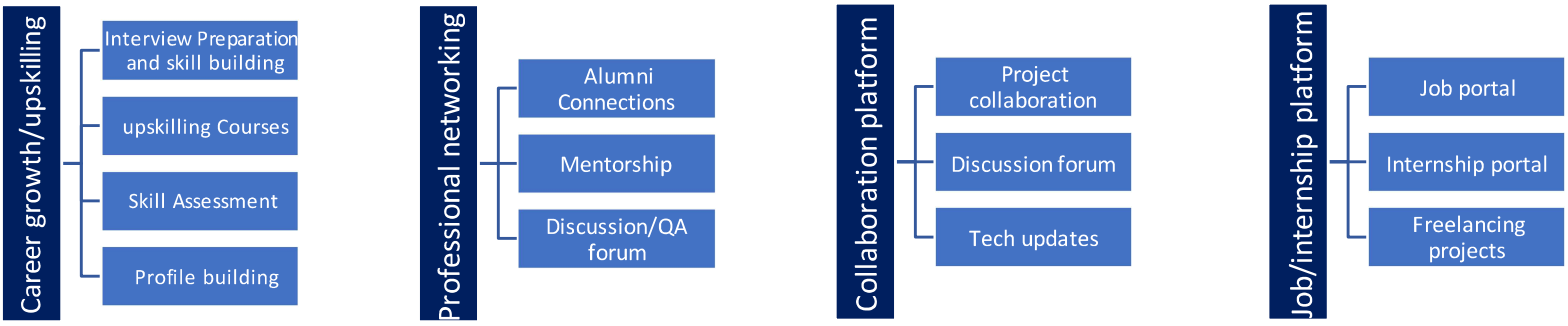
USC is a career development platform that delivers **personalized executive coaching** in a more affordable, scalable and measurable way.



Seeing need of upskilling in self paced manner along-with additional support services e.g. Internship, projects, interaction with Industry experts, Career growth Services

upSkill Campus aiming to upskill 1 million learners in next 5 year

https://www.upskillcampus.com/



2.3 The IoT Academy

The IoT academy is EdTech Division of UCT that is running long executive certification programs in collaboration with EICT Academy, IITK, IITR and IITG in multiple domains.

2.4 Objectives of this Internship program

The objective for this internship program was to

- ☛ get practical experience of working in the industry.
- ☛ to solve real world problems.
- ☛ to have improved job prospects.
- ☛ to have Improved understanding of our field and its applications.
- ☛ to have Personal growth like better communication and problem solving.

2.5 Reference

[1] https://www.freecodecamp.org/learn/machine-learning-with-python/tensorflow/introduction-machine-learning-fundamentals

[2] https://www.javatpoint.com/traffic-prediction-using-machine-learning

[3] https://github.com/mratsim/McKinsey-SmartCities-Traffic-Prediction

2.6 Glossary

Terms	Acronym

RNN	Recurrent Neural Network is a type of Neural Network where the output from the previous step is fed as input to the current step
GRU	GRU or Gated Recurrent Unit Networks are a type of RNN that use certain gating mechanisms to selectively update the hidden state at each time step.
Neural Networks	They are the subset of machine learning and work in a similar fashion how neurons send signal to one another.

3 Problem Statement

You are working with the government to transform your city into a smart city. The vision is to convert it into a digital and intelligent city to improve the efficiency of services for the citizens. One of the problems faced by the government is traffic. You are a data scientist working to manage the traffic of the city better and to provide input on infrastructure planning for the future.

The government wants to implement a robust traffic system for the city by being prepared for traffic peaks. They want to understand the traffic patterns of the four junctions of the city. Traffic patterns on holidays, as well as on various other occasions during the year, differ from normal working days. This is important to take into account for your forecasting.

The sensors on each of these junctions were collecting data at different times, hence I would see traffic data from different periods. To add to the complexity, some of the junctions have provided limited or sparse data requiring thoughtfulness when creating future projections. Depending on the historical data of 20 months, the government is looking to deliver accurate traffic projections for the coming four months. My algorithm will become the foundation of a larger transformation to make the city smart and intelligent.

4 Existing and Proposed solution

- **Autoregressive Integrated Moving Average (ARIMA):** ARIMA models are widely used for time series forecasting, including traffic prediction. They capture the temporal patterns in the data using autoregressive (AR), moving average (MA), and differencing (I) components. However, ARIMA models assume linear relationships and have limited ability to capture complex non-linear patterns in traffic data.
- **Long Short-Term Memory (LSTM) Networks**: LSTM networks are a type of recurrent neural network (RNN) that can capture long-term dependencies in sequential data. They have been successfully applied to traffic prediction by modeling the temporal patterns in traffic data. However, LSTM models may struggle with capturing complex spatial dependencies or considering external factors that influence traffic.
- **Convolutional Neural Networks (CNN):** CNNs have been applied to traffic prediction by treating traffic data as images, where each pixel represents a traffic attribute at a specific location. CNNs can capture spatial dependencies in the data and have shown promising results. However, CNN models may overlook temporal patterns or struggle with long-term forecasting.
- **Graph Neural Networks (GNN):** GNNs are designed to model relationships in graph-structured data, such as road networks. They can capture the spatial dependencies between different road segments and nodes. GNNs have shown potential in traffic prediction, but their performance can be limited when dealing with large-scale road networks or dynamically changing traffic conditions.
- **Ensemble Models:** Ensemble models combine predictions from multiple individual models to improve overall performance. They leverage the diversity of different models to enhance prediction accuracy. However, creating effective ensemble models requires careful selection and combination of individual models, which can be complex and time-consuming.
- **Hybrid Models:** Hybrid models combine multiple techniques or models to overcome the limitations of individual approaches. For example, combining LSTM with attention mechanisms, or integrating external factors such as weather data or event information. Hybrid models can improve prediction accuracy, but they may also introduce additional complexity in model design and training.

Limitations

Limitations across these models can include the need for extensive feature engineering, challenges in handling missing or noisy data, difficulties in capturing non-linear or dynamic patterns, sensitivity to hyperparameter tuning, and limited generalization to new or unseen traffic conditions. Furthermore, scalability and computational efficiency can be challenging, especially for large-scale traffic networks.

My proposed solution:

Using RNN for model building, Keras for fast prototyping and Tensorflow for Backend.

Future value addition

Using pytorch and optimising seq2seq.

4.1 Code submission (Github link)

https://github.com/emojivibe/upskill_campus_files/blob/main/code/Urvashi_pred_using_NN.ipynb

4.2 Report submission (Github link) :.

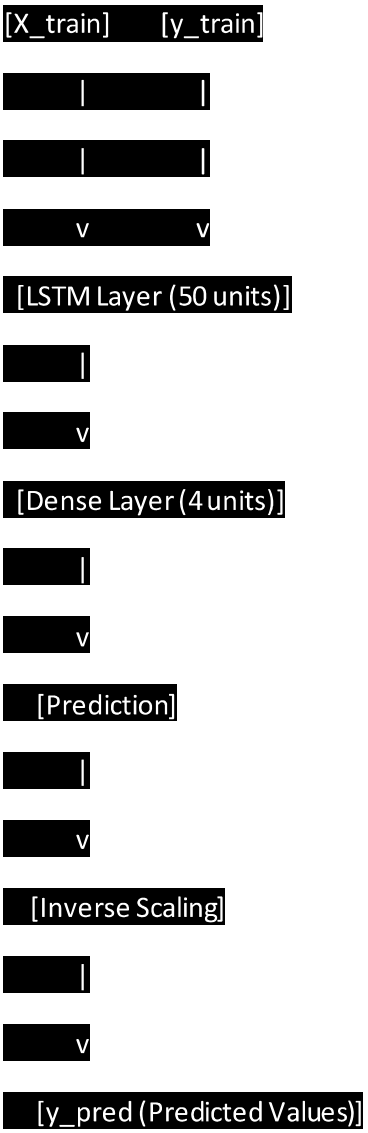
https://github.com/emojivibe/upskill_campus_files

5 Proposed Design/ Model

The proposed solution has the following part:

1. Data Preparation: The initial steps involve loading and preprocessing the data. This includes reading the training data from a CSV file, performing any necessary data transformations, and splitting the data into training and validation sets.
2. Feature Engineering: Lag features are generated using the gen_lag_features function, which creates lagged observations based on the specified number of input (X) and forecast (y) observations. This step helps incorporate temporal dependencies into the model by considering past observations as input features.
3. Data Scaling: The MinMaxScaler from sklearn. preprocessing is used to scale the data between the range of 0 and 1. Scaling the data helps ensure that all features have a similar scale, which can improve the training performance of the model.
4. Model Architecture: The model architecture is defined using the Keras Sequential API. The Sequential model allows for building a linear stack of layers. In this case, an LSTM layer with 50 units and ReLU activation is added as the input layer, followed by a dense layer with 4 units corresponding to the output for 4 junctions.
5. Model Compilation: The model is compiled by specifying the optimizer (adam) and the loss function (root_mean_squared_error). The adam optimizer is widely used for gradient-based optimization, and the custom loss function root_mean_squared_error is utilized to measure the performance of the model.
6. Model Training: The model is trained using the fit() method, passing the training data (X_train and y_train). The training process involves iterating over the training data for a specified number of epochs, and updating the model's weights using backpropagation and gradient descent.
7. Prediction: Once the model is trained, it is used to make predictions on the validation data (X_valid) using the predict() method. The predictions (y_pred) represent the model's estimated values for the target variable.
8. Inverse Scaling: The predicted values (y_pred) are inverse scaled using the scaler.inverse_transform() method to bring them back to their original scale.
9. Evaluation: The model's performance is evaluated by calculating the root mean squared error (RMSE) between the true values (y_truth) and the predicted values (y_pred). The RMSE provides a measure of the average difference between the predicted and true values, with lower values indicating better performance.
10. Visualization: The true and predicted traffic values are plotted for each junction using Matplotlib, allowing for visual comparison and analysis of the model's performance.

5.1 High Level Diagram (if applicable)



The input data X_train is fed into an LSTM layer with 50 units, which is a type of recurrent neural network (RNN) layer capable of capturing sequential dependencies in the data.

The output from the LSTM layer is passed to a dense layer with 4 units, representing the predicted values for 4 junctions.

The model then generates predictions based on the output of the dense layer.

The predictions are then subjected to inverse scaling to transform them back to their original scale.

The resulting y_pred represents the predicted values for the target variable.

Figure 1: HIGH-LEVEL DIAGRAM OF THE SYSTEM

5.2 Low-Level Diagram (if applicable)

A multi-attention RNN model typically consists of the following components:

- Input Layer: This layer receives the input data, which could be a sequence of features or time-series data.
- Recurrent Layer(s): These layers, often LSTM or GRU layers, capture the sequential dependencies in the input data. They recurrently process the input sequences, retaining memory of past information.
- Attention Mechanism: The attention mechanism allows the model to focus on different parts of the input sequence selectively. It assigns importance weights to different time steps or features based on their relevance to the task at hand. A multi-attention model typically involves multiple attention heads that attend to different aspects of the input.
- Fusion Layer: This layer combines the outputs of the attention heads, allowing the model to integrate information from multiple attention mechanisms.
- Output Layer: This layer produces the final predictions or outputs of the model, which could be regression values, class probabilities, or other relevant outputs for the task.

5.3 Interfaces (if applicable)

The block diagram for the LSTM-based regression model would typically consist of the following components:

- Input Layer: The input layer represents the input data, which in this case is a time series sequence. It receives the input data in the shape of (batch_size, time_steps, features).
- LSTM Layer: The LSTM layer is the core component of the model. It consists of a series of LSTM units that capture the temporal dependencies in the input sequence. Each LSTM unit processes the input sequence step by step and updates its internal states.
- Dense Layer: After the LSTM layer, a dense layer is added. This layer is a fully connected layer that receives the outputs from the LSTM layer and applies a linear transformation to produce the final output. In this case, the dense layer has units=4 to match the number of junctions in the output.
- Output Layer: The output layer represents the final output of the model. It produces the predicted values for each junction.

Overall, the LSTM-based regression model takes the input time series sequence, processes it through the LSTM layer, and then passes the outputs to the dense layer and the output layer to generate the final predictions.

6 Performance Test

For predictions, we will have to do a different pipeline since we don't have the previous hour traffic like in validation.

- We predict step by step and the previous prediction is the input of the next one
- We create 6 months min lag features
- We predict 3 months of traffic at each point (rolling forecast)

Features used:

- lag features,
- keras model- sequential,
- keras layers- Dense, LSTM,
- initializers- he_normal
- Regressor =sequential,

The various constraints of a neural network:

- Memory: Memory refers to the system's storage capacity required to store the neural network's parameters, intermediate activations, and other data structures during training and inference. In an ideal neural network, memory would be efficiently utilized to minimize memory footprint and optimize performance.
- MIPS: MIPS stands for Millions of Instructions Per Second. In the context of an ideal neural network, MIPS represents the computational power of the underlying hardware on which the network is running. An ideal neural network would have access to a high MIPS processor capable of executing a large number of instructions per second, enabling faster training and inference times.
- Accuracy: Accuracy is a performance metric used to evaluate the effectiveness of a machine learning model. In the context of an ideal neural network, accuracy refers to the ability of the network to make correct predictions or decisions on a given dataset. An ideal neural network would achieve high accuracy, accurately capturing patterns and relationships in the data it was trained on.
- Durability: Durability refers to the ability of a system or component to withstand stress, use, or degradation over time without experiencing failures or deterioration in performance. In the case of an ideal neural network, durability would imply that the network maintains its high performance and accuracy over extended periods, even when exposed to diverse datasets or changing environments.

- Power consumption: Power consumption refers to the amount of electrical power consumed by a system or device. In an ideal neural network, power consumption would be minimized to improve energy efficiency and reduce costs. An ideal neural network would be designed to efficiently utilize computational resources and optimize power consumption without sacrificing performance or accuracy.

It's important to note that an "ideal" neural network is a theoretical concept. In practice, neural networks are subject to various limitations, such as the availability of hardware resources, data quality and quantity, algorithmic complexity, and practical constraints. Researchers and engineers continuously strive to improve neural network architectures and training techniques to approach these ideal characteristics as closely as possible.

Solutions:

I should use a dynamic framework like PyTorch to share the state between the RNN. The model can capture fast change quite well. So stacking both + an xgboost model should improve the results a lot.

An alternative approach would be to use WaveNet and pure CNNs instead of RNNs

6.1 Test Plan/ Test Cases

Test Plan:

1. Data Preprocessing:

Test the preprocessing steps, such as data loading, handling missing values, and feature engineering.

Verify that the data is correctly transformed and prepared for model training.

2. Model Training:

Test the training process by verifying that the model can be successfully trained on the provided training data.

Evaluate the training time and resource usage to ensure feasibility.

3. Model Evaluation:

Perform evaluation metrics calculations, such as RMSE, to assess the model's performance.

Compare the model's performance against baseline models or industry standards.

4. Prediction Accuracy:

Generate predictions for a separate test dataset or unseen data.

Compare the predicted values against the ground truth to measure the model's accuracy.

5. Generalization:

Test the model's ability to generalize by evaluating its performance on new, unseen data.

Measure the model's robustness and consistency across different datasets.

Test Cases:

1. Data Loading:

Verify that the provided dataset can be successfully loaded into the model without any errors.

Check that the required columns and features are present in the loaded dataset.

2. Missing Data Handling:

Test the model's behavior when encountering missing data in the input dataset.

Assess how the model handles missing values during training and prediction.

3. Model Training Convergence:

Validate that the model training process converges and completes without errors.

Monitor and analyze the training loss to ensure it decreases over iterations.

4. Prediction Accuracy:

Generate predictions for a subset of the training data and manually compare them against the ground truth.

Verify that the predicted values are within an acceptable range of the true values.

5. Model Persistence:

Save the trained model to disk and verify that it can be successfully reloaded for future use.

Ensure that the reloaded model produces the same results as the original trained model.

6. Performance and Scalability:

Test the model's performance with different batch sizes and evaluate its scalability.

Measure the model's training and prediction time for various dataset sizes.

6.2 Test Procedure

- **Data Preparation**: Prepare the test dataset that will be used to evaluate the model's performance. This dataset should be separate from the training dataset and representative of real-world data.
- **Data Preprocessing**: Apply the same preprocessing steps that were used for the training dataset to the test dataset. This ensures consistency and compatibility between the data used during training and testing.
- **Model Loading:** Load the trained model that was saved after training. Ensure that the model can be successfully loaded without any errors or issues.
- **Prediction:** Use the loaded model to make predictions on the test dataset. Pass the test dataset through the model and obtain the predicted values.
- **Inverse Scaling**: If necessary, perform inverse scaling on the predicted values to transform them back to their original scale or format. This step is required if the data was scaled or transformed before training the model.
- **Performance Evaluation:** Compare the predicted values obtained from the model with the true values in the test dataset. Calculate appropriate evaluation metrics, such as the root mean squared error (RMSE), to assess the model's performance. You can use the rmse(y_truth, y_pred) function mentioned earlier to calculate the RMSE.
- **Visualization and Analysis**: Visualize the predicted values and the true values for each junction or any other relevant visualization. Analyze the results to gain insights into the model's performance, such as identifying patterns or discrepancies between the predicted and true values.
- **Reporting and Iteration:** Document the test results, including the evaluation metrics, visualizations, and any observations or insights. If necessary, iterate on the model or experiment with different configurations to improve its performance based on the test results.

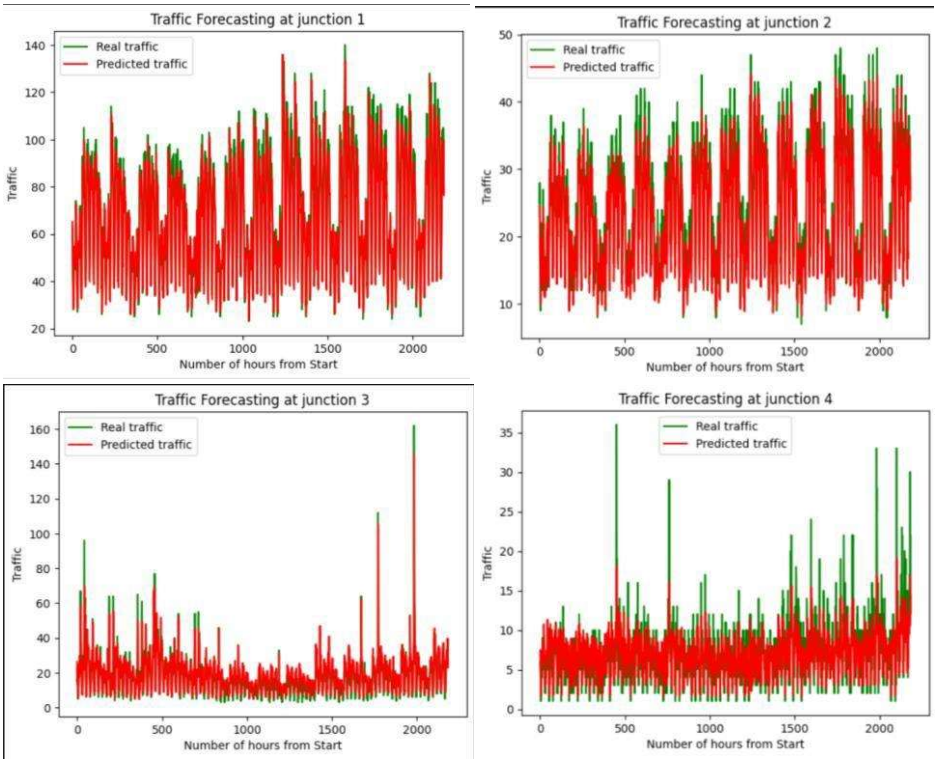
6.3 Performance Outcome

To determine the performance of the model, you can use the root mean squared error (RMSE) value obta ined from the rmse(y_truth, y_pred) function. The RMSE measures the average difference between the p redicted values and the true values, with lower values indicating better performance.

The model that I used has rmse as 5.987196690309687

7 My learnings

I learned the basic ins and outs of building a neural network. Though I already knew how to do feature engineering and fixing missing data. This internship provided me with an offer for better tricks to focus on. I liked building the model as it was very complex and I had to redo the entire thing many times if I had to do it right.



8 Future work scope

I should use a dynamic framework like PyTorch to share the state between the RNN. The model can capture fast change quite well. So stacking both + an xgboost model should improve the results a lot.

An alternative approach would be to use WaveNet and pure CNNs instead of RNNs