Ex.No: 13 Learning – Use Supervised Learning

DATE: 11.11.24

REGISTER NUMBER : 212222040091

AIM:

To write a program to train the classifier for traffic.

Algorithm:

```
Step 1: Start the program.
Step 2: Import the necessary libraries, including NumPy and Pandas, for
data manipulation and visualization.
Step 3: Import additional libraries such as Seaborn, Matplotlib, and
Datetime for visualizing data and handling dates.
Step 4: Import machine learning libraries, including TensorFlow, Keras,
and statsmodels, along with utility functions for data scaling and
evaluation.
Step 5: Load the traffic dataset using Pandas read_csv function.
Step 6: Display the first few rows of the dataset to understand its
structure and contents.
Step 7: Convert the "DateTime" column to a datetime format to allow for
accurate time-series analysis.
Step 8: Drop any unnecessary columns, such as the "ID" column, to
simplify the dataset.
Step 9: Get detailed information about the dataset, including data types
and any missing values.
```

```
Step 10: Create a copy of the data and store it in a new DataFrame for
Exploratory Data Analysis (EDA) without altering the original dataset.
step 11: stop the program
```

Program:

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 Data
data = pd.read_csv("/content/traffic.csv")
data.head()
```

```
data["DateTime"]= pd.to_datetime(data["DateTime"])
data = data.drop(["ID"], axis=1) #dropping IDs
data.info()
```

```
#df to be used for EDA
df=data.copy()
#Let's plot the Timeseries
colors = [ "#FFD4DB","#BBE7FE","#D3B5E5","#dfe2b6"]
plt.figure(figsize=(20,4),facecolor="#627D78")
Time_series=sns.lineplot(x=df['DateTime'],y="Vehicles",data=df,
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")
```

```
df["Year"]= df['DateTime'].dt.year
df["Month"]= df['DateTime'].dt.month
df["Date_no"]= df['DateTime'].dt.day
df["Hour"]= df['DateTime'].dt.hour
df["Day"]= df.DateTime.dt.strftime("%A")
df.head()
```

```
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["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")
```

```
sns.pairplot(data=df, hue= "Junction",palette=colors)
```

```
df_J = data.pivot(columns="Junction", index="DateTime")
df_J.describe()
```

```
df_1 = df_J[[('Vehicles', 1)]]
df_2 = df_J[[('Vehicles', 2)]]
df_3 = df_J[[('Vehicles', 3)]]
df_4 = df_J[[('Vehicles', 4)]]
df_4 = df_4.dropna() #Junction 4 has limited data only for a few months

#Dropping level one in dfs's index as it is a multi index data frame
list_dfs = [df_1, df_2, df_3, df_4]
for i in list_dfs:
    i.columns= i.columns.droplevel(level=1)

#Function to plot comparitive plots of dataframes
def Sub_Plots4(df_1, df_2,df_3,df_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=df_1,color=colors[0])
    #pl_1=plt.ylabel()
    axes[0].set(ylabel ="Junction 1")
    #J2
    pl_2=sns.lineplot(ax=axes[1],data=df_2,color=colors[1])
    axes[1].set(ylabel ="Junction 2")
    #J3
    pl_3=sns.lineplot(ax=axes[2],data=df_3,color=colors[2])
    axes[2].set(ylabel ="Junction 3")
    #J4
    pl_4=sns.lineplot(ax=axes[3],data=df_4,color=colors[3])
    axes[3].set(ylabel ="Junction 4")


#Plotting the dataframe to check for stationarity
Sub_Plots4(df_1.Vehicles,
df_2.Vehicles,df_3.Vehicles,df_4.Vehicles,"Dataframes Before
Transformation")
```

```
# Normalize Function
def Normalize(df,col):
    average = df[col].mean()
    stdev = df[col].std()
    df_normalized = (df[col] - average) / stdev
    df_normalized = df_normalized.to_frame()
    return df_normalized, average, stdev

# Differencing Function
def Difference(df,col, interval):
    diff = []
    for i in range(interval, len(df)):
        value = df[col][i] - df[col][i - interval]
        diff.append(value)
    return diff
```

```
df_N1, av_J1, std_J1 = Normalize(df_1, "Vehicles")
Diff_1 = Difference(df_N1, col="Vehicles", interval=(24*7)) #taking a
week's diffrence
df_N1 = df_N1[24*7:]
df_N1.columns = ["Norm"]
df_N1["Diff"]= Diff_1

df_N2, av_J2, std_J2 = Normalize(df_2, "Vehicles")
Diff_2 = Difference(df_N2, col="Vehicles", interval=(24)) #taking a
day's diffrence
df_N2 = df_N2[24:]
df_N2.columns = ["Norm"]
```

```
df_N2["Diff"]= Diff_2

df_N3, av_J3, std_J3 = Normalize(df_3, "Vehicles")
Diff_3 = Difference(df_N3, col="Vehicles", interval=1) #taking an hour's
diffrence
df_N3 = df_N3[1:]
df_N3.columns = ["Norm"]
df_N3["Diff"]= Diff_3

df_N4, av_J4, std_J4 = Normalize(df_4, "Vehicles")
Diff_4 = Difference(df_N4, col="Vehicles", interval=1) #taking an hour's
diffrence
df_N4 = df_N4[1:]
df_N4.columns = ["Norm"]
df_N4["Diff"]= Diff_4
```

```
Sub_Plots4(df_N1.Diff, df_N2.Diff,df_N3.Diff,df_N4.Diff,"Dataframes
After Transformation")
```

```
def Stationary_check(df):
    check = adfuller(df.dropna())
    print(f"ADF Statistic: {check[0]}")
    print(f"p-value: {check[1]}")
    print("Critical Values:")
    for key, value in check[4].items():
        print('\t%s: %.3f' % (key, value))
    if check[0] > check[4]["1%"]:
        print("Time Series is Non-Stationary")
    else:
        print("Time Series is Stationary")


#Checking if the series is stationary

List_df_ND = [ df_N1["Diff"], df_N2["Diff"], df_N3["Diff"],
df_N4["Diff"]]
print("Checking the transformed series for stationarity:")
for i in List_df_ND:
    print("\n")
    Stationary_check(i)
```

Output:

Untitled28.ipynb
File Edit View Insert Runtime Tools Help All changes saved

```
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")
```

```
data = pd.read_csv("/content/traffic.csv")
data.head()
```

	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

Next steps: Generate code with data View recommended plots New interactive sheet

0s completed at 9:05 AM

Untitled28.ipynb
File Edit View Insert Runtime Tools Help All changes saved

4	2015-11-01 04:00:00	1	9	20151101041

Next steps: Generate code with data View recommended plots New interactive sheet

```
data["DateTime"]= pd.to_datetime(data["DateTime"])
data = data.drop(["ID"], axis=1) #dropping IDs
data.info()
```

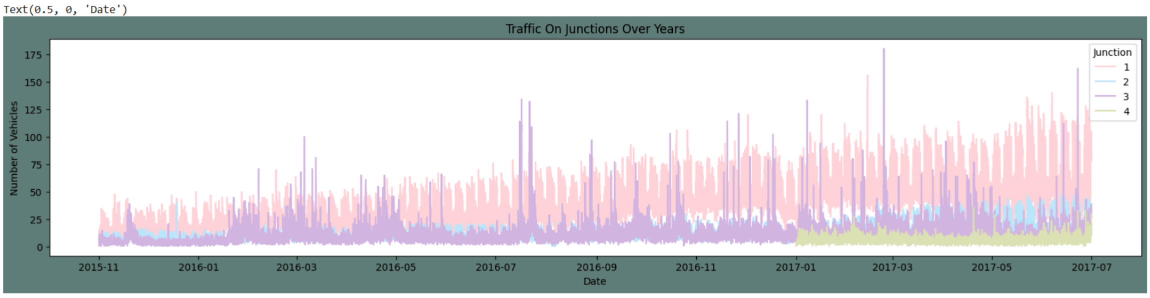
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48120 entries, 0 to 48119
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   DateTime  48120 non-null  datetime64[ns]
 1   Junction  48120 non-null  int64
 2   Vehicles  48120 non-null  int64
dtypes: datetime64[ns](1), int64(2)
memory usage: 1.1 MB
```

```
#df to be used for EDA
df=data.copy()
#Let's plot the Timeseries
colors = [ "#FFD4DB","#BBE7FE","#D3B5E5","#dfe2b6"]
plt.figure(figsize=(20,4),facecolor="#627D78")
Time_series=sns.lineplot(x=df['DateTime'],y="Vehicles",data=df, 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")
```

Untitled28.ipynb
File Edit View Insert Runtime Tools Help All changes saved

```
#df to be used for EDA
df=data.copy()
#Let's plot the Timeseries
colors = [ "#FFD4DB","#BBE7FE","#D3B5E5","#dfe2b6"]
plt.figure(figsize=(20,4),facecolor="#627D78")
Time_series=sns.lineplot(x=df['DateTime'],y="Vehicles",data=df, 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")
```


```
Text(0.5, 0, 'Date')
```



Untitled28.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

[7]
```
df["Year"]= df['DateTime'].dt.year
df["Month"]= df['DateTime'].dt.month
df["Date_no"]= df['DateTime'].dt.day
df["Hour"]= df['DateTime'].dt.hour
df["Day"]= df.DateTime.dt.strftime("%A")
df.head()
```

	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

Next steps: Generate code with df | View recommended plots | New interactive sheet

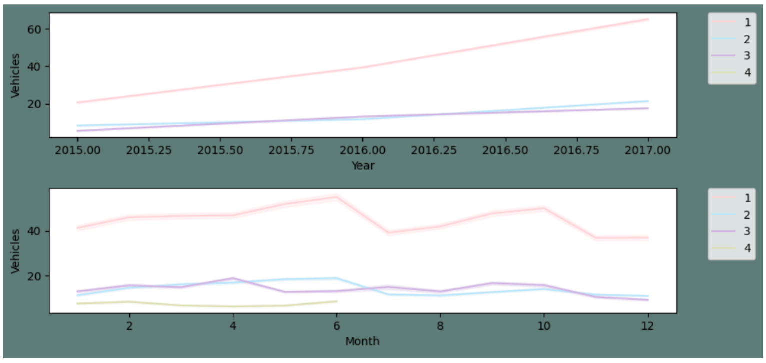
Untitled28.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

[8]
```
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.)
```

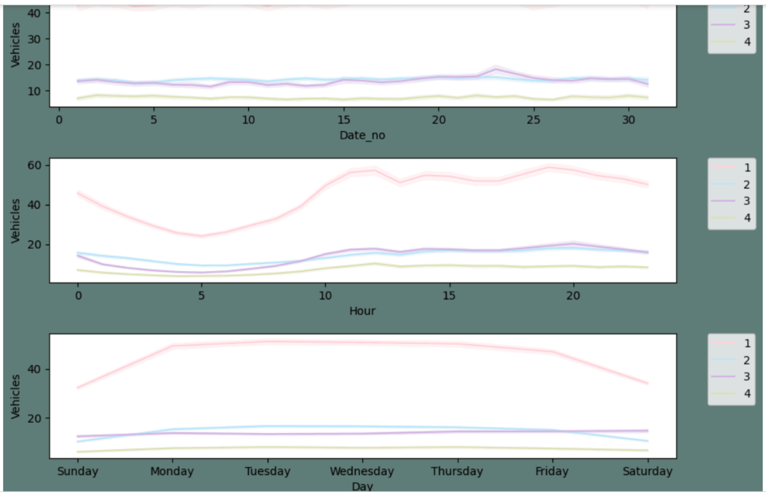


Untitled28.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

[8]

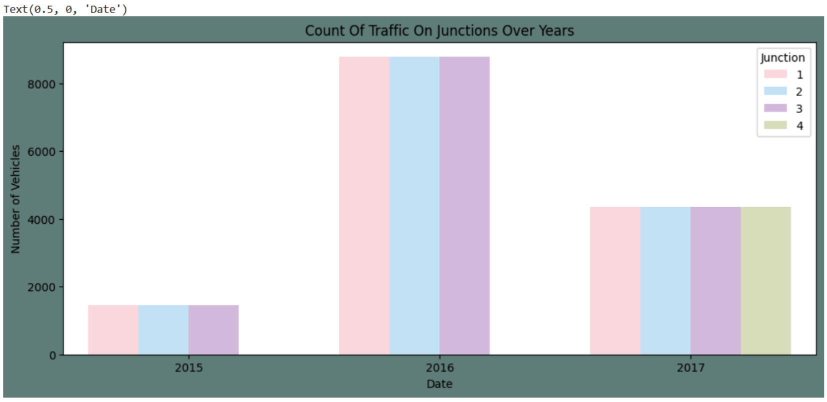


Untitled28.ipynb

File Edit View Insert Runtime Tools Help All changes saved

```
[9] count.set_title("Count Of Traffic On Junctions Over Years")
    count.set_ylabel("Number of Vehicles")
    count.set_xlabel("Date")
```

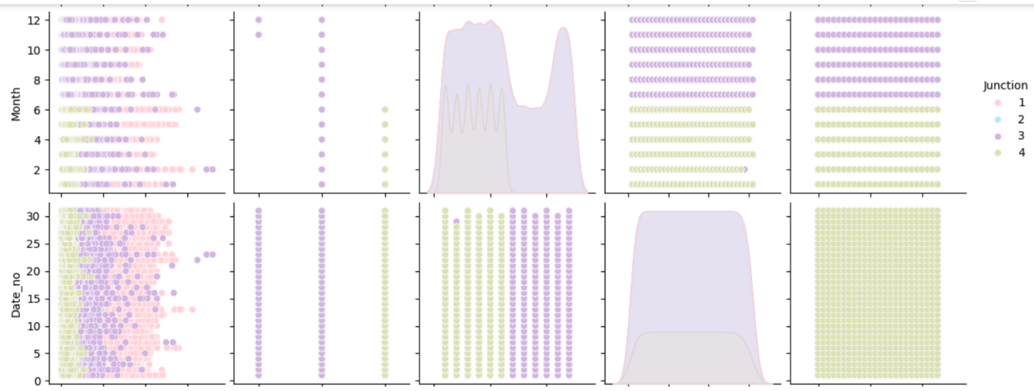
Text(0.5, 0, 'Date')



Untitled28.ipynb

File Edit View Insert Runtime Tools Help All changes saved

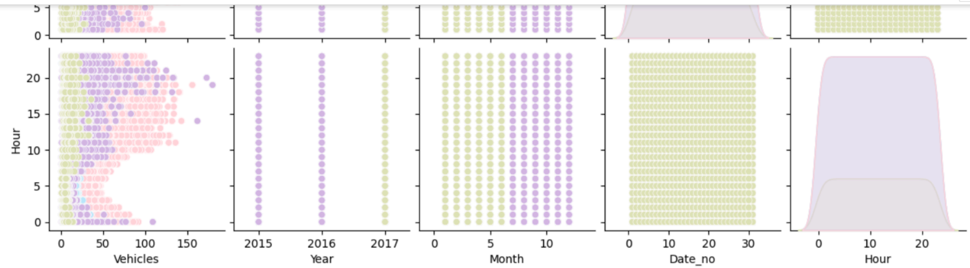
[11]



Untitled28.ipynb

File Edit View Insert Runtime Tools Help All changes saved

[11]

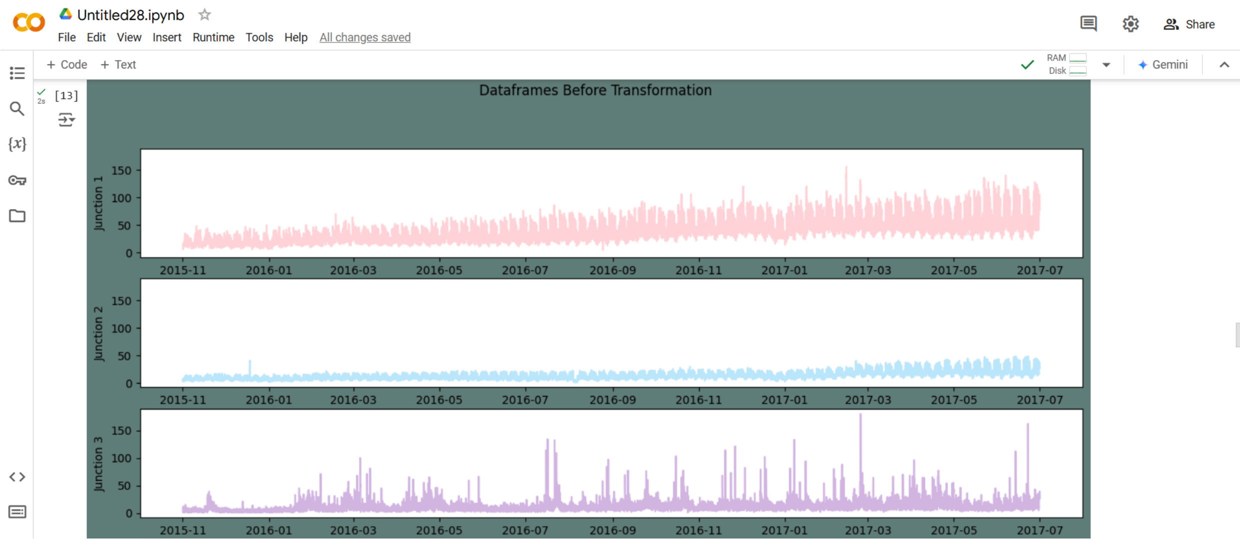


Untitled28.ipynb

File Edit View Insert Runtime Tools Help All changes saved

```
[12] df_J = data.pivot(columns="Junction", index="DateTime")
     df_J.describe()
```

	Vehicles			
Junction	1	2	3	4
count	14592.000000	14592.000000	14592.000000	4344.000000
mean	45.052906	14.253221	13.694010	7.251611
std	23.008345	7.401307	10.436005	3.521455
min	5.000000	1.000000	1.000000	1.000000
25%	27.000000	9.000000	7.000000	5.000000
50%	40.000000	13.000000	11.000000	7.000000
75%	59.000000	17.000000	18.000000	9.000000
max	156.000000	48.000000	180.000000	36.000000



Untitled28.ipynb

File Edit View Insert Runtime Tools Help All changes saved

[17]

```
ADF Statistic: -15.265303390415434
p-value: 4.798539876396819e-28
Critical Values:
        1%: -3.431
        5%: -2.862
        10%: -2.567
Time Series is Stationary


ADF Statistic: -21.795891026940144
p-value: 0.0
Critical Values:
        1%: -3.431
        5%: -2.862
        10%: -2.567
Time Series is Stationary


ADF Statistic: -28.001759908832977
p-value: 0.0
Critical Values:
        1%: -3.431
        5%: -2.862
        10%: -2.567
Time Series is Stationary


ADF Statistic: -17.979092563052305
p-value: 2.7787875325953405e-30
Critical Values:
        1%: -3.432
        5%: -2.862
        10%: -2.567
Time Series is Stationary
```

Result:

Thus the system was trained successfully and the prediction was carried out.