Introduction-

This is a project submission regarding traffic forecaster which is build using LSTM neural network, which is most basic type of RNN.

This is a time series problem as the traffic to be predicted depends on the historic data of traffic on different days, and that historic data of traffic is time dependent.

First a sequential model is built, which is predicted after the model get compiled.

The predicted values are plotted against the actual data and with keeping its consideration , it can be said that the LSTM models fits well as it overlaps properly with the given data.

Objective-

‘Building a traffic forecaster that can predict the traffic flow in a smart city accurately and in real-time.’

Dataset is given with features namely hr, month, day, total_volume. Our aim is to predict the volume of next hour ,which depend upon the volume of some previous volumes. Here, by volume I mean the number of vehicle contributing to the traffic.

So, it is quite obvious that it is a time series problem. In which the output to be predicted depends only upon the time dependent single input (i.e. total_volume in this problem).

Two famous models to solve time series problems are ARMA(Autoregressive moving average) and ARIMA(Autoregressive integrated moving average).But they are mainly linear models and cannot describe the stochastic and nonlinear nature of traffic flow.

In recent years, deep learning based methods have been devised as alternatives for traffic flow prediction. We have used Long Short-Term Memory (LSTM) method to predict short-term traffic flow.

Let's begin the coding part:

```
data = pd.read_csv('traffic_data_1hr.csv') print(data.shape)
print(data.head())
```

read_csv(), shape, head() methods of pandas are used to read the csv dataset file, get the shape and first few(5) rows of dataset respectively.

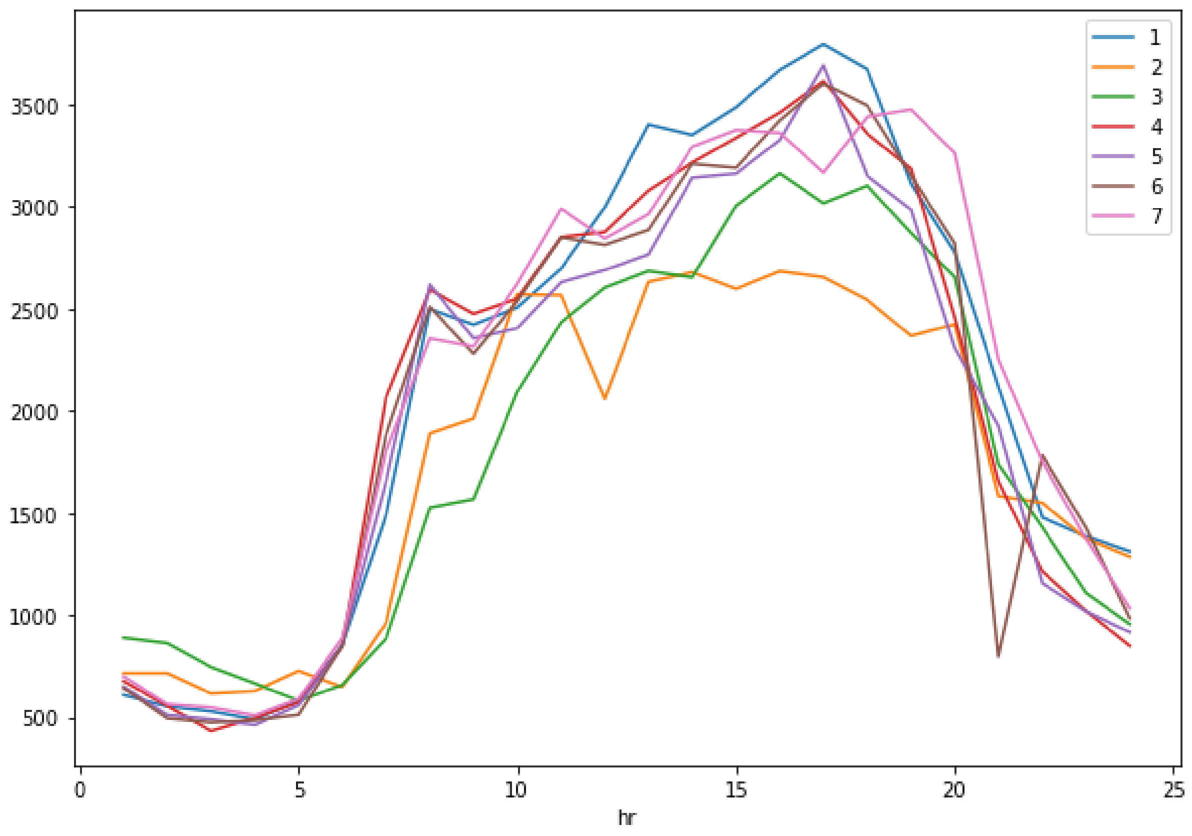
```
df = data['total_volume']
dataset = df.values.astype('float32')
dataset = np.reshape(dataset, (len(dataset),1))
```

First, total_volume column is extracted from data as it is the information upon which the prediction will depend. It is better to convert df into float as later some scaling transformation will be applied. Using reshape() method dataset is shaped to 3624×1.

Let's see how the traffic_volume varies along with hr-

```
plot_data = data[(data.month == 5) & (data.day <= 7)]
fig, ax = plt.subplots(figsize=(10,7))
plot_data.set_index('hr', inplace=True)
pt = plot_data.groupby('weekday')
['total_volume'].plot(legend=True)
```

I am plotting the graph for first 7 days of month 5(i.e. may) against hr. To plot total_volume with respect to hr, first make hr as index, then plot. Also group the plot_data by days to get different lines for each seven days.



Let's star building the traffic forecasting model.

First import all the required Python libraries.

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM, SimpleRNN
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from keras.optimizers import SGD
import warnings warnings.filterwarnings("ignore")
```

LSTM model is very sensitive to the scale of the input data, it is a standard practice to standardize the data to the range of 0 to 1.

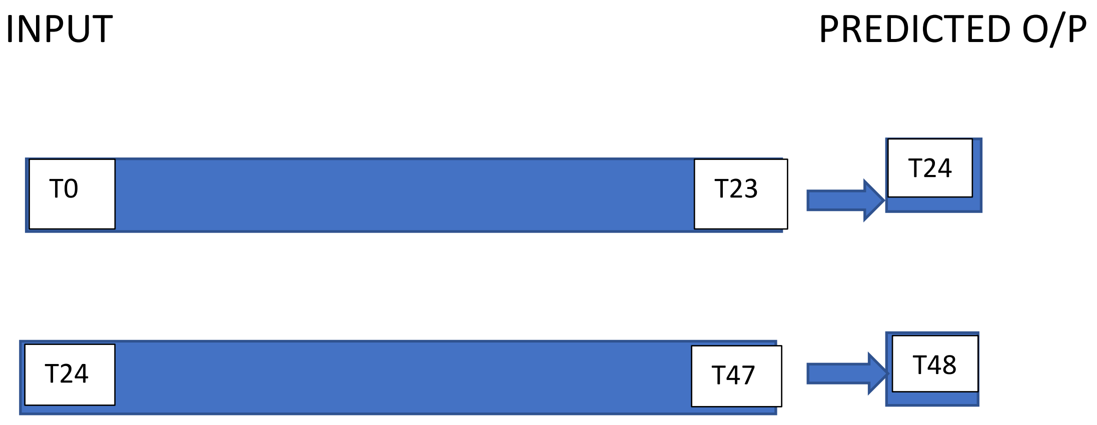
sklearn.preprocessing module has MinMaxScaler class that helps us standardize the data easily.

```
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
```

In classification or regression problem, we usually do this by splitting the dataset randomly into training data and test data. But in time series analysis, the sequence of data is also important. So, initial 70% of the dataset is used for training and rest 30% for testing purpose.

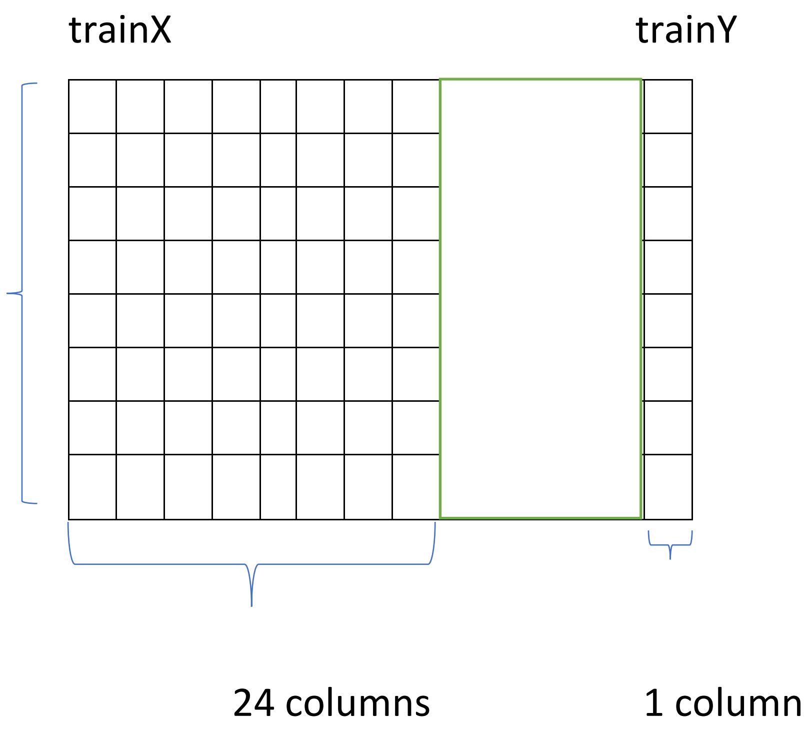
```
train_size = int(len(dataset) * 0.70)

test_size = len(dataset) - train_size

train, test = dataset[0:train_size , :], dataset[train_size:len(dataset)
, :]
```

Before start building the model, first have to divide the training and testing data into independent(X) and dependent data(Y). In this problem both independent and dependent data is total_volume which is stored in dataset variable. traffic_volume of present hr is the output, predicted by the traffic_volume of some previous hours as inputs. The count of observations considered as input to predict the output is called timesteps. Here, 24 timesteps(lookback) is selected. Means, there is need of 24 traffic_volume to predict the traffic_volume of 25th hour. The value of timestep is selected by trying different values for it and then best suited value is choosed.



```
def transform_dataset(dataset, look_back):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return np.array(dataX), np.array(dataY)
```

```
look_back = 24
trainX, trainY = transform_dataset(train, look_back)
testX, testY = transform_dataset(test, look_back)
```



Now the RNN takes 3D input as (batch_size, timesteps, input_dim). So, we have to convert the trainX and testX into required dimensions.

trainX = np.reshape(trainX, (trainX.shape[0], trainX.shape[1], 1))

testX = np.reshape(testX, (testX.shape[0], testX.shape[1], 1))

We are going to stochastic gradient descent optimizer. For that use the SGD and fix the learning rate as 0.1.

sgd=SGD(lr=0.1)

Define a sequential model with 2 layers as LSTM and Dense. Second layer is the last layer which is the output layer also.

model = Sequential()

model.add(LSTM(1, input_shape=(look_back,1)))

model.add(Dense(1))

After compiling the model with mean squared error, fit it with 10 epochs and 1 batch size.

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

model.fit(trainX, trainY, epochs=10, batch_size=1, verbose=2)

Next step is to predict the model with training and testing data i.e. trainX and testX.

trainPredict = model.predict(trainX)

testPredict = model.predict(testX)

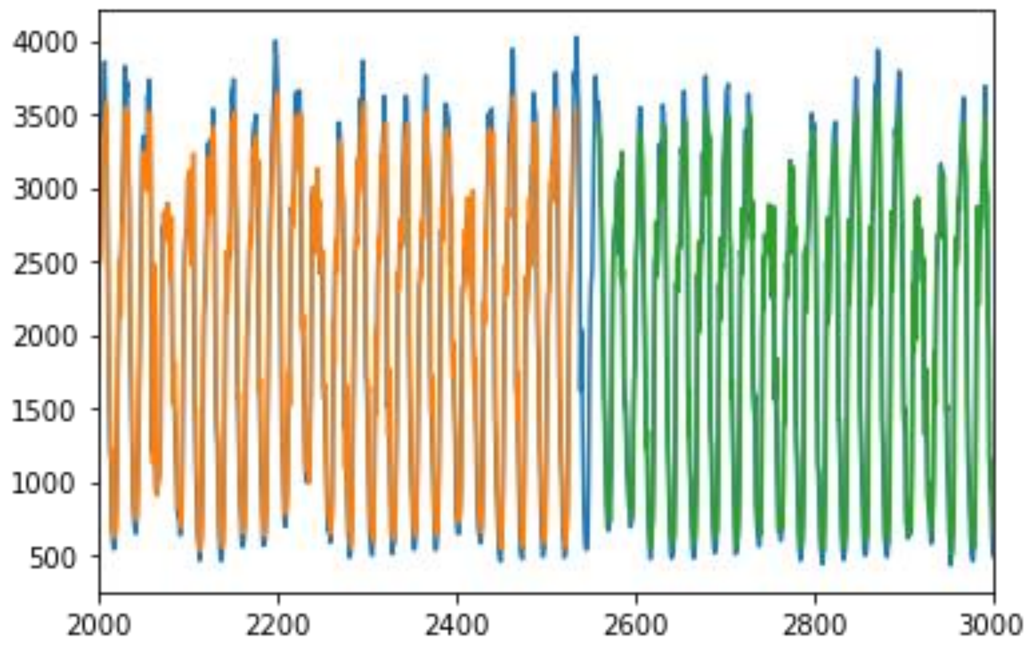
Initially the whole dataset was transformed to between 0 and 1, to get the initial values back use inverse transform function.

```
trainPredict = scaler.inverse_transform(trainPredict)

trainY = scaler.inverse_transform([trainY])

testPredict = scaler.inverse_transform(testPredict)

testY = scaler.inverse_transform([testY])
```

Now plot the actual values and the predicted values on a chart.

```
import matplotlib

from matplotlib import pyplot as plt


# shift train predictions for plotting

trainPredictPlot = np.empty_like(dataset)

trainPredictPlot[:, :] = np.nan

trainPredictPlot[look_back:len(trainPredict)+look_back, :] =
trainPredict

# shift test predictions for plotting

testPredictPlot = np.empty_like(dataset)

testPredictPlot[:, :] = np.nan

testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :]
= testPredict
```

```
# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.xlim(2000, 3000)
plt.show()
```


4000
3500
3000
2500
2000
1500
1000
500
2000
2200
2400
2600
2800
3000