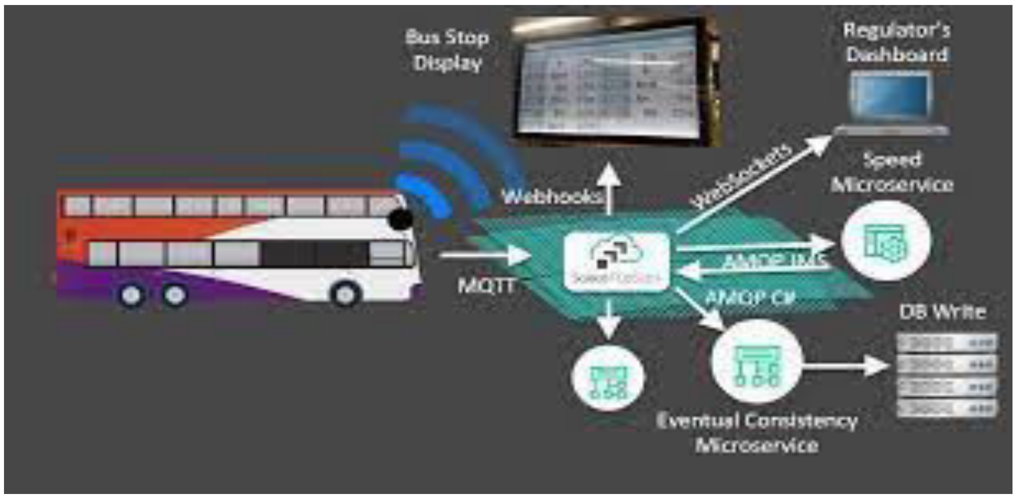
ARRIVAL TIME PREDICTION USING MACHINE LEARNING

Phase 2 Submission Document

Project : Public Transport Optimization



Introduction:

✔ Public transport optimization (PTO) systems are essential to human mobility. PT investments continue to grow, in order to improve PT services. Accurate PT arrival time prediction (PTO-ATP) is vital for PT systems delivering an attractive service, since the waiting experience for urban residents is an urgent problem to be solved. However, accurate PT-ATP is a challenging task due to the fact that urban traffic conditions are complex and changeable.

✔ Nowadays thousands of PT agencies publish their public transportation route and timetable information with the General Transit Feed Specification (GTFS) as the standard open format. Such data provide new opportunities for using the data-driven approaches to provide effective bus information system. This paper proposes a new framework to address the PT-ATP problem by using GTFS data. Also, an overview of various ML models for PT-ATP purposes is presented, along with the insightful findings through the comparison procedure based on real GTFS datasets. The results showed that the neural network -based method outperforms its rivals in terms of prediction accuracy.

✔ Briefly introduce the public transport optimization and importance of the arrival time prediction.
Highlight the limitaations of traditional regression models in capturing complex relationship.

✓ Emphasize the need for advanced regression techniques like RG(Random Forest) and KNN(K-nearest neighbour) to enhance the prediction accuracy.

Content for Project Phase 2 :

Consider exploring advance regression technique like Random forest and k-nearest neighbour for improving the prediction accuracy.

Data Source

A good data source for arrival time prediction using the machine learning should be accurate,Complete,Covering the public area of interest,Accessible.

Date	Time	Junction	Vehicles	ID
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
2015-11-01	05:00:00	1	6	20151101051
2015-11-01	06:00:00	1	9	20151101061
2015-11-01	07:00:00	1	8	20151101071
2015-11-01	08:00:00	1	11	20151101081
2015-11-01	09:00:00	1	12	20151101091
2015-11-01	10:00:00	1	15	20151101101
2015-11-01	11:00:00	1	17	20151101111
2015-11-01	12:00:00	1	16	20151101121
2015-11-01	13:00:00	1	15	20151101131
2015-11-01	14:00:00	1	16	20151101141
2015-11-01	15:00:00	1	12	20151101151
2015-11-01	16:00:00	1	12	20151101161
2015-11-01	17:00:00	1	16	20151101171
2015-11-01	18:00:00	1	17	20151101181
2015-11-01	19:00:00	1	20	20151101191
2015-11-01	20:00:00	1	17	20151101201
2015-11-01	21:00:00	1	19	20151101211
2015-11-01	22:00:00	1	20	20151101221
2015-11-01	23:00:00	1	15	20151101231
2015-11-02	00:00:00	1	14	20151102001
2015-11-02	01:00:00	1	12	20151102011
2015-11-02	02:00:00	1	14	20151102021
2015-11-02	03:00:00	1	12	20151102031
2015-11-02	04:00:00	1	12	20151102041
2015-11-02	05:00:00	1	11	20151102051
2015-11-02	06:00:00	1	13	20151102061
2015-11-02	07:00:00	1	14	20151102071
2015-11-02	08:00:00	1	12	20151102081
2015-11-02	09:00:00	1	22	20151102091
2015-11-02	10:00:00	1	32	20151102101
2015-11-02	11:00:00	1	31	20151102111
2015-11-02	12:00:00	1	35	20151102121
2015-11-02	13:00:00	1	26	20151102131
2015-11-02	14:00:00	1	34	20151102141
2015-11-02	15:00:00	1	30	20151102151

Data Collection and Preprocessing:

✓ Importing the dataset: Obtain a comprehensive dataset containing the relevant features such as date, time, junction, id, vehicle etc...

✓ Data preprocessing: Clean the data by handling the missing values, outliers, and categorical variables. Standardize or normalize numerical features.

Exploratory Data Analysis:

✓ Visualize and analyze the dataset to gain insight into the relationship between variables.

✓ Identify correlation and patterns that can inform features selection and engineering.

✓ Present various data visualizations to gain insights into the dataset.

✓ Explore correlations between features and the target variable (prediction time).

✓ Discuss any significant findings from the EDA phase that informs the feature selection.

Features Engineerig:

✓ Create new features or transform existing ones to capture valuable information.

✓ Utilize domain knowledge to engineer feature that may impact traffic condition such as arrival time, vehicle count and time complexity.

✓ Explain the process of creating the new features or transforming existing ones.

✓ Showcase domain-specific feature engineering, such as historical data, statistical-based method, machine learning and hybrid methods.

✓ Emphasize the impact of engineered features on model performance.

Advanced Regession Technique:

Time Series Analysis: It is a specific way of analyzing a sequence of data points collected over an interval of time.

Neutral Network: It predict the traffic volume in two levels such as short-term and mid-term.

Random Forest: It creates multiple decision tree and merges their data to obtain accurate prediction.

K-Nearest Neighbor: This algorithm is used to spatially screen the station data to determine the points with high correlation and then input the BILSTM model for prediction.

Model Evaluvation and Selection:

✔ Split the dataset into training and testing sets.

✔ Evaluate models using appropriate metrices (e.g., Mean Absolute Error, Mean Squared Error, R-Squared) to access their performance.

✔ Use cross-validation techniques to tune hyperparameters and ensure model stability .

✔ compare the results with the traditional regression model to highlight improvement.

✔ Select the best-performing model for further analysis.

Model Interpretability:

✔ Explain how to interpret features importances from Random Forest and K-Nearest Neighbor models.

✔ Discuss the insight gained from feature importance analysis and their relevance to arrival time prediction.

✔ Interpret feature importance from ensemble models like Random Forest and time series analysis to understand the factors influencing arrival time.

Deployment and Prediction:

✔ Deploy the chosen regression model to predict arrival time

✔ Develop the user friendly interface for user to input properly features and receive arrival time prediction.

Program:

Arrival Time Prediction

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
```

In [1]:

```
#Loading Data
data = pd.read_csv("../input/traffic-prediction-dataset/traffic.csv")
data.head()
```

out[1]:

DateTime	Junction	Vehicles	ID	
0	**2015-11-01 00:00:00**	**1**	**15**	**20151101001**
1	**2015-11-01 01:00:00**	**1**	**13**	**20151101011**
2	**2015-11-01 02:00:00**	**1**	**10**	**20151101021**
3	**2015-11-01 03:00:00**	**1**	**7**	**20151101031**
4	**2015-11-01 04:00:00**	**1**	**9**	**20151101041**

In [2]:

```
data["DateTime"]= pd.to_datetime(data["DateTime"])

data = data.drop(["ID"], axis=1) #dropping IDs

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>

  RangeIndex: 48120 entries, 0 to 48119
```

Data columns (total 3 columns):

out[2]:

#	Column	Non-Null Count	Dtype
---	------	--------------	-----
0	DateTime	48120 non-null	datetime64[ns]
1	Junction	48120 non-null	int64
2	Vehicles	48120 non-null	int64

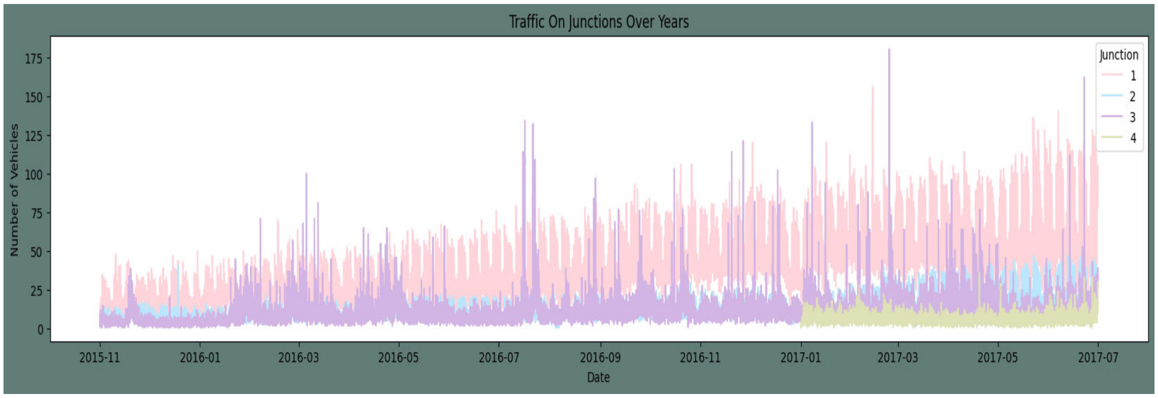
dtypes: datetime64[ns](1), int64(2

In [3]:

```
#Let's plot the Timeseries
colors = [ "#FFD4DB","#BBE7FE","#D3B5E5","#dfe2b6"]
plt.figure(figsize=(20,4),facecolor="#627D78")
Time_series=sns.lineplot(x=data['DateTime'],y="Vehicles",data=data,
  hue="Junction", palette=colors)
Time_series.set_title("Traffic On Junctions Over Years")
Time_series.set_ylabel("Number of Vehicles")
Time_series.set_xlabel("Date")
```

out[3]:

Text(0.5, 0, 'Date')



In [4]:

```
# Extract year, month, day, hour, and day of the week as features
data['Year'] = data['DateTime'].dt.year
data['Month'] = data['DateTime'].dt.month
data['Date_no'] = data['DateTime'].dt.day
data['Hour'] = data['DateTime'].dt.hour
data['Day'] = data['DateTime'].dt.strftime("%A")
data.head()
```

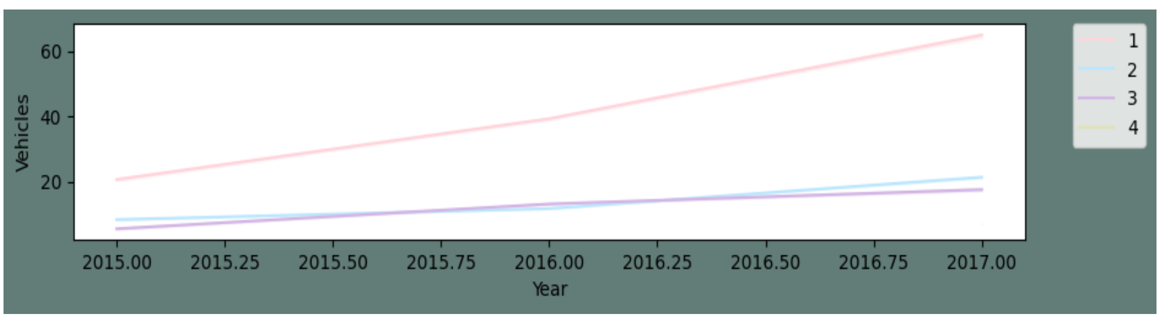
out[4]:

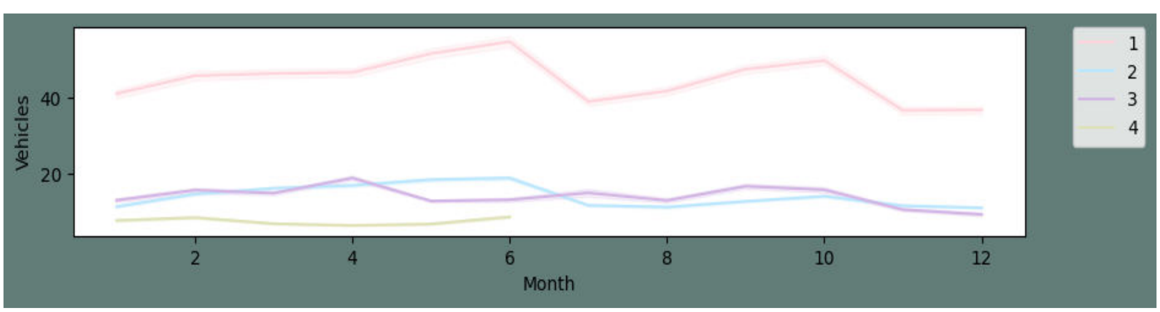
	DateTime	Junction	Vehicles	Year	Month	Date_no	Hour	Day
0	2015-11-01 00:00:00	1	15	2015	11	1	0	Sunday
1	2015-11-01 01:00:00	1	13	2015	11	1	1	Sunday
2	2015-11-01 02:00:00	1	10	2015	11	1	2	Sunday
3	2015-11-01 03:00:00	1	7	2015	11	1	3	Sunday
4	2015-11-01 04:00:00	1	9	2015	11	1	4	Sunday

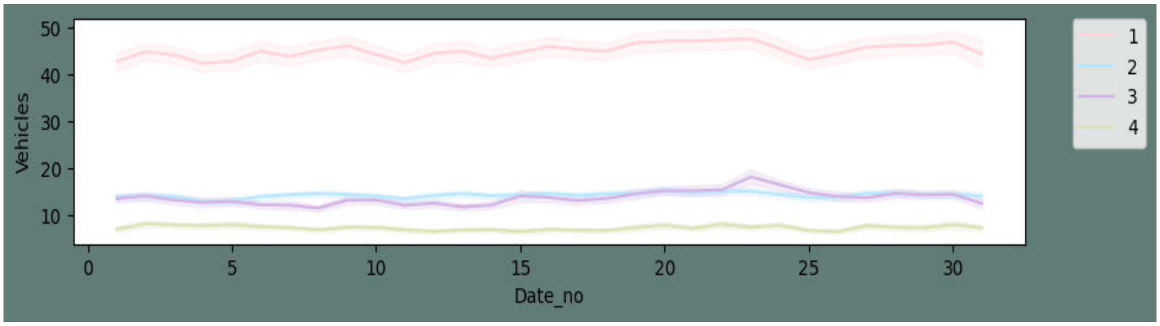
In [5]:

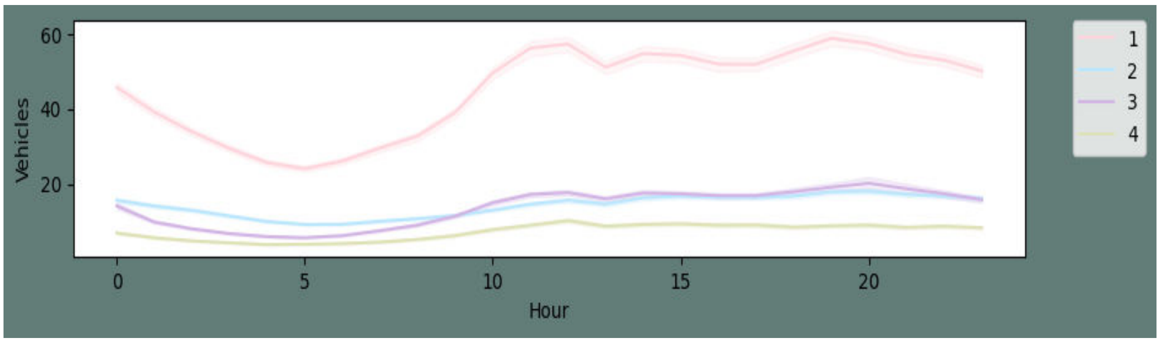
```
#Let's plot the Timeseries

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

for i in new_features:

plt.figure(figsize=(10,2),facecolor="#627D78")

ax=sns.lineplot(x=data[i],y="Vehicles",data=data, hue="Junction", palette=colors )

plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```

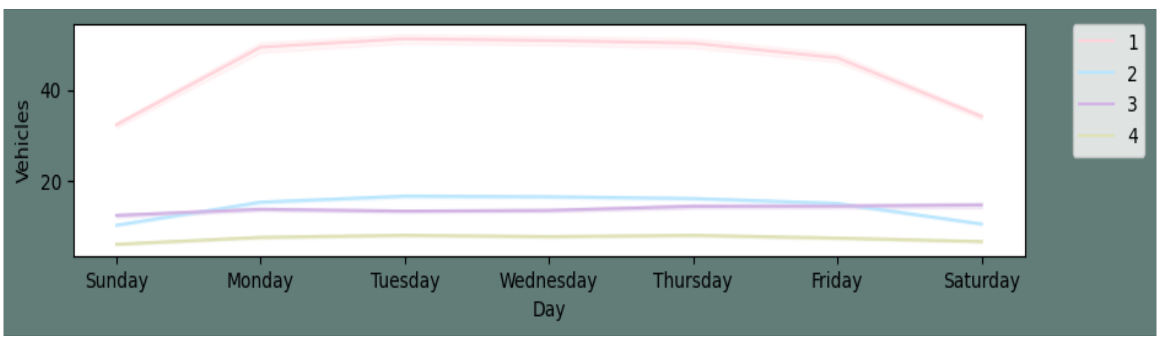
out[5]:







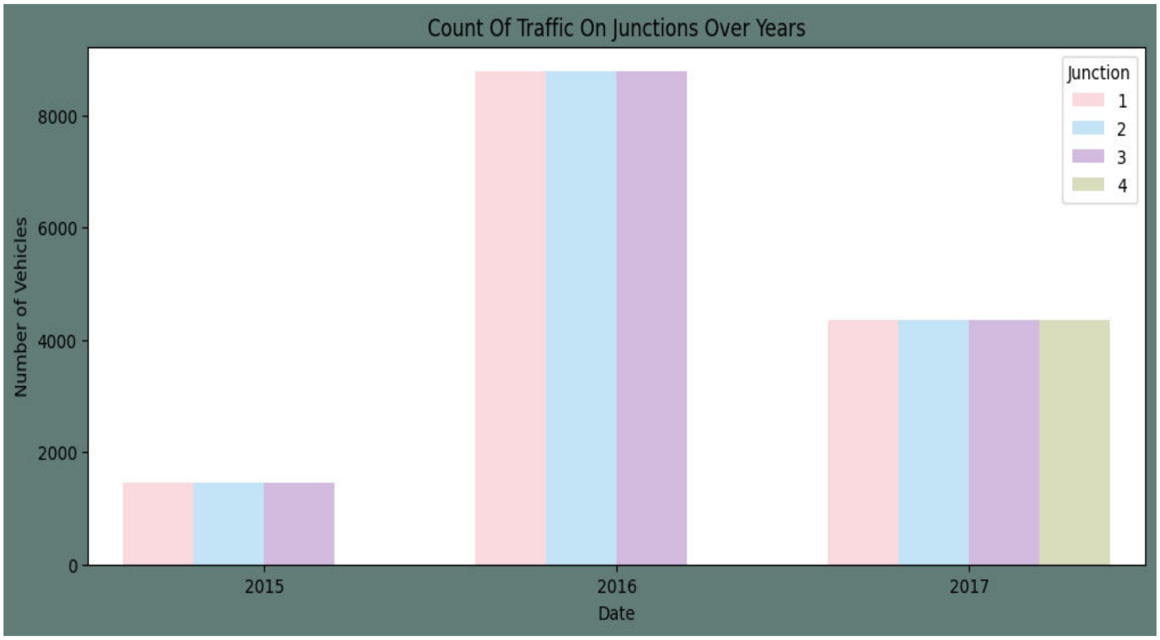




In [6]:

```
plt.figure(figsize=(12,5),facecolor="#627D78")
count = sns.countplot(data=data, x =data["Year"], hue="Junction", palette=colors)
count.set_title("Count Of Traffic On Junctions Over Years")
count.set_ylabel("Number of Vehicles")
count.set_xlabel("Date")
```

Out[6]:

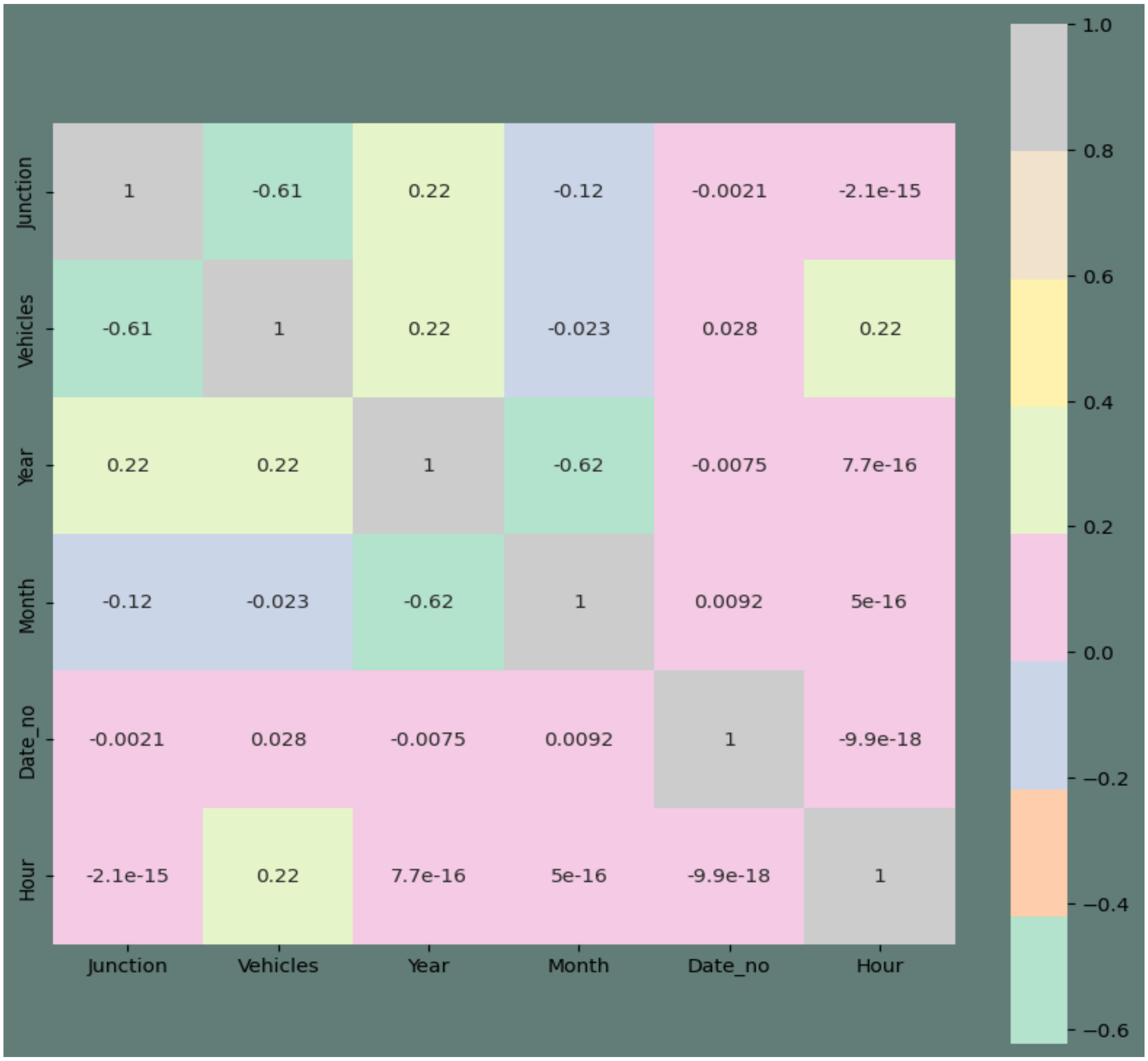


In [7]:

```
numeric_df = data.select_dtypes(include=[np.number])     # Select only numeric
columns
corrmat = numeric_df.corr()
plt.subplots(figsize=(10,10),facecolor="#627D78")
sns.heatmap(corrmat,cmap= "Pastel2",annot=True,square=True, )
```

Out[7]:

<Axes: >



In [8]:

```
# One-hot encode the 'Day' column
data = pd.get_dummies(data, columns=['Day'], prefix=['Day'])

# Implement linear regression from scratch
def train_linear_regression(X, y):
# Add a column of ones to X for the intercept term
X = np.column_stack((np.ones(X.shape[0]), X))
# Calculate the coefficients using the normal equation
coefficients = np.linalg.inv(X.T @ X) @ X.T @ y
return coefficients
```

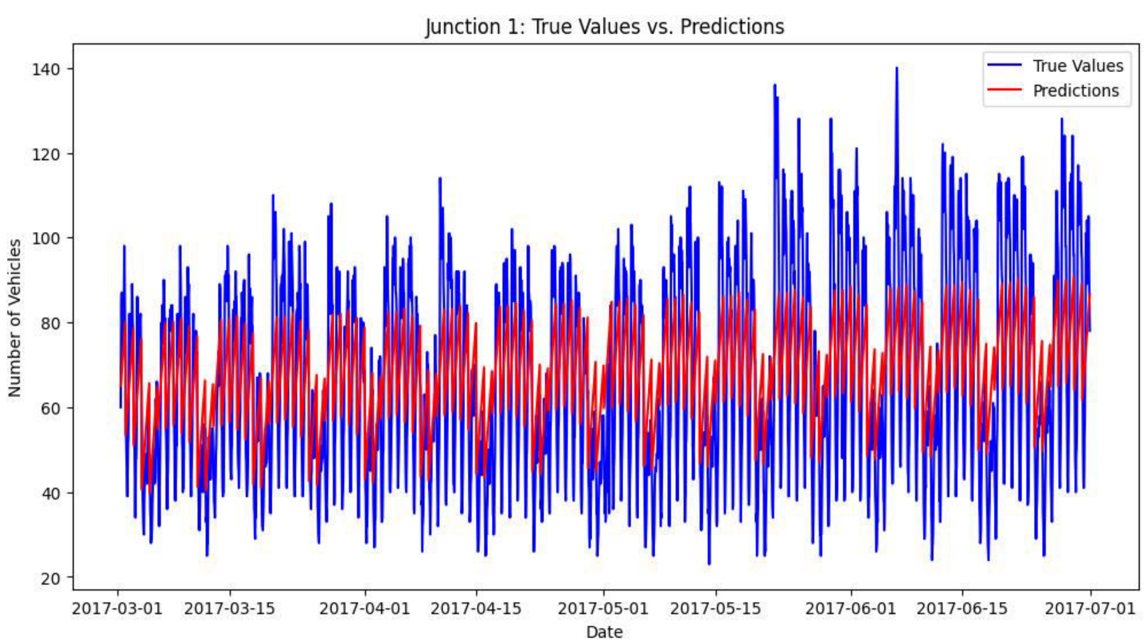
```
def predict_linear_regression(coefficients, X):
# Add a column of ones to X for the intercept term
X = np.column_stack((np.ones(X.shape[0]), X))
# Make predictions
y_pred = X @ coefficients
return y_pred
```

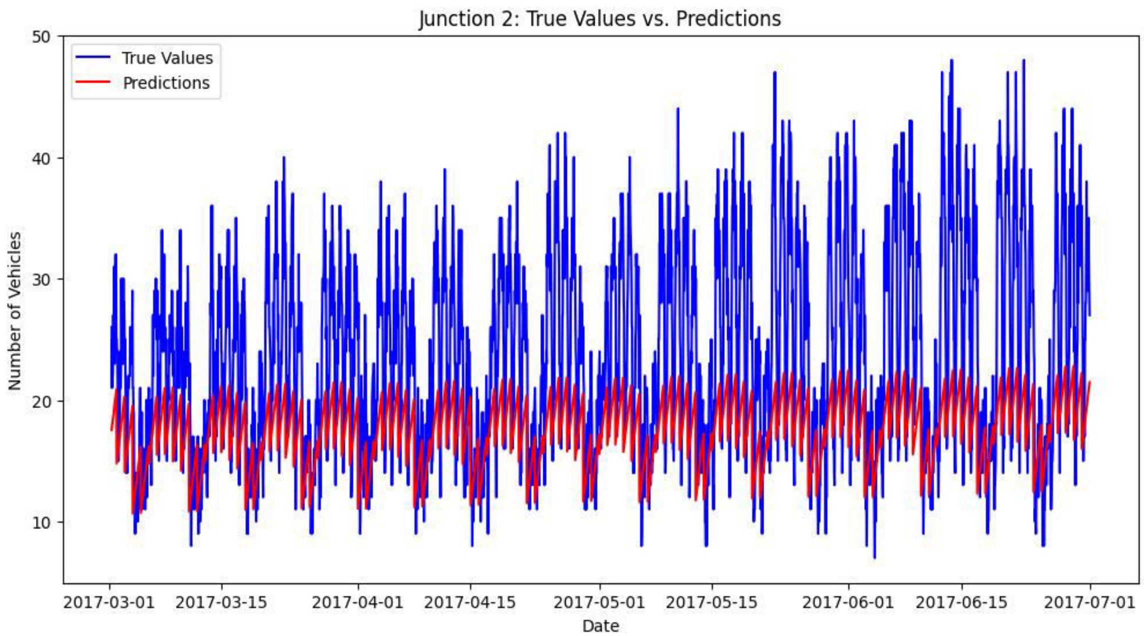
In [9]:

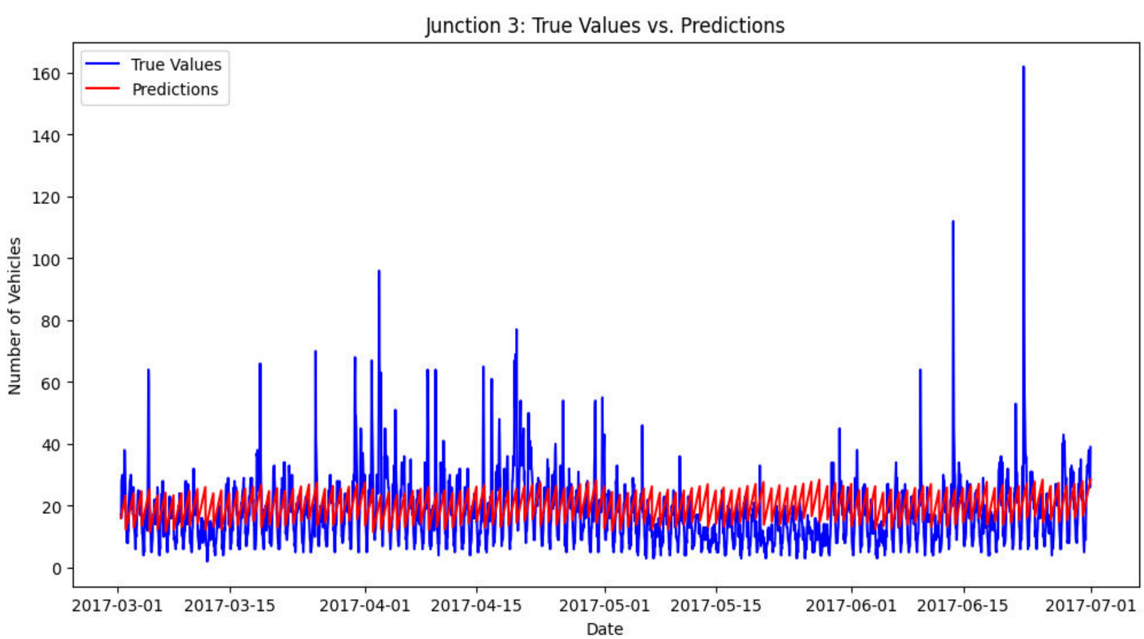
```
# Select a junction (replace 1 with your desired junction number)
for junction_number in range(1,4):
df = data[data['Junction'] == junction_number].copy()

# Split your data into training and test sets (adjust the split ratio as needed)
train_size = int(0.8 * len(df))
train_data, test_data = df[:train_size], df[train_size:]


# Prepare features and target variables
X_train = train_data[['Year', 'Month', 'Date_no', 'Hour', 'Day_Monday',
'Day_Tuesday', 'Day_Wednesday', 'Day_Thursday', 'Day_Friday', 'Day_Saturday',
'Day_Sunday']]
y_train = train_data['Vehicles']
X_test = test_data[['Year', 'Month', 'Date_no', 'Hour', 'Day_Monday',
'Day_Tuesday', 'Day_Wednesday', 'Day_Thursday', 'Day_Friday', 'Day_Saturday',
'Day_Sunday']]
y_test = test_data['Vehicles']

# Scale your input features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train the linear regression model
coefficients = train_linear_regression(X_train_scaled, y_train
```

```python
# Make predictions on the test set
y_pred = predict_linear_regression(coefficients, X_test_scaled)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print(f'MSE: {mse:.4f}')


# Plot predictions vs. true values
plt.figure(figsize=(12, 6))
plt.plot(test_data['DateTime'], y_test, label='True Values', color='blue')
plt.plot(test_data['DateTime'], y_pred, label='Predictions', color='red')
plt.title(f'Junction {junction_number}: True Values vs. Predictions')
plt.xlabel('Date')
plt.ylabel('Number of Vehicles')
plt.legend()
plt.show()
```

Out[9]:




Junction 2: True Values vs. Predictions
True Values
Predictions
Number of Vehicles
50
40
30
20
10
2017-03-01
2017-03-15
2017-04-01
2017-04-15
2017-05-01
2017-05-15
2017-06-01
2017-06-15
2017-07-01
Date


Junction 3: True Values vs. Predictions
True Values
Predictions
Number of Vehicles
160
140
120
100
80
60
40
20
0
2017-03-01
2017-03-15
2017-04-01
2017-04-15
2017-05-01
2017-05-15
2017-06-01
2017-06-15
2017-07-01
Date