```
import numpy as np
import pandas as pd
```

```
import matplotlib.pyplot as plt
%matplotlib inline
from datetime import datetime
import time
import seaborn as sns

from sklearn.preprocessing import *
from sklearn.model_selection import *
from sklearn.metrics import *
import lightgbm as lgb
import xgboost as xgb
```

```
dataset_train=pd.read_csv('/content/train_aWnotuB.csv')
dataset_test=pd.read_csv('/content/test_BdBKkAj.csv')
```

```
dataset_train.head()
```

```
dataset_test.head()
```

```
dataset_train.tail()
```

```
dataset_train.info()
```

```
dataset_train.duplicated
```

```
<bound method DataFrame.duplicated of                    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
...                    ...       ...       ...          ...
48115  2017-06-30 19:00:00         4        11  20170630194
48116  2017-06-30 20:00:00         4        30  20170630204
48117  2017-06-30 21:00:00         4        16  20170630214
48118  2017-06-30 22:00:00         4        22  20170630224
48119  2017-06-30 23:00:00         4        12  20170630234

[48120 rows x 4 columns]>
```

```
dataset_train.duplicated()
```

```
new_dataset=dataset_train.drop_duplicates()
new_dataset
```

```
new_dataset.shape
```

```
new_dataset.describe()
```

```
new_dataset['DateTime'] = pd.to_datetime(new_dataset['DateTime'])

new_dataset['Date'] = new_dataset['DateTime'].dt.date
new_dataset['Time'] = new_dataset['DateTime'].dt.time

new_dataset['Date'] = new_dataset['Date'].astype(str)
new_dataset['Time'] = new_dataset['Time'].astype(str)
```

```
new_dataset.head()
```

	DateTime	Junction	Vehicles	ID	Date	Time
0	2015-11-01 00:00:00	1	15	20151101001	2015-11-01	00:00:00
1	2015-11-01 01:00:00	1	13	20151101011	2015-11-01	01:00:00

```
new_dataset = new_dataset.drop('DateTime', axis = 1)
```

```
new_dataset['Date'] = pd.to_datetime(new_dataset['Date'])

new_dataset['DayOfWeek'] = new_dataset['Date'].dt.dayofweek

new_dataset['Month'] = new_dataset['Date'].dt.month

new_dataset['Year'] = new_dataset['Date'].dt.year
```

```
new_dataset.head()
```

	Junction	Vehicles	ID	Date	Time	DayOfWeek	Month	Year
0	1	15	20151101001	2015-11-01	00:00:00	6	11	2015
1	1	13	20151101011	2015-11-01	01:00:00	6	11	2015
2	1	10	20151101021	2015-11-01	02:00:00	6	11	2015
3	1	7	20151101031	2015-11-01	03:00:00	6	11	2015
4	1	9	20151101041	2015-11-01	04:00:00	6	11	2015

```
import holidays
new_dataset['Date'] = pd.to_datetime(new_dataset['Date'])

indian_holidays = holidays.India(years=new_dataset['Date'].dt.year.unique())


def is_holiday_in_india(date):
    return date in indian_holidays


new_dataset['IsHoliday'] = new_dataset['Date'].apply(lambda x: is_holiday_in_india(x))
```

```
new_dataset['IsHoliday'].unique()
```

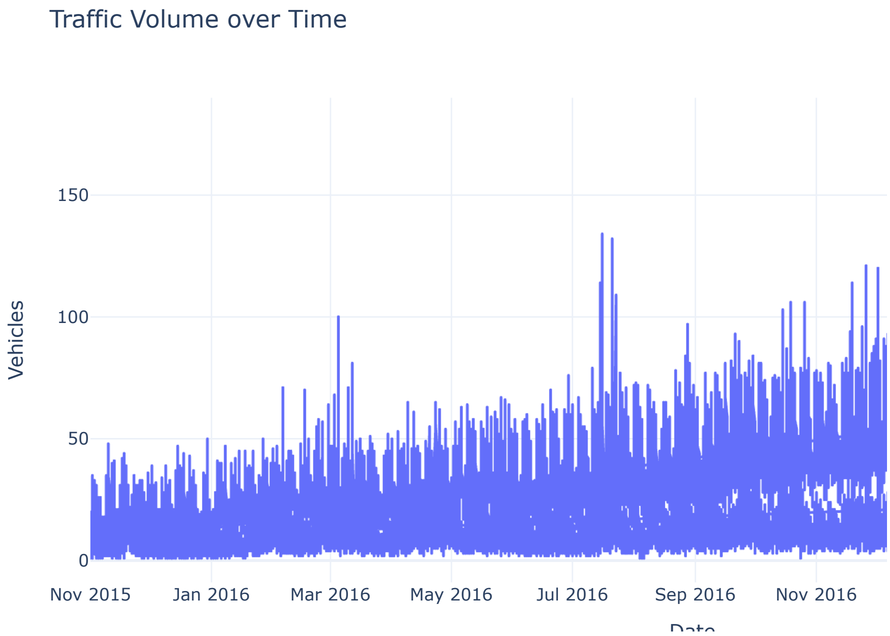
```
array([False,  True])
```

```
new_dataset = new_dataset.drop(['ID'], axis = 1)
new_dataset.head()
```

	Junction	Vehicles	Date	Time	DayOfWeek	Month	Year	IsHoliday
0	1	15	2015-11-01	00:00:00	6	11	2015	False
1	1	13	2015-11-01	01:00:00	6	11	2015	False
2	1	10	2015-11-01	02:00:00	6	11	2015	False
3	1	7	2015-11-01	03:00:00	6	11	2015	False
4	1	9	2015-11-01	04:00:00	6	11	2015	False

```
import plotly.express as px
import plotly.graph_objects as go
import plotly.io as pio
import matplotlib.pyplot as plt
import seaborn as sns
pio.templates.default="plotly_white"
```

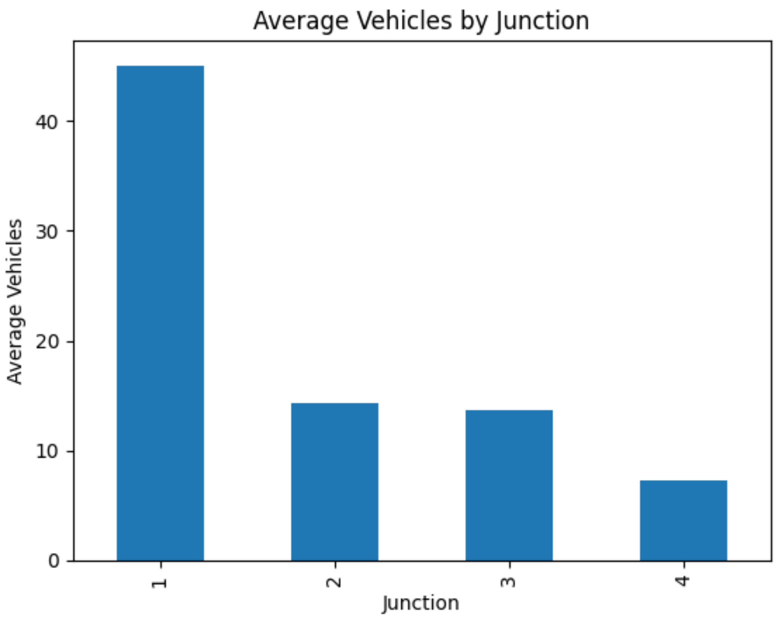
```
fig = px.line(new_dataset, x='Date', y='Vehicles', title='Traffic Volume over Time')
fig.show()
```



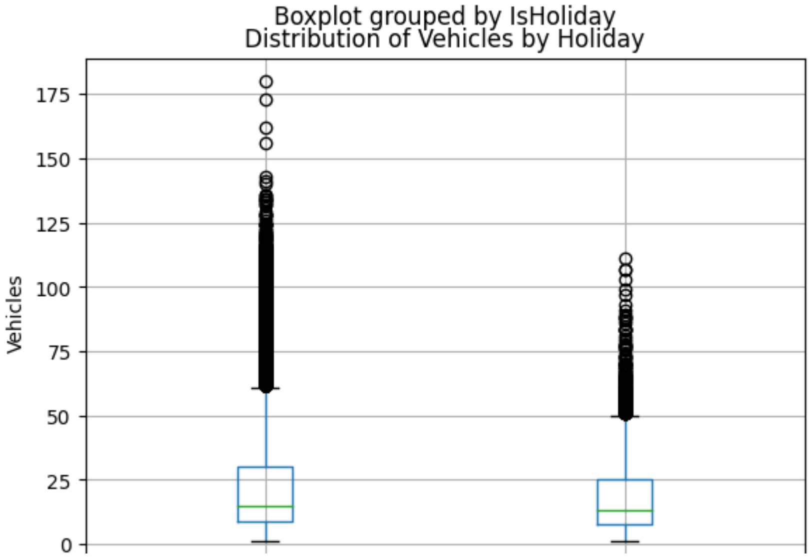

```
station_stats= new_dataset.groupby('Junction')['Vehicles'].describe()
print(station_stats)
```

```
            count       mean        std  min   25%   50%   75%    max
Junction
1         14592.0  45.052906  23.008345  5.0  27.0  40.0  59.0  156.0
2         14592.0  14.253221   7.401307  1.0   9.0  13.0  17.0   48.0
3         14592.0  13.694010  10.436005  1.0   7.0  11.0  18.0  180.0
4          4344.0   7.251611   3.521455  1.0   5.0   7.0   9.0   36.0
```

```
import matplotlib.pyplot as plt
new_dataset.groupby('Junction')['Vehicles'].mean().plot(kind='bar')
plt.xlabel('Junction')
plt.ylabel('Average Vehicles')
plt.title('Average Vehicles by Junction')
plt.show()
```




```
new_dataset.boxplot(column='Vehicles', by='IsHoliday')
plt.xlabel('Is Holiday')
plt.ylabel('Vehicles')
plt.title('Distribution of Vehicles by Holiday')
plt.show()
```



Hypothesis Testing- T-test

```
from scipy.stats import ttest_ind

jun_1 = new_dataset[new_dataset['Junction'] == 1]['Vehicles']
jun_2 = new_dataset[new_dataset['Junction'] == 2]['Vehicles']

t_stat, p_value = ttest_ind(junction_1, junction_2)
print(f"t-statistic: {t_stat}, p-value: {p_value}")
```

```
t-statistic: 153.93470815373487, p-value: 0.0
```

The t-statistic of 153.93 underscores a significant disparity in the average number of vehicles between the two junctions. This is further supported by the extremely low p-value of 0.0, signifying that the probability of encountering such data assuming no distinction between the means of the two groups is nearly impossible. Therefore, the null hypothesis can be confidently dismissed.

Feature Engineering

```
new_dataset['PreviousDayVehicles']= new_dataset['Vehicles'].shift(24)
new_dataset['PreviousHourVehicles']=new_dataset['Vehicles'].shift(1)
```

```
new_dataset.head()
```

	Junction	Vehicles	Date	Time	DayOfWeek	Month	Year	IsHoliday	Previo
0	1	15	2015-11-01	00:00:00	6	11	2015	False	
1	1	13	2015-11-01	01:00:00	6	11	2015	False	
2	1	10	2015-11-01	02:00:00	6	11	2015	False	
			2015-						

```
new_dataset.isnull().sum()
```

```
Junction                0
Vehicles                0
Date                    0
Time                    0
DayOfWeek               0
Month                   0
Year                    0
IsHoliday               0
PreviousDayVehicles    24
PreviousHourVehicles    1
dtype: int64
```

```
new_dataset['PreviousDayVehicles'].fillna(0,inplace=True)
new_dataset['PreviousDayVehicles'].fillna(0, inplace=True)
new_dataset.head()
```

	Junction	Vehicles	Date	Time	DayOfWeek	Month	Year	IsHoliday	Previo
0	1	15	2015-11-01	00:00:00	6	11	2015	False	
1	1	13	2015-11-01	01:00:00	6	11	2015	False	
2	1	10	2015-11-01	02:00:00	6	11	2015	False	
			2015-						

```
new_dataset.shape
```

```
(48120, 10)
```

```
dataset_test.shape
```

```
(11808, 3)
```

```
from keras.models import Sequential
from keras.layers import LSTM, Dense
from sklearn.preprocessing import MinMaxScaler
```

```
new_dataset['Vehicles'] = new_dataset['Vehicles'].astype(float)

scaler = MinMaxScaler()
new_dataset[['Vehicles']] = scaler.fit_transform(new_dataset[['Vehicles']])
```

```
new_dataset['Year'] = new_dataset['Date'].dt.year
new_dataset['Month'] = new_dataset['Date'].dt.month
new_dataset['Day'] = new_dataset['Date'].dt.day
new_dataset['Hour'] = new_dataset['Date'].dt.hour
```

```
new_dataset.drop(columns=['Date'], inplace=True)
```

```
print(new_dataset.dtypes)
```

```
Junction                  int64
Vehicles                float64
Time                     object
DayOfWeek                 int64
Month                     int64
Year                      int64
IsHoliday                  bool
PreviousDayVehicles     float64
PreviousHourVehicles    float64
Day                       int64
Hour                      int64
dtype: object
```

```
new_dataset['IsHoliday'] = new_dataset['IsHoliday'].astype(int)
```

```
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.metrics import mean_absolute_error, mean_squared_error
sequence_length = 24

X_columns = ['Junction', 'Vehicles', 'DayOfWeek', 'Month', 'Year', 'IsHoliday',
             'PreviousDayVehicles', 'PreviousHourVehicles']
y_column = ['Vehicles']


X = []
y = []

data_array = new_dataset.values

data_array = new_dataset[X_columns + y_column].values
```

```
for i in range(len(data_array) - sequence_length):
    X.append(data_array[i:i + sequence_length, :-1])
    y.append(data_array[i + sequence_length, -1])

X = np.array(X)
y = np.array(y)


# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)



model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(LSTM(units=50))
model.add(Dense(units=1))


model.compile(optimizer='adam', loss='mean_squared_error')


model.fit(X_train, y_train, epochs=30, batch_size=32, validation_split=0.2)
```

```
962/962 [==============================] - 25s 26ms/step - loss: 0.0016 - val_loss: 0.0013
Epoch 3/30
962/962 [==============================] - 25s 26ms/step - loss: 0.0015 - val_loss: 0.0031
Epoch 4/30
962/962 [==============================] - 26s 27ms/step - loss: 0.0014 - val_loss: 0.0011
Epoch 5/30
962/962 [==============================] - 25s 26ms/step - loss: 0.0013 - val_loss: 0.0014
Epoch 6/30
962/962 [==============================] - 25s 26ms/step - loss: 0.0012 - val_loss: 0.0011
Epoch 7/30
962/962 [==============================] - 26s 27ms/step - loss: 0.0012 - val_loss: 9.9593e-04
Epoch 8/30
962/962 [==============================] - 25s 26ms/step - loss: 0.0012 - val_loss: 0.0015
Epoch 9/30
962/962 [==============================] - 25s 26ms/step - loss: 0.0011 - val_loss: 0.0010
Epoch 10/30
962/962 [==============================] - 27s 29ms/step - loss: 0.0011 - val_loss: 8.6550e-04
Epoch 11/30
962/962 [==============================] - 25s 26ms/step - loss: 0.0011 - val_loss: 0.0011
Epoch 12/30
962/962 [==============================] - 25s 26ms/step - loss: 0.0010 - val_loss: 0.0012
Epoch 13/30
962/962 [==============================] - 25s 26ms/step - loss: 0.0011 - val_loss: 0.0013
Epoch 14/30
962/962 [==============================] - 25s 26ms/step - loss: 0.0010 - val_loss: 0.0011
Epoch 15/30
962/962 [==============================] - 25s 26ms/step - loss: 0.0010 - val_loss: 8.2458e-04
Epoch 16/30
962/962 [==============================] - 25s 26ms/step - loss: 9.4444e-04 - val_loss: 9.8143e-04
Epoch 17/30
962/962 [==============================] - 26s 27ms/step - loss: 9.1680e-04 - val_loss: 7.4712e-04
Epoch 18/30
962/962 [==============================] - 25s 26ms/step - loss: 9.3331e-04 - val_loss: 7.2376e-04
Epoch 19/30
962/962 [==============================] - 26s 27ms/step - loss: 9.1574e-04 - val_loss: 8.4632e-04
Epoch 20/30
962/962 [==============================] - 25s 26ms/step - loss: 8.5862e-04 - val_loss: 7.2133e-04
Epoch 21/30
962/962 [==============================] - 25s 26ms/step - loss: 8.3523e-04 - val_loss: 6.7221e-04
Epoch 22/30
962/962 [==============================] - 26s 27ms/step - loss: 8.5343e-04 - val_loss: 6.3129e-04
Epoch 23/30
962/962 [==============================] - 25s 26ms/step - loss: 7.4102e-04 - val_loss: 6.1327e-04
Epoch 24/30
962/962 [==============================] - 26s 27ms/step - loss: 7.5761e-04 - val_loss: 7.0838e-04
Epoch 25/30
962/962 [==============================] - 25s 26ms/step - loss: 7.5383e-04 - val_loss: 6.2130e-04
Epoch 26/30
962/962 [==============================] - 25s 26ms/step - loss: 7.1037e-04 - val_loss: 6.6853e-04
Epoch 27/30
962/962 [==============================] - 26s 27ms/step - loss: 7.0032e-04 - val_loss: 8.0198e-04
Epoch 28/30
962/962 [==============================] - 25s 26ms/step - loss: 8.1497e-04 - val_loss: 8.2609e-04
Epoch 29/30
962/962 [==============================] - 26s 27ms/step - loss: 7.4450e-04 - val_loss: 6.4588e-04
Epoch 30/30
962/962 [==============================] - 27s 29ms/step - loss: 7.4614e-04 - val_loss: 6.3321e-04
<keras.src.callbacks.History at 0x7fd5abe11a20>
```

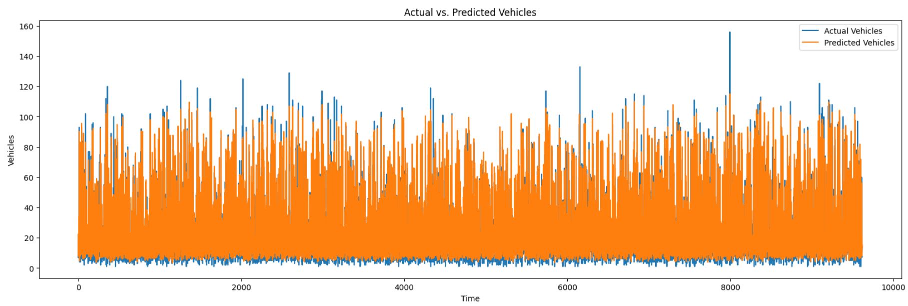
```
loss = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss}")


train_loss = model.evaluate(X_train, y_train)
print(f"Training Loss: {train_loss}")
```

```
301/301 [==============================] - 3s 7ms/step - loss: 6.1496e-04
Test Loss: 0.0006149551481939852
1203/1203 [==============================] - 9s 8ms/step - loss: 6.5297e-04
Training Loss: 0.0006529669044539332
```

```
y_pred = model.predict(X_test)


y_pred = y_pred.reshape(-1, 1)
y_test = y_test.reshape(-1, 1)



y_pred_original = scaler.inverse_transform(y_pred)


y_test_original = scaler.inverse_transform(y_test)
```

```
301/301 [==============================] - 3s 7ms/step
```

```
plt.figure(figsize=(20, 6))
plt.plot(y_test_original, label='Actual Vehicles')
plt.plot(y_pred_original, label='Predicted Vehicles')
plt.xlabel('Time')
plt.ylabel('Vehicles')
plt.title('Actual vs. Predicted Vehicles')
plt.legend()
plt.show()
```




```
new_dataset = new_dataset.fillna(0)


mae = mean_absolute_error(y_test_original, y_pred_original)
mse = mean_squared_error(y_test_original, y_pred_original)
rmse = np.sqrt(mse)

print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
```

```
Mean Absolute Error (MAE): 3.1034452600935145
Mean Squared Error (MSE): 19.70377080119958
Root Mean Squared Error (RMSE): 4.438892970234761
```

```
r2=r2_score(y_test_original, y_pred_original)
print(r2)
```

```
0.9528434389852125
```

```
new_dataset.head()
```

	Junction	Vehicles	Time	DayOfWeek	Month	Year	IsHoliday	PreviousDayV
0	1	0.078212	00:00:00	6	11	2015	0	
1	1	0.067039	01:00:00	6	11	2015	0	
2	1	0.050279	02:00:00	6	11	2015	0	
3	1	0.033520	03:00:00	6	11	2015	0	
4	1	0.044693	04:00:00	6	11	2015	0	

```
test_timestamps = new_dataset.iloc[-len(y_test_original):]['Time']


plt.figure(figsize=(25, 12))
plt.plot(test_timestamps, y_test_original, label='Actual Traffic Counts', color='blue')


forecasted_timestamps =  new_dataset.iloc[-len(y_test_original):]['Time']


plt.plot(forecasted_timestamps, y_pred_original, label='Forecasted Traffic Counts', color='red')

plt.xlabel('Time')
plt.ylabel('Traffic Counts')
plt.title('Forecasted / Predicted vs. Actual Traffic Counts')
plt.legend()
plt.grid(True)
plt.show()
```