**Ideation Phase Innovation**

|  |  |
| --- | --- |
| Date | 10 October 2023 |
| Team ID | NM2023TM0448 |
| Project Name | 4123-Traffic Management System |
| Team Name | Proj\_227233\_Team\_1 |

**Consider integrating historical traffic data and machine learning algorithms to predict congestion patterns.**

Traffic prediction has always been a challenge for transportation planners and city managers. With the increasing growth of cities and the number of vehicles on the roads, the need for accurate and reliable traffic predictions has become more pressing. In recent years, machine learning has shown great promise in solving this problem.

Traffic prediction involves estimating the future behavior of traffic in a particular area. This information is useful for a variety of purposes, including **reducing congestion, optimizing transportation systems, and improving road safety**. In the past, traffic prediction has been based on traditional methods such as rule-based models and time-series analysis. However, these methods are often limited in their ability to capture the complexity and variability of traffic patterns.

Machine learning, on the other hand, is well-suited to handle large and complex datasets, making it an ideal tool for traffic prediction. Machine learning algorithms can automatically identify patterns and relationships in traffic data and use these to make predictions about future traffic conditions.

There are several types of machine learning algorithms that can be used for traffic prediction, including **regression, time-series analysis, and artificial neural networks**. Regression models use historical traffic data to predict future traffic conditions based on past trends. Time-series analysis models look at the patterns in traffic data over time and use these patterns to make predictions. Artificial neural networks, which are modeled on the structure of the human brain, are also commonly used for traffic prediction.

One of the key advantages of machine learning for traffic prediction is its ability to handle large and complex datasets. For example, traffic data may include information on traffic flow, vehicle speed, and traffic density, as well as other factors such as weather conditions, road conditions, and time of day. Machine learning algorithms can process this data and identify the most important factors that influence traffic patterns, making them ideal for traffic prediction.

Another advantage of machine learning for traffic prediction is its ability to adapt to changing conditions. Traditional traffic prediction methods are often limited in their ability to handle changes in traffic patterns, but machine learning algorithms can automatically adjust to these changes and continue to make accurate predictions.

In addition to these advantages, machine learning can also be used to improve the accuracy of traffic predictions by incorporating other sources of data, such as GPS data from vehicles, traffic cameras, and social media. For example, GPS data from vehicles can provide real-time information on traffic conditions, while traffic cameras can provide detailed information on traffic flow and density. Social media data, such as tweets about traffic conditions, can also be used to help improve the accuracy of traffic predictions.

While machine learning has many advantages for traffic prediction, it is not without its challenges. One of the biggest challenges is the **quality of the data used for training the machine learning algorithms.** For example, traffic data may be incomplete or inaccurate, and this can affect the accuracy of the predictions. Additionally, machine learning algorithms require large amounts of data to be effective, and this can be difficult to obtain in some cases.

Another challenge is the complexity of the algorithms used for traffic prediction. Machine learning algorithms can be difficult to understand and interpret, making it challenging to identify the factors that are driving the predictions. This can make it difficult to make changes to the algorithms or to improve their accuracy.

Now we will be exploring the dataset of four junctions and building a model to predict traffic on the same. This could potentially help in solving the traffic congestion problem by providing a better understanding of traffic patterns that will further help in building an infrastructure to eliminate the problem.

Traffic index is a conceptual measure of traffic congestion, with a value between 0 and 10. The higher the value, the more severe the traffic congestion is. As an advanced integrated traffic management system, intelligent transportation system can provide diversified services for traffic participants, which is the development direction of future transportation system. As a part of intelligent transportation system, traffic index prediction plays a positive role in promoting the development of intelligent transportation system.

## Code Implementation

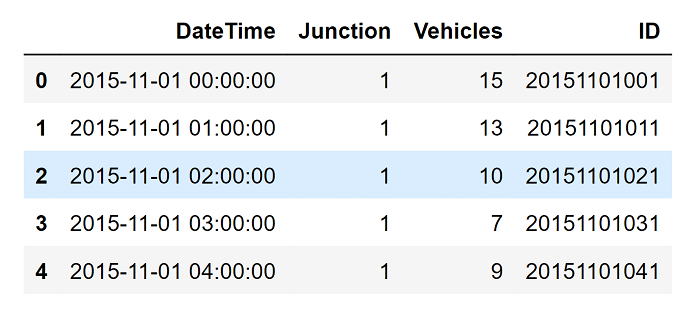
## Importing Libraries

|  |
| --- |
| **import** numpy as np  **import** pandas as pd  **import** matplotlib.pyplot as plt  **import** seaborn as sns  **import** datetime  **import** tensorflow  from statsmodels.tsa.stattools **import** adfuller  from sklearn.preprocessing **import** MinMaxScaler  from tensorflow **import** keras  from keras **import** callbacks  from tensorflow.keras **import** Sequential  from tensorflow.keras.layers **import** Conv2D, Flatten, Dense, LSTM, Dropout, GRU, Bidirectional  from tensorflow.keras.optimizers **import** SGD  **import** math  from sklearn.metrics **import** mean\_squared\_error    **import** warnings  warnings.filterwarnings("ignore") |

**Loading the Dataset**

|  |
| --- |
| dataset = pd.read\_csv("traffic.csv")  dataset.head() |

**Output:**



**About the Data**

This dataset is a compilation of hourly counts of automobiles at four intersections. There are four features in the CSV file:

* **DateTime**
* **Junctions**
* **Vehicles**
* **ID**

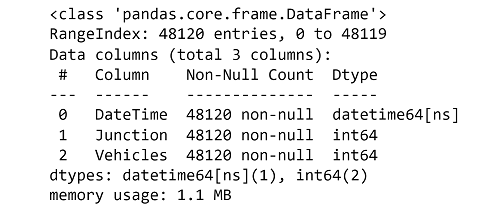
The traffic data comes from several time periods since the sensors on each of these intersections were gathering data at different times. Data from several of the intersections were scarce or restricted.

### Data Exploration

* Feature engineering for EDA
* Plotting time series
* Parsing dates

|  |
| --- |
| dataset["DateTime"]= pd.to\_datetime(dataset["DateTime"])  dataset = dataset.drop(["ID"], axis=1) #dropping IDs column  dataset.info() |

**Output:**



|  |
| --- |
| # dataframe to be used **for** EDA  dataframe=dataset.copy()    # Let's plot the Timeseries  colors = [ "#FFD4DB","#BBE7FE","#D3B5E5","#dfe2b6"]  plt.figure(figsize=(20,4),facecolor="#627D78")  Time\_series=sns.lineplot(x=dataframe['DateTime'],y="Vehicles",data=dataframe, hue="Junction", palette=colors)  Time\_series.set\_title("Years of Traffic at Junctions")  Time\_series.set\_ylabel("Vehicles in Number")  Time\_series.set\_xlabel("Date") |

**Output:**

Text(0.5, 0, 'Date')

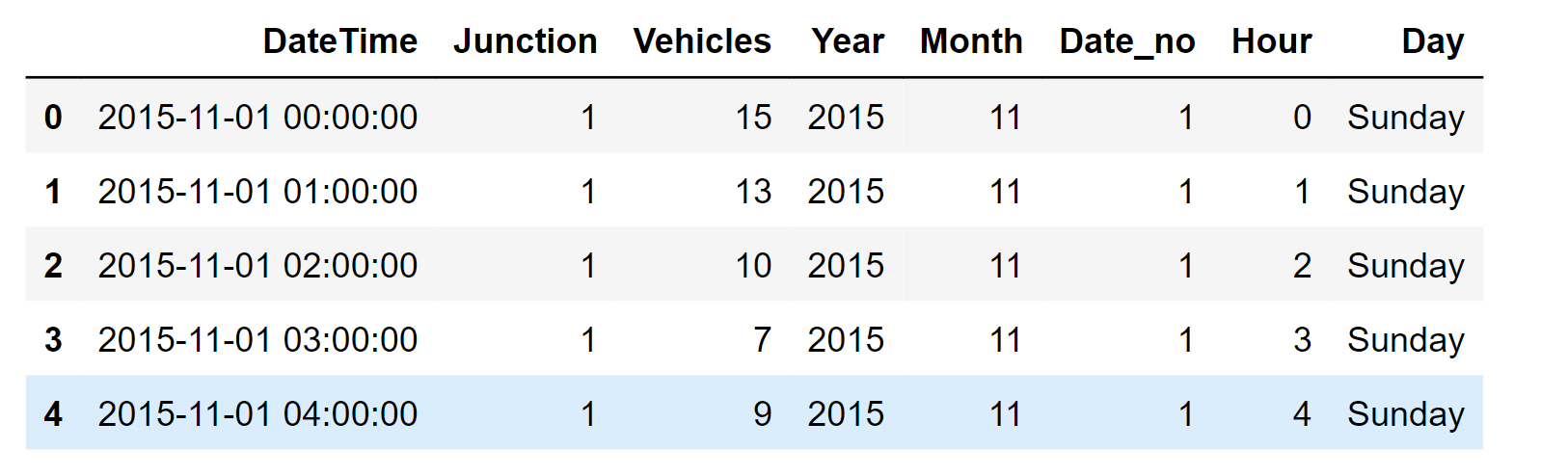
## Feature Engineering

At this stage, We are using DateTime to build a few additional functionalities. Namely:

* Year
* Month
* Date in the given month
* Days of week
* Hour

|  |
| --- |
| # Exploring more features  dataframe["Year"]= dataframe['DateTime'].dt.year  dataframe["Month"]= dataframe['DateTime'].dt.month  dataframe["Date\_no"]= dataframe['DateTime'].dt.day  dataframe["Hour"]= dataframe['DateTime'].dt.hour  dataframe["Day"]= dataframe.DateTime.dt.strftime("%A")  dataframe.head() |

**Output:**

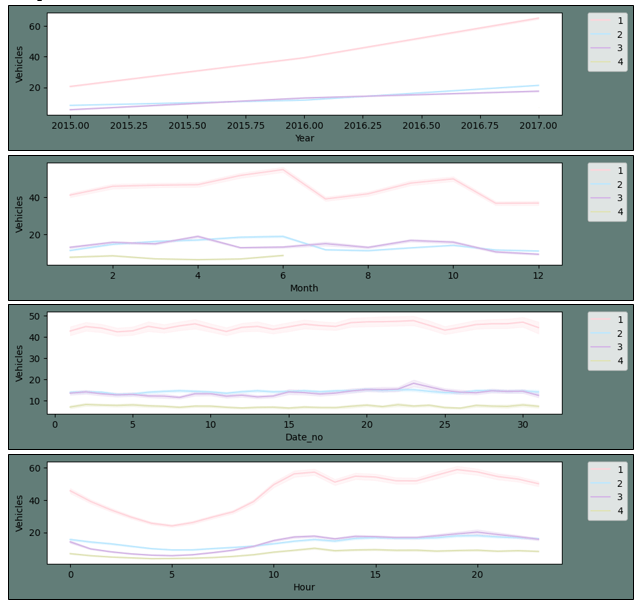


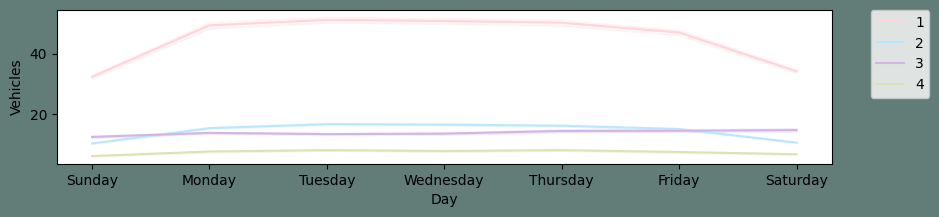
## Exploratory Data Analysis

The newly formed features are going to be plotted now.

|  |
| --- |
| # Let's plot the Timeseries  new\_features = [ "Year","Month", "Date\_no", "Hour", "Day"]    **for** i in new\_features:  plt.figure(figsize=(10,2),facecolor="#627D78")  ax=sns.lineplot(x=dataframe[i],y="Vehicles",data=dataframe, hue="Junction", palette=colors )  plt.legend(bbox\_to\_anchor=(1.05, 1), loc=2, borderaxespad=0.) |

**Output:**





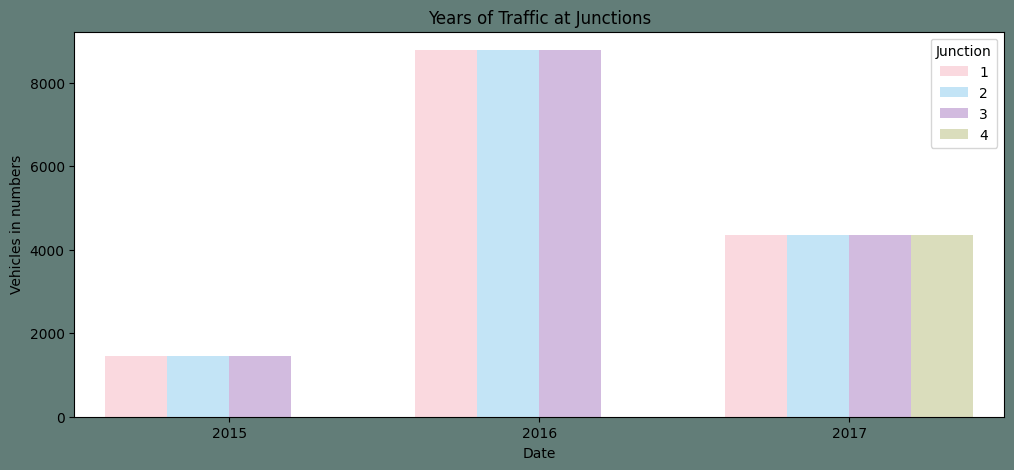
### The plot described above leads to the following conclusions:

* With the exception of the fourth junction, all junctions have shown a rising yearly tendency. As was already stated above, the fourth junction contains scant data that doesn't go back more than a year.
* We can observe that around June, there is an increase in traffic at the first and second crossroads. This, we assume, may be related to summer vacation and related activities.
* There is considerable consistency in the data on a monthly basis across all dates.
* We may observe that there are peaks in the morning and evening and a fall in activity throughout the night for a given
* day. This is what was predicted.
* Due to fewer automobiles on the roadways on Sundays than on other days of the week, traffic flows more smoothly. The traffic is consistent from Monday through Friday.

|  |
| --- |
| plt.figure(figsize=(12,5),facecolor="#627D78")  count = sns.countplot(data=dataframe, x =dataframe["Year"], hue="Junction", palette=colors)  count.set\_title("Years of Traffic at Junctions")  count.set\_ylabel("Vehicles in numbers")  count.set\_xlabel("Date") |

**Output:**

Text(0.5, 0, 'Date')



### Conclusions that We've Reached After This EDA:

* Each of the four intersections has a different range of data. Just 2017's data are available for the fourth junction.
* The annual trend for Junctions 1, 2, and 3 has varying slopes.
* The first junction has a stronger weekly seasonality than the other junctions.

For the aforementioned reasons, we believe junctions should be modified to suit each one's specific requirements.

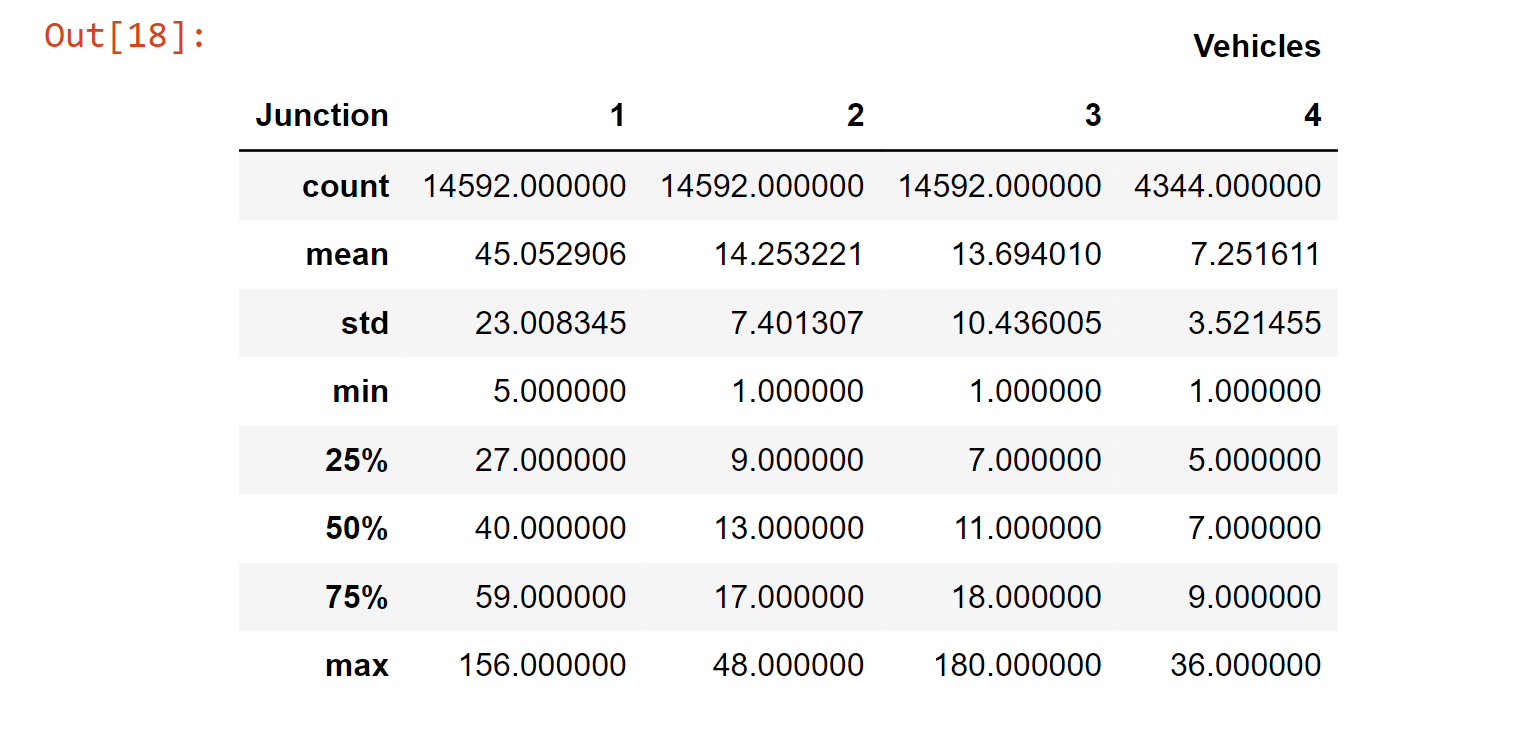
## Data Transformation and Preprocessing

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.

|  |
| --- |
| # Pivoting dataset from junction  dataframe\_junction = dataset.pivot(columns="Junction", index="DateTime")  dataframe\_junction.describe() |

**Output:**



If a time series lacks a pattern or seasonality, it is said to be stagnant. Nonetheless, we observed a weekly periodicity and an increased tendency in the EDA over time. It is once again clear from the graphic above that Junctions one and two are trending higher. We will be able to notice the weekly seasonality more clearly if we restrict the span. At this time, we will skip that step and continue with the appropriate dataset transformations.

## Preparing the data for the neural network

* Splitting the test train sets
* Assigning X as features and y as target
* Reshaping data for neural net

|  |
| --- |
| # Several NA values were produced as a result of differencing using a week's worth of data.  dataframe\_J1 = dataframe\_N1["Diff"].dropna()  dataframe\_J1 = dataframe\_J1.to\_frame()    dataframe\_J2 = dataframe\_N2["Diff"].dropna()  dataframe\_J2 = dataframe\_J2.to\_frame()    dataframe\_J3 = dataframe\_N3["Diff"].dropna()  dataframe\_J3 = dataframe\_J3.to\_frame()  dataframe\_J4 = dataframe\_N4["Diff"].dropna()  dataframe\_J4 = dataframe\_J4.to\_frame()    # Splitting the dataset  def Split\_data(dataframe):  training\_size = **int**(len(dataframe)\*0.90)  data\_len = len(dataframe)  train, test = dataframe[0:training\_size],dataframe[training\_size:data\_len]  train, test = train.values.reshape(-1, 1), test.values.reshape(-1, 1)  **return** train, test  # Splitting the training and test datasets  Junction1\_train, Junction1\_test = Split\_data(dataframe\_J1)  Junction2\_train, Junction2\_test = Split\_data(dataframe\_J2)  Junction3\_train, Junction3\_test = Split\_data(dataframe\_J3)  Junction4\_train, Junction4\_test = Split\_data(dataframe\_J4)    # Target and Feature  def target\_and\_feature(dataframe):  end\_len = len(dataframe)  X = []  y = []  steps = 32  **for** i in range(steps, end\_len):  X.append(dataframe[i - steps:i, 0])  y.append(dataframe[i, 0])  X, y = np.array(X), np.array(y)  **return** X ,y    # fixing the shape of X\_test and X\_train  def FeatureFixShape(train, test):  train = np.reshape(train, (train.shape[0], train.shape[1], 1))  test = np.reshape(test, (test.shape[0],test.shape[1],1))  **return** train, test    # Assigning features and target  X\_train\_Junction1, y\_train\_Junction1 = target\_and\_feature(Junction1\_train)  X\_test\_Junction1, y\_test\_Junction1 = target\_and\_feature(Junction1\_test)  X\_train\_Junction1, X\_test\_Junction1 = FeatureFixShape(X\_train\_Junction1, X\_test\_Junction1)    X\_train\_Junction2, y\_train\_Junction2 = target\_and\_feature(Junction2\_train)  X\_test\_Junction2, y\_test\_Junction2 = target\_and\_feature(Junction2\_test)  X\_train\_Junction2, X\_test\_Junction2 = FeatureFixShape(X\_train\_Junction2, X\_test\_Junction2)    X\_train\_Junction3, y\_train\_Junction3 = target\_and\_feature(Junction3\_train)  X\_test\_Junction3, y\_test\_Junction3 = target\_and\_feature(Junction3\_test)  X\_train\_Junction3, X\_test\_Junction3 = FeatureFixShape(X\_train\_Junction3, X\_test\_Junction3)  X\_train\_Junction4, y\_train\_Junction4 = target\_and\_feature(Junction4\_train)  x\_test\_Junction4, y\_test\_Junction4 = target\_and\_feature(Junction4\_test)  X\_train\_Junction4, x\_test\_Junction4 = FeatureFixShape(X\_train\_Junction4, x\_test\_Junction4) |

## Model Building

We have decided to employ a Gated Recurrent Unit for our project (GRU). We are developing a function in this part that the neural network may use to access and fit the data frames for all four junctions.

|  |
| --- |
| #Model **for** the prediction  def GRU\_model(X\_Train, y\_Train, X\_Test):  early\_stopping = callbacks.EarlyStopping(min\_delta=0.001,patience=10, restore\_best\_weights=True)    #The GRU model  model = Sequential()  model.add(GRU(units=150, return\_sequences=True, input\_shape=(X\_Train.shape[1],1), activation='tanh'))  model.add(Dropout(0.2))  model.add(GRU(units=150, return\_sequences=True, input\_shape=(X\_Train.shape[1],1), activation='tanh'))  model.add(Dropout(0.2))  model.add(GRU(units=50, return\_sequences=True, input\_shape=(X\_Train.shape[1],1), activation='tanh'))  model.add(Dropout(0.2))  model.add(GRU(units=50, return\_sequences=True, input\_shape=(X\_Train.shape[1],1), activation='tanh'))  model.add(Dropout(0.2))    model.add(GRU(units=50, input\_shape=(X\_Train.shape[1],1), activation='tanh'))  model.add(Dropout(0.2))  model.add(Dense(units=1))    # Compiling the model  model.compile(optimizer=SGD(decay=1e-7, momentum=0.9),loss='mean\_squared\_error')  model.fit(X\_Train,y\_Train, epochs=50, batch\_size=150,callbacks[early\_stopping])  pred\_GRU= model.predict(X\_Test)  **return** pred\_GRU    # To determine the root mean squared prediction error  def RMSE\_Value(test,predicted):  rmse = math.sqrt(mean\_squared\_error(test, predicted))  print("The root mean squared error is {}.".format(rmse))  **return** rmse    # Plotting the goal and forecast comparison plot  def PredictionsPlot(test,predicted,m):  plt.figure(figsize=(12,5),facecolor="#627D78")  plt.plot(test, color=colors[m],label="True Value",alpha=0.5 )  plt.plot(predicted, color="#627D78",label="Predicted Values")  plt.title("GRU Traffic Prediction Vs True values")  plt.xlabel("DateTime")  plt.ylabel("Number of Vehicles")  plt.legend()  plt.show() |

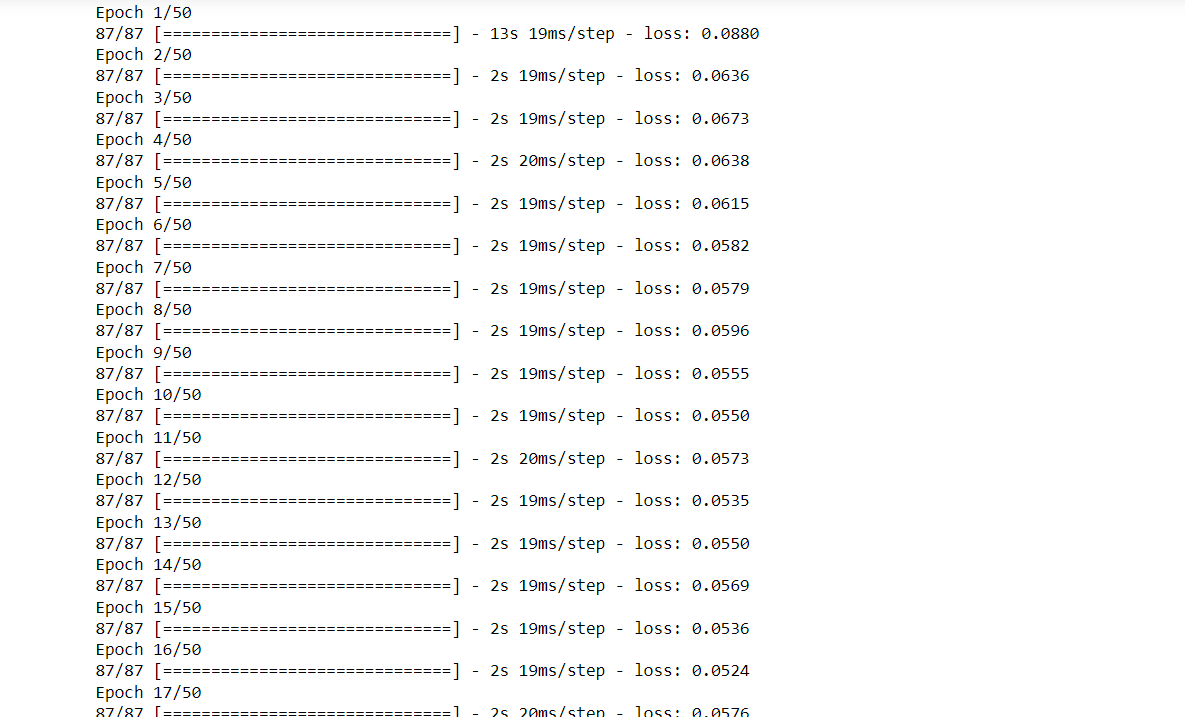
## Fitting the Model

We will now fit the four-joint training sets that have been changed to the constructed model and contrast them with the altered test sets.

**Plotting the predictions and test set while fitting the first junction**

|  |
| --- |
| #Predictions For First Junction  PredJ1 = GRU\_model(X\_train\_Junction1,y\_train\_Junction1,X\_test\_Junction1) |

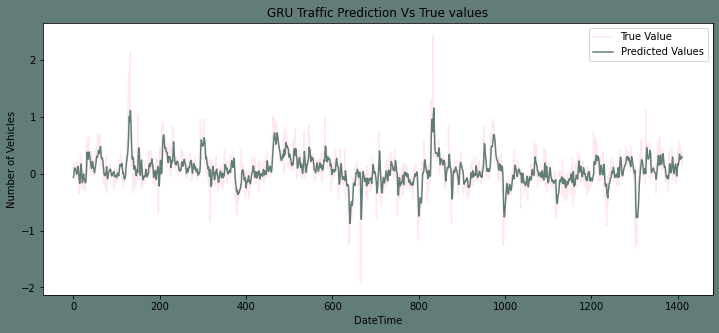
**Output:**



|  |
| --- |
| #Results **for** J1  RMSE\_J1=RMSE\_Value(y\_test\_Junction1,PredJ1)  PredictionsPlot(y\_test\_Junction1,PredJ1,0) |

**Output:**

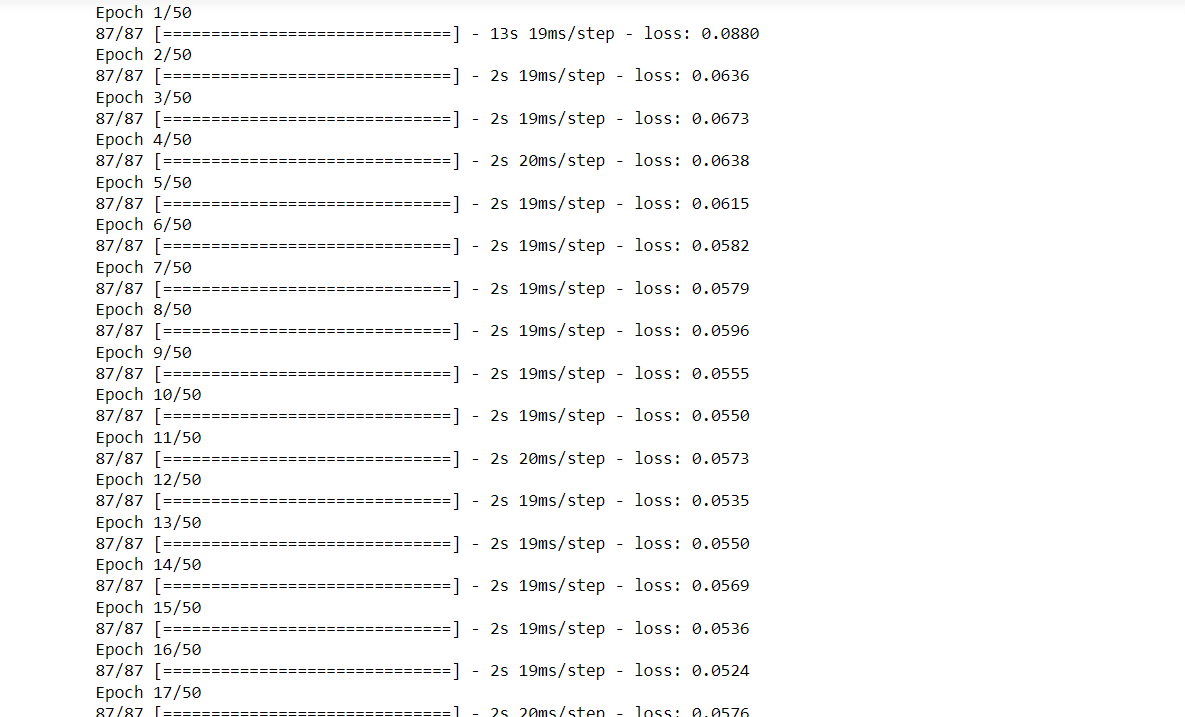
The root mean squared error is 0.245881146563882.



**Plotting the predictions and test set while fitting the second junction**

|  |
| --- |
| #Predictions For Second Junction  PredJ2 = GRU\_model(X\_train\_Junction2,y\_test\_Junction1,X\_test\_Junction2) |

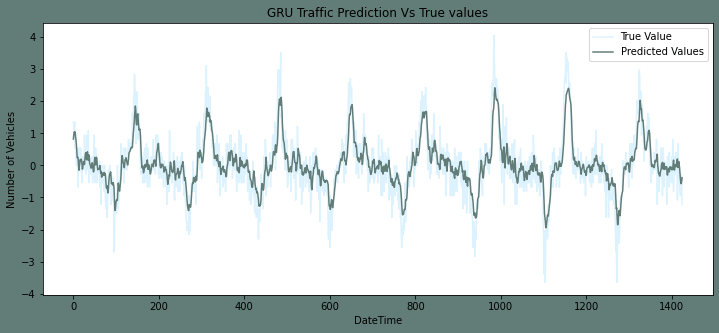
**Output:**



|  |
| --- |
| #Results **for** J2  RMSE\_J2=RMSE\_Value(y\_test\_Junction2,PredJ2)  PredictionsPlot(y\_test\_Junction2,PredJ2,1) |

**Output:**

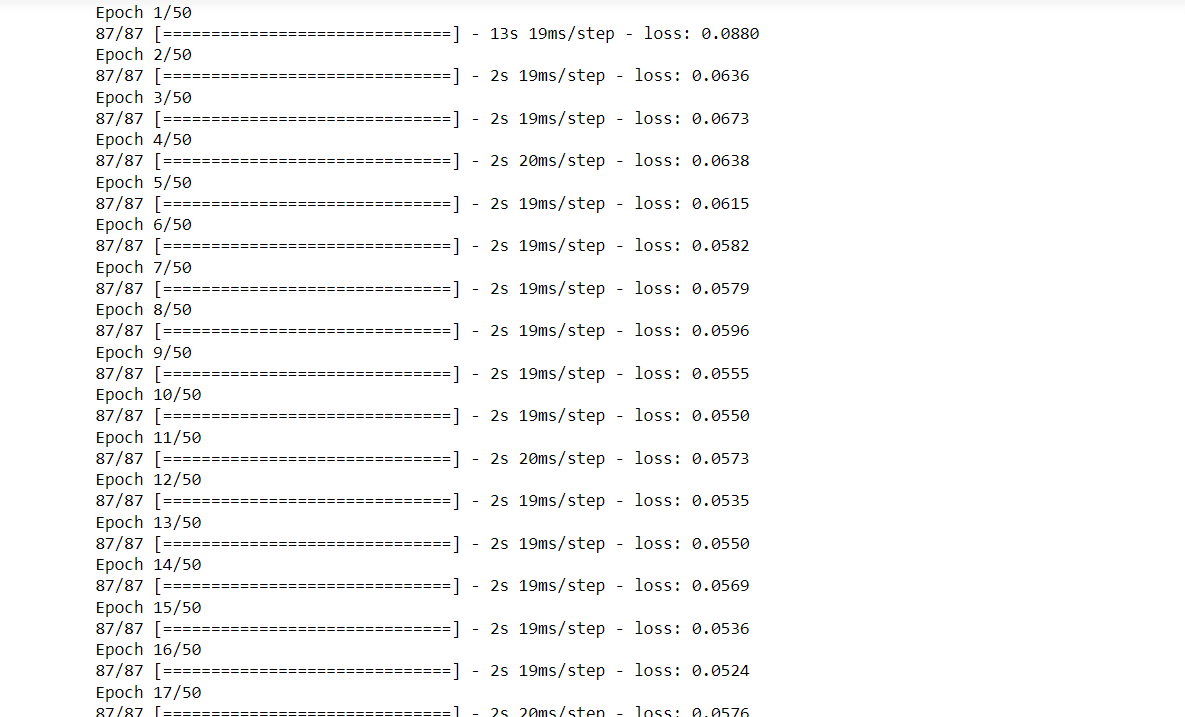
The root mean squared error is 0.5585970393765944.



**Plotting the predictions and test set while fitting the third junction**

|  |
| --- |
| 1. #Predictions For Third Junction 2. PredJ3 = GRU\_model(X\_train\_Junction3,y\_train\_Junction3,X\_test\_Junction3) |

**Output:**



|  |
| --- |
| #Results **for** J3  RMSE\_J3=RMSE\_Value(y\_test\_Junction3,PredJ3)  PredictionsPlot(y\_test\_Junction3,PredJ3,2) |

**Output:**

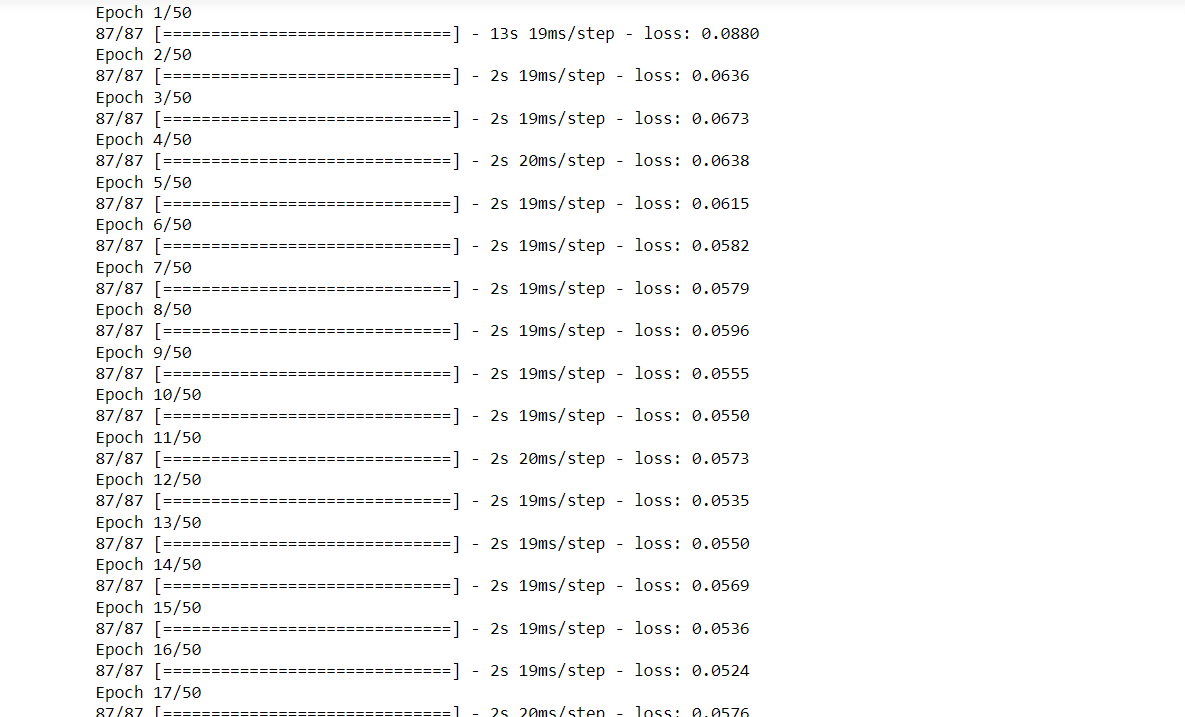
The root mean squared error is 0.6061366783632264.



### Plotting the predictions and test set while fitting the fourth junction

|  |
| --- |
| #Predictions For Forth Junction  PredJ4 = GRU\_model(X\_train\_Junction4,y\_train\_Junction4,x\_test\_Junction4) |

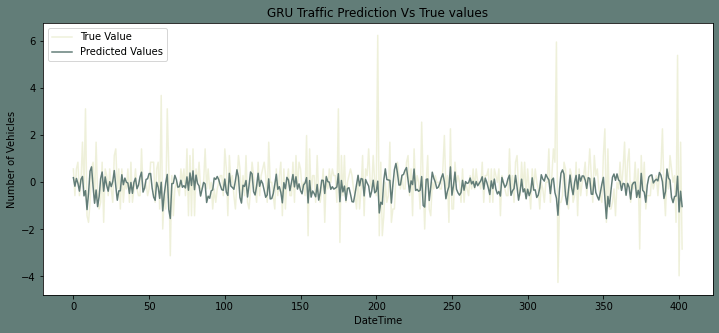
**Output:**



|  |
| --- |
| #Results **for** J4  RMSE\_J4=RMSE\_Value(y\_test\_Junction4,PredJ4)  PredictionsPlot(y\_test\_Junction4,PredJ4,3) |

**Output:**

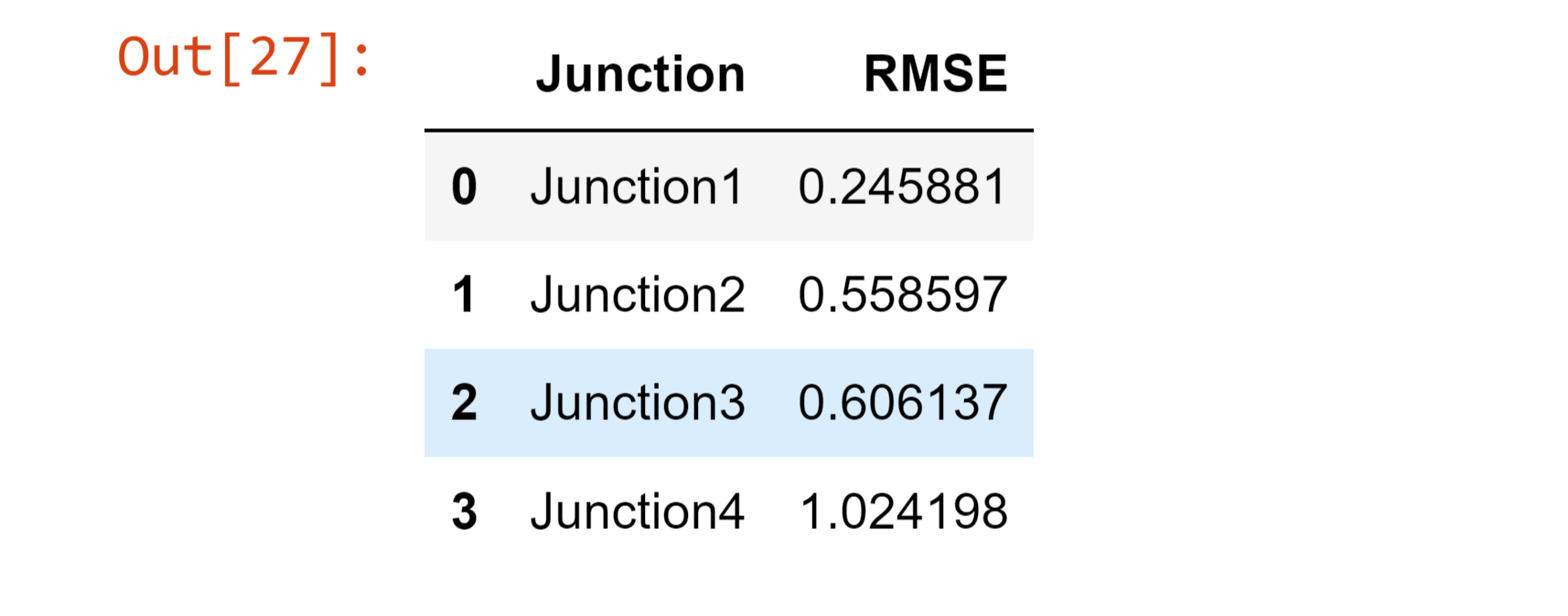
The root mean squared error is 1.0241982484501175.



## Results of the model

|  |
| --- |
| # Set the data in lists to the initial error values of the four junctions.  Junctions = ["Junction1", "Junction2", "Junction3", "Junction4"]  RMSE = [RMSE\_J1, RMSE\_J2, RMSE\_J3, RMSE\_J4]  list\_of\_tuples = list(zip(Junctions, RMSE))  # Creates pandas DataFrame.  Results = pd.DataFrame(list\_of\_tuples, columns=["Junction", "RMSE"])  Results.style.background\_gradient(cmap="Pastel1") |

**Output:**



## **Inversing the Transformation of the Data**

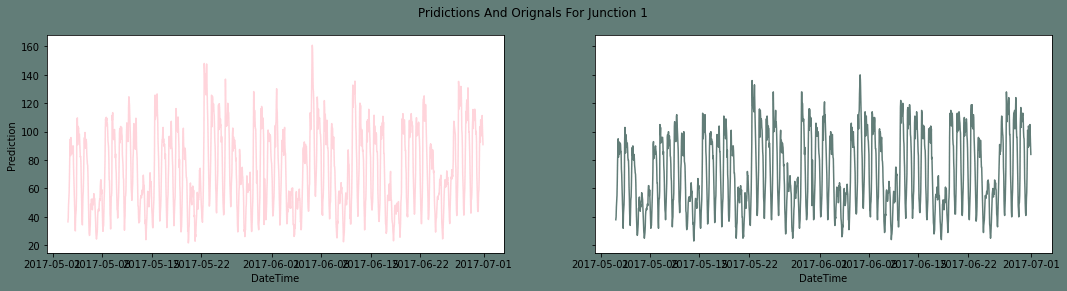
In this part, we will reverse the transformations we used to take the seasonality and trends out of the datasets. By carrying out this procedure, the forecasts will return to their previous level of accuracy.

|  |
| --- |
| # Functions to inverse transforms and plot comparative plots  # invert differenced forecast  def inverse\_difference(last\_ob, value):  inversed = value + last\_ob  **return** inversed    #Plotting the comparison  def Sub\_Plots2(df\_1, df\_2,title,m):  fig, axes = plt.subplots(1, 2, figsize=(18,4), sharey=True,facecolor="#627D78")  fig.suptitle(title)    pl\_1=sns.lineplot(ax=axes[0],data=df\_1,color=colors[m])  axes[0].set(ylabel ="Prediction")    pl\_2=sns.lineplot(ax=axes[1],data=df\_2["Vehicles"],color="#627D78")  axes[1].set(ylabel ="Orignal") |

**The first junction's inverse transform**

|  |
| --- |
| # invert the differenced forecast **for** Junction 1  recover1 = dataframe\_N1.Norm[-1412:-1].to\_frame()  recover1["Pred"]= PredJ1  Transform\_reverssed\_J1 = inverse\_difference(recover1.Norm, recover1.Pred).to\_frame()  Transform\_reverssed\_J1.columns = ["Pred\_Normed"]  #Invert the normalization J1  Final\_J1\_Pred = (Transform\_reverssed\_J1.values\* std\_J1) + avg\_J1  Transform\_reverssed\_J1["Pred\_Final"] =Final\_J1\_Pred  #Plotting the Predictions with originals  Sub\_Plots2(Transform\_reverssed\_J1["Pred\_Final"], dataframe\_1[-1412:-1],"Pridictions And Orignals For Junction 1", 0) |

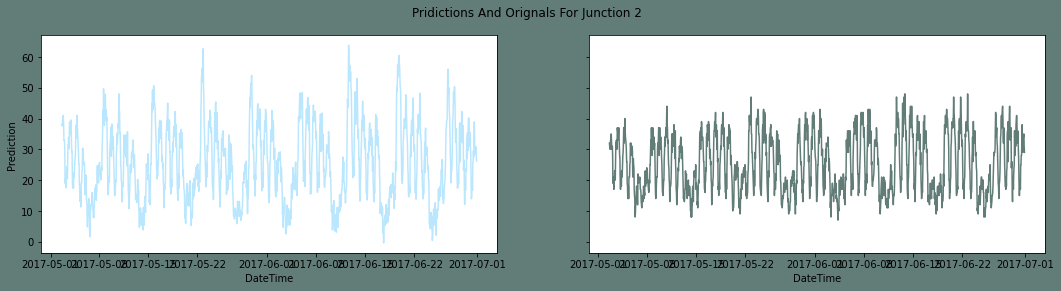
**Output:**



**The second junction's inverse transformation**

|  |
| --- |
| #Invert the differenced J2  recover2 = dataframe\_N2.Norm[-1426:-1].to\_frame() #len as per the diff  recover2["Pred"]= PredJ2  Transform\_reverssed\_J2 = inverse\_difference(recover2.Norm, recover2.Pred).to\_frame()  Transform\_reverssed\_J2.columns = ["Pred\_Normed"]  Final\_J2\_Pred = (Transform\_reverssed\_J2.values\* std\_J2) + avg\_J2  Transform\_reverssed\_J2["Pred\_Final"] =Final\_J2\_Pred  #Plotting the Predictions with originals  Sub\_Plots2(Transform\_reverssed\_J2["Pred\_Final"], dataframe\_2[-1426:-1],"Pridictions And Origna |

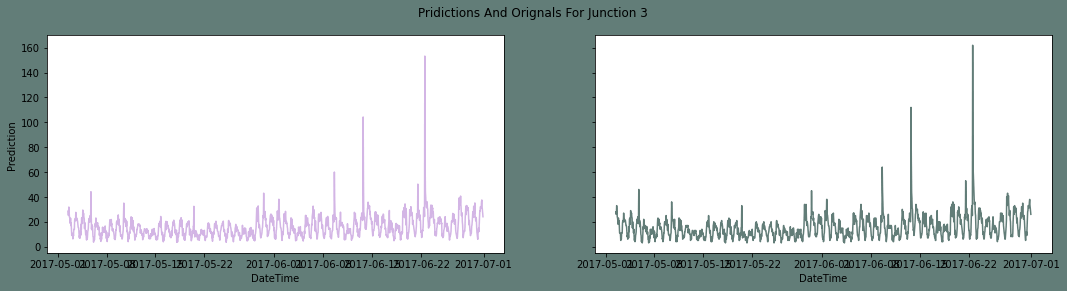
**Output:**



**The third junction's inverse transform**

|  |
| --- |
| #Invert the differenced J3  recover3 = dataframe\_N3.Norm[-1429:-1].to\_frame() #len as per the diff  recover3["Pred"]= PredJ3  Transform\_reverssed\_J3 = inverse\_difference(recover3.Norm, recover3.Pred).to\_frame()  Transform\_reverssed\_J3.columns = ["Pred\_Normed"]  #Invert the normalization J3  Final\_J3\_Pred = (Transform\_reverssed\_J3.values\* std\_J3) + avg\_J3  Transform\_reverssed\_J3["Pred\_Final"] =Final\_J3\_Pred  Sub\_Plots2(Transform\_reverssed\_J3["Pred\_Final"], dataframe\_3[-14 |

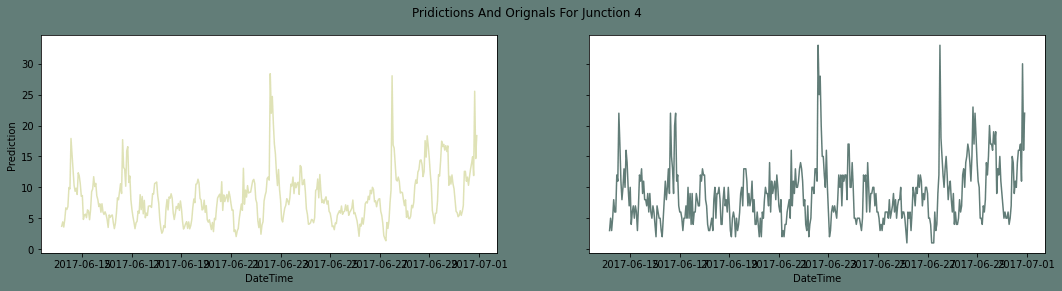
**Output:**



**The fourth junction's inverse transformation**

|  |
| --- |
| #Invert the differenced J4  recover4 = dataframe\_N4.Norm[-404:-1].to\_frame() #len as per the test set  recover4["Pred"]= PredJ4  Transform\_reverssed\_J4 = inverse\_difference(recover4.Norm, recover4.Pred).to\_frame()  Transform\_reverssed\_J4.columns = ["Pred\_Normed"]  #Invert the normalization J4  Final\_J4\_Pred = (Transform\_reverssed\_J4.values\* std\_J4) + avg\_J4  Transform\_reverssed\_J4["Pred\_Final"] =Final\_J4\_Pred  Sub\_Plots2(Transform\_reverssed\_J4["Pred\_Final"], dataframe |

**Output:**



## **Summary on the Dataset**

Here To anticipate the traffic at four crossroads, we trained a **GRU Neural network**. To create a stationary time series, we employed a normalization and differentiating transform. We used a different strategy for each intersection to make it stationary because the junctions vary in trends and seasonality. We used the root mean squared error as the model's assessment measure. Also, we placed the predictions next to the initial test results. Conclusions are drawn from the data analysis:

**Compared to junctions two and three, junction one is seeing a faster increase in the number of cars**. Junction Four has very little data. Therefore, we can't draw any conclusions from it.

The Junction one's traffic has a stronger weekly seasonality as well as hourly seasonality. In comparison, other junctions are significantly linear.

## **Conclusion**

In conclusion, traffic prediction using machine learning is an effective solution for addressing traffic congestion in urban areas. With the availability of vast amounts of traffic data, machine learning algorithms can accurately predict traffic flow and congestion patterns in real time. These predictions can be used to optimize traffic flow and improve the overall efficiency of transportation systems. While there are some challenges associated with traffic prediction using machine learning, the potential benefits are significant and can lead to improved transportation systems and reduced economic losses.

***Thank*** ***You !***