TRAFFIC MANAGEMENT SYSTEM

TEAM MEMBER

953121106060 : VIDHYA M

PHASE 5 SUBMISSION DOCUMENT

Project Title : TRAFFIC MANAGEMENT SYSTEM

Phase 5 : Project Documentation And Submission

Topic : In this section we will document the complete project and prepare it for submission

TRAFFIC MANAGEMENT SYSTEM

INTRODUCTION

- In our rapidly urbanizing world, managing traffic flow has become a paramount challenge. Urban centers are experiencing unprecedented population growth, leading to increased congestion, longer commuting times, and environmental concerns. Traffic management systems have emerged as a critical solution to address these issues efficiently. These systems leverage advanced technologies and data-driven approaches to optimize traffic flow, enhance road safety, reduce congestion, and minimize environmental impact.
- The rise in the number of vehicles on roads, coupled with limited infrastructure expansion, has led to traffic bottlenecks and delays. Urban planners and transportation authorities face the daunting task of ensuring smooth vehicular movement while prioritizing pedestrian safety and environmental sustainability. Traffic accidents, unexpected road incidents, and adverse weather conditions further compound these challenges. The need for an intelligent, adaptive, and integrated Traffic Management System has never been more pressing.
- Modern traffic management systems rely on an array of cutting-edge technologies such as IoT sensors, real-time data analytics, artificial intelligence, and machine learning algorithms. These technologies enable the collection and analysis of vast amounts of data from various sources, including traffic cameras, GPS devices, weather stations, and social media platforms. By

processing this data in real-time, traffic engineers can make informed decisions, implement dynamic traffic control strategies, and provide timely information to commuters.

- By analyzing traffic patterns and adjusting signal timings in real-time, traffic management systems optimize the flow of vehicles, reducing congestion and travel times.
- These systems detect accidents, monitor road conditions, and alert emergency services promptly, ensuring rapid response to incidents and minimizing potential hazards.
- By reducing traffic congestion and optimizing routes, traffic management systems contribute to lowering carbon emissions and promoting eco-friendly transportation.
- Traffic management systems provide real-time information to commuters through digital signage, mobile apps, and online platforms, enabling informed decision-making regarding route selection and travel times.
- Data collected by traffic management systems offer valuable insights for urban planning, helping design road networks that are efficient, safe, and adaptable to future needs.

GIVEN DATA SET

Junction 1 (H-1)	Junction 2 (H-1)	Junction 3 (H-1)	Junction 4 (H-1)	Junction 1 (H)	Junction 2 (H)	Junction 3 (H)	Junction 4 (H)	
DateTime								
2015-11-01 01:00:00	15.0	6.0	9.0	0.0	13.0	6.0	7.0	0.0
2015-11-01 02:00:00	13.0	6.0	7.0	0.0	10.0	5.0	5.0	0.0
2015-11-01 03:00:00	10.0	5.0	5.0	0.0	7.0	6.0	1.0	0.0
2015-11-01 04:00:00	7.0	6.0	1.0	0.0	9.0	7.0	2.0	0.0
2015-11-01 05:00:00	9.0	7.0	2.0	0.0	6.0	2.0	2.0	0.0
...	...	...	...	...	...	...	...	...

Junction 1 (H-1)	Junction 2 (H-1)	Junction 3 (H-1)	Junction 4 (H-1)	Junction 1 (H)	Junction 2 (H)	Junction 3 (H)	Junction 4 (H)	
DateTime								
2017-06-30 19:00:00	95.0	34.0	38.0	17.0	105.0	34.0	33.0	11.0
2017-06-30 20:00:00	105.0	34.0	33.0	11.0	96.0	35.0	31.0	30.0
2017-06-30 21:00:00	96.0	35.0	31.0	30.0	90.0	31.0	28.0	16.0
2017-06-30 22:00:00	90.0	31.0	28.0	16.0	84.0	29.0	26.0	22.0
2017-06-30 23:00:00	84.0	29.0	26.0	22.0	78.0	27.0	39.0	12.0

GIVEN PROGRAM

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
In [2]:
# Display only top 5 and bottom 5 rows
pd.set_option('display.max_rows', 10)
```

```
In [3]:
def nl():
    print('\n')
```

```
In [4]:
df_train = pd.read_csv('/kaggle/input/DataSet/train.csv', parse_dates=[0], infer_d
atetime_format=True)
df_test = pd.read_csv('/kaggle/input/DataSet/test.csv', parse_dates=[0], infer_dat
etime_format=True)
```

```
In [5]:
nl()
print('Size of training set: ' + str(df_train.shape))
print(' Size of testing set: ' + str(df_test.shape))

nl()
print('Columns in train: ' + str(df_train.columns.tolist()))
print(' Columns in test: ' + str(df_test.columns.tolist()))

df_train.head()
```

```
Size of training set: (48120, 4)
 Size of testing set: (11808, 3)


Columns in train: ['DateTime', 'Junction', 'Vehicles', 'ID']
 Columns in test: ['DateTime', 'Junction', 'ID']
```

Out[5]:

	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

EDA

In [6]:
```
df_tmp = df_train.set_index(['Junction', 'DateTime'])
```

In [7]:
```
level_values = df_tmp.index.get_level_values
```

In [8]:
```
time_targets = df_tmp.groupby([level_values(0)] + [pd.Grouper(freq='1M', level=-1)
])['Vehicles'].sum()
time_targets
```

Out[8]:
```
Junction  DateTime
1         2015-11-30    14736
          2015-12-31    15487
          2016-01-31    17940
          2016-02-29    20813
          2016-03-31    22215
                        ...
4         2017-02-28     5564
          2017-03-31     4931
          2017-04-30     4454
          2017-05-31     4877
          2017-06-30     6097
```

```
Name: Vehicles, Length: 66, dtype: int64
```

In [9]:
```
del df_tmp
del time_targets
```

lag_features

In [10]:
```
train = df_train.pivot(index='DateTime',columns='Junction', values='Vehicles')
train
```

Out[10]:

Junction	1	2	3	4
DateTime				
2015-11-01 00:00:00	15.0	6.0	9.0	NaN
2015-11-01 01:00:00	13.0	6.0	7.0	NaN
2015-11-01 02:00:00	10.0	5.0	5.0	NaN
2015-11-01 03:00:00	7.0	6.0	1.0	NaN
2015-11-01 04:00:00	9.0	7.0	2.0	NaN
...	...	...	...	...
2017-06-30 19:00:00	105.0	34.0	33.0	11.0
2017-06-30 20:00:00	96.0	35.0	31.0	30.0

Junction	1	2	3	4
DateTime				
2017-06-30 21:00:00	90.0	31.0	28.0	16.0
2017-06-30 22:00:00	84.0	29.0	26.0	22.0
2017-06-30 23:00:00	78.0	27.0	39.0	12.0

14592 rows × 4 columns

In [11]:
```
train = train.fillna(0) #Fill NaNs with 0 vehicles
```

Generate lag features

In [12]:
```
def gen_lag_features(df, n_in=1, n_out=1, dropnan=True):
    """
    Arguments:
        data: Dataframe of observations.
        n_in: Number of lag observations as input (X).
        n_out: Number of forecast observations as output (y).
        dropnan: Boolean whether or not to drop rows with NaN values.
    Returns:
        Dataframe.
    """
    n_vars = df.shape[1]
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('Junction %d (H-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('Junction %d (H)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('Junction %d (H+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = pd.concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
```

```
        agg.dropna(inplace=True)
    return agg
```

```
In [13]:
Xy_train = gen_lag_features(train)
Xy_train
```

Out[13]:

	Junction 1 (H-1)	Junction 2 (H-1)	Junction 3 (H-1)	Junction 4 (H-1)	Junction 1 (H)	Junction 2 (H)	Junction 3 (H)	Junction 4 (H)
DateTime								
2015-11-01 01:00:00	15.0	6.0	9.0	0.0	13.0	6.0	7.0	0.0
2015-11-01 02:00:00	13.0	6.0	7.0	0.0	10.0	5.0	5.0	0.0
2015-11-01 03:00:00	10.0	5.0	5.0	0.0	7.0	6.0	1.0	0.0
2015-11-01 04:00:00	7.0	6.0	1.0	0.0	9.0	7.0	2.0	0.0
2015-11-01 05:00:00	9.0	7.0	2.0	0.0	6.0	2.0	2.0	0.0
...	...	...	...	...	...	...	...	...
2017-06-30 19:00:00	95.0	34.0	38.0	17.0	105.0	34.0	33.0	11.0
2017-06-30 20:00:00	105.0	34.0	33.0	11.0	96.0	35.0	31.0	30.0

	Junction 1 (H-1)	Junction 2 (H-1)	Junction 3 (H-1)	Junction 4 (H-1)	Junction 1 (H)	Junction 2 (H)	Junction 3 (H)	Junction 4 (H)
DateTime								
2017-06-30 21:00:00	96.0	35.0	31.0	30.0	90.0	31.0	28.0	16.0
2017-06-30 22:00:00	90.0	31.0	28.0	16.0	84.0	29.0	26.0	22.0
2017-06-30 23:00:00	84.0	29.0	26.0	22.0	78.0	27.0	39.0	12.0

14591 rows × 8 columns

Normalize features

In [14]:
```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
# scaler = MinMaxScaler(feature_range=(-1, 1))
# scaler = StandardScaler()
scaler = MinMaxScaler(feature_range=(0, 1))
Xy_train[Xy_train.columns] = scaler.fit_transform(Xy_train[Xy_train.columns])

Xy_train
```

Out[14]:

	Junction 1 (H-1)	Junction 2 (H-1)	Junction 3 (H-1)	Junction 4 (H-1)	Junction 1 (H)	Junction 2 (H)	Junction 3 (H)	Junction 4 (H)
DateTime								
2015-11-01 01:00:00	0.066225	0.106383	0.044693	0.000000	0.052980	0.106383	0.033520	0.000000
2015-11-01 02:00:00	0.052980	0.106383	0.033520	0.000000	0.033113	0.085106	0.022346	0.000000

	Junction 1 (H-1)	Junction 2 (H-1)	Junction 3 (H-1)	Junction 4 (H-1)	Junction 1 (H)	Junction 2 (H)	Junction 3 (H)	Junction 4 (H)
DateTime								
2015-11-01 03:00:00	0.033113	0.085106	0.022346	0.000000	0.013245	0.106383	0.000000	0.000000
2015-11-01 04:00:00	0.013245	0.106383	0.000000	0.000000	0.026490	0.127660	0.005587	0.000000
2015-11-01 05:00:00	0.026490	0.127660	0.005587	0.000000	0.006623	0.021277	0.005587	0.000000
...	...	...	...	...	...	...	...	...
2017-06-30 19:00:00	0.596026	0.702128	0.206704	0.472222	0.662252	0.702128	0.178771	0.305556
2017-06-30 20:00:00	0.662252	0.702128	0.178771	0.305556	0.602649	0.723404	0.167598	0.833333
2017-06-30 21:00:00	0.602649	0.723404	0.167598	0.833333	0.562914	0.638298	0.150838	0.444444
2017-06-30 22:00:00	0.562914	0.638298	0.150838	0.444444	0.523179	0.595745	0.139665	0.611111
2017-06-30 23:00:00	0.523179	0.595745	0.139665	0.611111	0.483444	0.553191	0.212291	0.333333

14591 rows × 8 columns

Split train and valid (and normalize for real)

In [15]:
```
X_train = Xy_train[Xy_train.index < '2017-04-01'].iloc[:,0:4]
```

```
X_train
```

Out[15]:

	Junction 1 (H-1)	Junction 2 (H-1)	Junction 3 (H-1)	Junction 4 (H-1)
DateTime				
2015-11-01 01:00:00	0.066225	0.106383	0.044693	0.000000
2015-11-01 02:00:00	0.052980	0.106383	0.033520	0.000000
2015-11-01 03:00:00	0.033113	0.085106	0.022346	0.000000
2015-11-01 04:00:00	0.013245	0.106383	0.000000	0.000000
2015-11-01 05:00:00	0.026490	0.127660	0.005587	0.000000
...	...	...	...	...
2017-03-31 19:00:00	0.476821	0.574468	0.178771	0.166667
2017-03-31 20:00:00	0.496689	0.531915	0.156425	0.222222
2017-03-31 21:00:00	0.483444	0.638298	0.156425	0.222222
2017-03-31 22:00:00	0.403974	0.574468	0.150838	0.250000
2017-03-31 23:00:00	0.423841	0.553191	0.162011	0.166667

12407 rows × 4 columns

In [16]:

```
y_train = Xy_train[Xy_train.index < '2017-04-01'].iloc[:,4:]
```

```
y_train
```

Out[16]:

	Junction 1 (H)	Junction 2 (H)	Junction 3 (H)	Junction 4 (H)
DateTime				
2015-11-01 01:00:00	0.052980	0.106383	0.033520	0.000000
2015-11-01 02:00:00	0.033113	0.085106	0.022346	0.000000
2015-11-01 03:00:00	0.013245	0.106383	0.000000	0.000000
2015-11-01 04:00:00	0.026490	0.127660	0.005587	0.000000
2015-11-01 05:00:00	0.006623	0.021277	0.005587	0.000000
...	...	...	...	...
2017-03-31 19:00:00	0.496689	0.531915	0.156425	0.222222
2017-03-31 20:00:00	0.483444	0.638298	0.156425	0.222222
2017-03-31 21:00:00	0.403974	0.574468	0.150838	0.250000
2017-03-31 22:00:00	0.423841	0.553191	0.162011	0.166667
2017-03-31 23:00:00	0.417219	0.553191	0.162011	0.166667

12407 rows × 4 columns

Reshape the Data

In [17]:
```
print(X_train.shape, y_train.shape)
```

```
(12407, 4) (12407, 4)
```

In [18]:
```
X_train = np.expand_dims(X_train.values, axis=2)
print(X_train.shape)

y_train = y_train.values
print(y_train.shape)
```

```
(12407, 4, 1)
(12407, 4)
```

Modeling

In [19]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.initializers import he_normal
import keras.backend as K

def root_mean_squared_error(y_true, y_pred):
        return K.sqrt(K.mean(K.square(y_pred - y_true), axis=-1))
```

In [20]:
```
# Initialising the RNN
regressor = Sequential()

# Adding the input layer and the LSTM layer
regressor.add(LSTM(units = 50,
                   activation = 'relu', # default is tanh
                   kernel_initializer = he_normal(seed=0),
                   input_shape = (None, 1)))

# Output for 4 junctions
regressor.add(Dense(units = 4))

# Compiling the RNN
regressor.compile(optimizer = 'adam', loss = root_mean_squared_error)
```

```
2023-01-17 13:10:05.795741: I tensorflow/core/common_runtime/process_util.cc:1
46] Creating new thread pool with default inter op setting: 2. Tune using inte
r_op_parallelism_threads for best performance.
```

In [21]:
```
# Fitting the RNN to the Training set
regressor.fit(X_train, y_train, batch_size = 128, epochs = 10, verbose = 1)
```

```
2023-01-17 13:10:06.112880: I tensorflow/compiler/mlir/mlir_graph_optimization
_pass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)
Epoch 1/10
97/97 [==============================] - 2s 5ms/step - loss: 0.0846
Epoch 2/10
97/97 [==============================] - 0s 5ms/step - loss: 0.0396
Epoch 3/10
97/97 [==============================] - 1s 5ms/step - loss: 0.0376
Epoch 4/10
97/97 [==============================] - 1s 5ms/step - loss: 0.0360
```

```
Epoch 5/10
97/97 [==============================] - 1s 6ms/step - loss: 0.0347
Epoch 6/10
97/97 [==============================] - 1s 6ms/step - loss: 0.0338
Epoch 7/10
97/97 [==============================] - 1s 6ms/step - loss: 0.0333
Epoch 8/10
97/97 [==============================] - 1s 5ms/step - loss: 0.0330
Epoch 9/10
97/97 [==============================] - 1s 5ms/step - loss: 0.0328
Epoch 10/10
97/97 [==============================] - 0s 5ms/step - loss: 0.0328
```

Out[21]:

```
<keras.callbacks.History at 0x7f9e8afcf490>
```

Validating

In [22]:

```
X_valid = Xy_train[Xy_train.index >= '2017-04-01'].iloc[:,0:4]
X_valid
```

Out[22]:

	Junction 1 (H-1)	Junction 2 (H-1)	Junction 3 (H-1)	Junction 4 (H-1)
DateTime				
2017-04-01 00:00:00	0.417219	0.553191	0.162011	0.166667
2017-04-01 01:00:00	0.384106	0.510638	0.122905	0.166667
2017-04-01 02:00:00	0.317881	0.574468	0.078212	0.138889
2017-04-01 03:00:00	0.238411	0.361702	0.083799	0.111111
2017-04-01 04:00:00	0.225166	0.361702	0.055866	0.111111
...	...	...	...	...
2017-06-30 19:00:00	0.596026	0.702128	0.206704	0.472222

	Junction 1 (H-1)	Junction 2 (H-1)	Junction 3 (H-1)	Junction 4 (H-1)
DateTime				
2017-06-30 20:00:00	0.662252	0.702128	0.178771	0.305556
2017-06-30 21:00:00	0.602649	0.723404	0.167598	0.833333
2017-06-30 22:00:00	0.562914	0.638298	0.150838	0.444444
2017-06-30 23:00:00	0.523179	0.595745	0.139665	0.611111

2184 rows × 4 columns

In [23]:
```
X_valid = np.expand_dims(X_valid.values, axis=2)
y_pred = regressor.predict(X_valid)
```

In [24]:
```
# We rescale y in the integer count range
# To do that we must first reconcatenate with the X data as scaler expects a shape of 8

y_pred = scaler.inverse_transform(np.concatenate((X_valid.squeeze(), y_pred), axis = 1))[:, 4:]
y_pred
```

Out[24]:
```
array([[64.11693594, 23.68348882, 25.74211836,  6.7378158 ],
       [60.57301778, 22.05180272, 20.59766436,  6.59938967],
       [54.43582478, 22.45377263, 13.3742146 ,  5.35569495],
       ...,
       [76.16106272, 29.0721696 , 26.24132636, 18.22499657],
       [88.44806713, 28.38077956, 26.44887759, 13.39262259],
       [78.60926619, 25.17595142, 25.8021808 , 15.08910155]])
```

In [25]:
```
y_truth = train[train.index >= '2017-04-01']
y_truth
```

Out[25]:

Junction	1	2	3	4

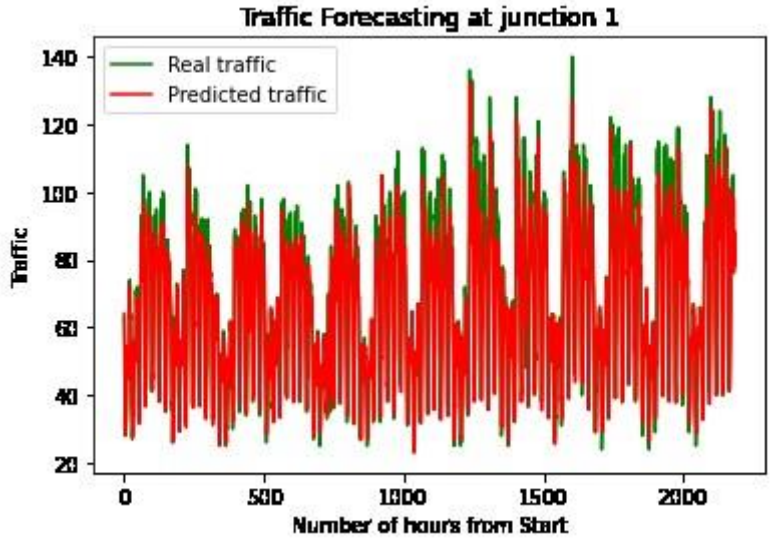
DateTime				
2017-04-01 00:00:00	63.0	25.0	23.0	6.0
2017-04-01 01:00:00	53.0	28.0	15.0	5.0
2017-04-01 02:00:00	41.0	18.0	16.0	4.0
2017-04-01 03:00:00	39.0	18.0	11.0	4.0
2017-04-01 04:00:00	31.0	15.0	10.0	1.0
...	...	...	...	...
2017-06-30 19:00:00	105.0	34.0	33.0	11.0
2017-06-30 20:00:00	96.0	35.0	31.0	30.0
2017-06-30 21:00:00	90.0	31.0	28.0	16.0
2017-06-30 22:00:00	84.0	29.0	26.0	22.0
2017-06-30 23:00:00	78.0	27.0	39.0	12.0

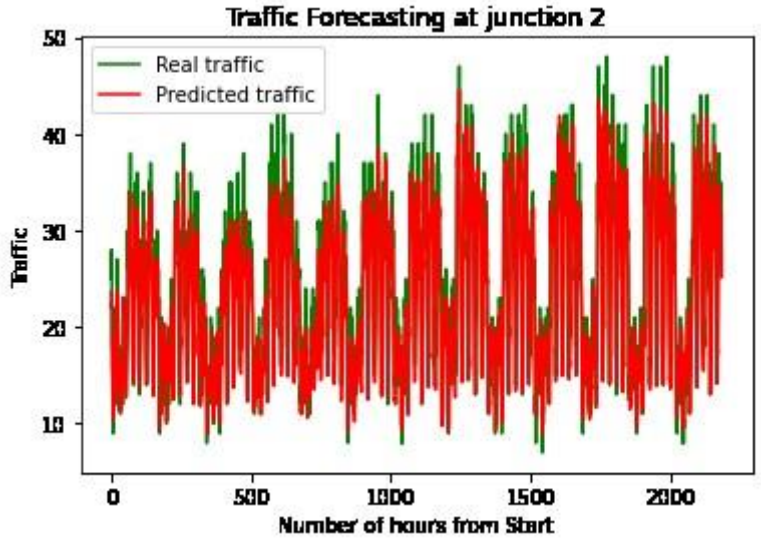
2184 rows × 4 columns

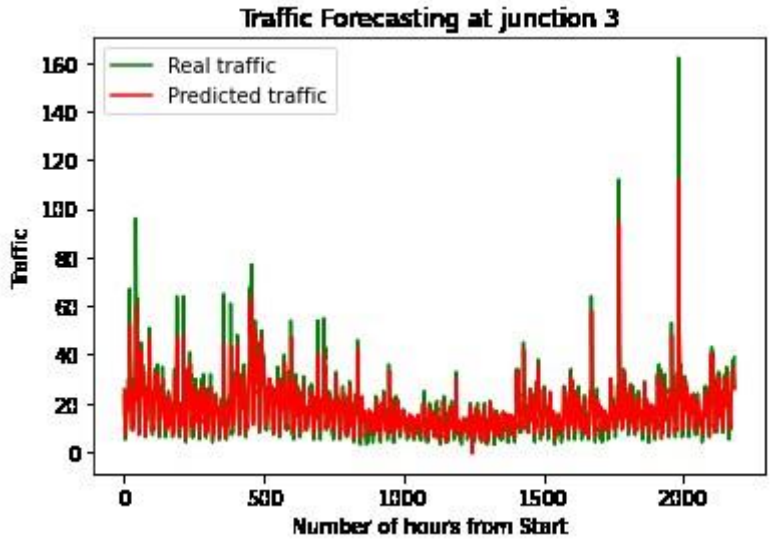
In [26]:

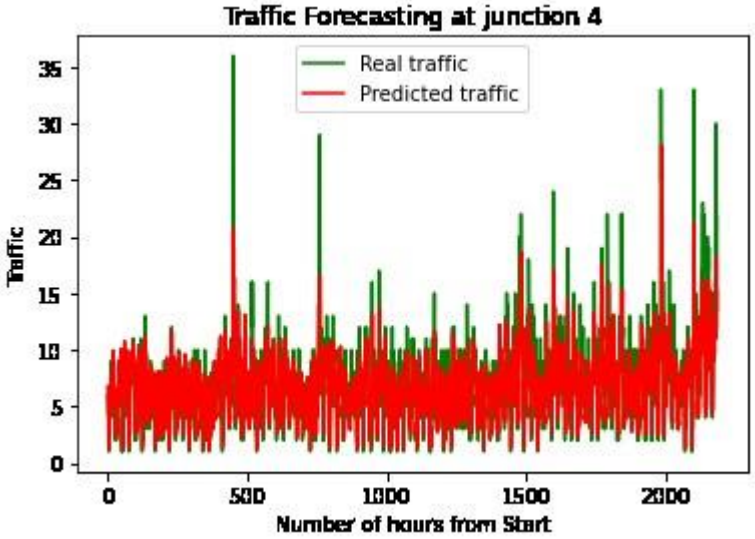
```
# Visualising Result for the junctions
for junction in range(4):
    plt.figure
    plt.plot(y_truth.values[:,junction], color = 'green', label = 'Real traffic')
    plt.plot(y_pred[:,junction], color = 'red', label = 'Predicted traffic')
    plt.title('Traffic Forecasting at junction %i' % (junction+1))
    plt.xlabel('Number of hours from Start')
```

```
plt.ylabel('Traffic')
plt.legend()
plt.show()
```









In [27]:
```
from sklearn.metrics import mean_squared_error
from math import sqrt

def rmse(y_true, y_pred):
    return sqrt(mean_squared_error(y_true, y_pred))
```

In [28]:
```
rmse(y_truth, y_pred)
```

Out[28]:
```
6.428161184734964
```

In [29]:
```
import pandas as pd
import numpy as np
trdf = pd.read_csv('/kaggle/input/DataSet/train.csv')
trainMat = trdf.to_numpy()
tedf = pd.read_csv('/kaggle/input/DataSet/test.csv')
testMat = tedf.to_numpy()
```

```
train = []
target = []
print (trainMat)
for i in trainMat:
    s = i[3]
    year = s / (10**7)
    s = s % (10**7)
    month = s / (10**5)
    s = s % (10**5)
    date = s / (10**3)
    s = s % (10**3)
    time = s / (10)
    s = s % (10)
    junction = s
    train.append([year, month, date, time, junction])
    target.append(i[2])
X = np.array(train)
y = np.array(target)
```

```
[['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]
 ...
 ['2017-06-30 21:00:00' 4 16 20170630214]
 ['2017-06-30 22:00:00' 4 22 20170630224]
 ['2017-06-30 23:00:00' 4 12 20170630234]]
```

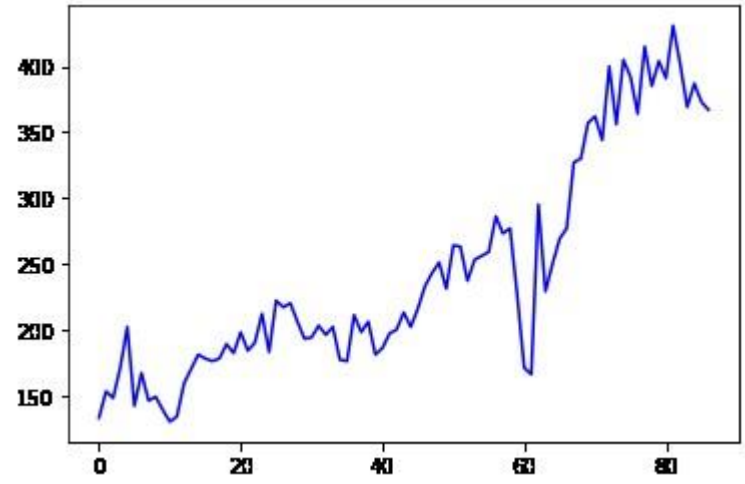
```
In [30]:
jun1 = []
jun2 = []
jun3 = []
jun4 = []
jun5 = []
jun = [jun1, jun2, jun3, jun4, jun5]
for i in range(0,len(train), 24):
    ct = 0
    for j in range(24):
        ct += target[i+j]
    #print train[i][4]
    jun[train[i][4]-1].append(ct)
jun[3] = [0]*(len(jun[0])- len(jun[3])) + jun[3]
print (len(jun[0]), len(jun[1]), len(jun[2]), len(jun[3]))

k = 7
week = [[] for i in range(k)]
for i in range(len(jun[1])):
    week[i%k].append(jun[1][i])
for i in range(k):
    print (np.mean(week[i]))


hour = [[] for i in range(24)]
for i in range(len(jun[0])*24+len(jun[1])*24, len(jun[0])*24+len(jun[1])*24+len(ju
n[2])*24):
    hour[i%24].append(target[i])

for i in range(24):
    print (np.mean(hour[i]))

temp = [-i for i in jun[3]]
```

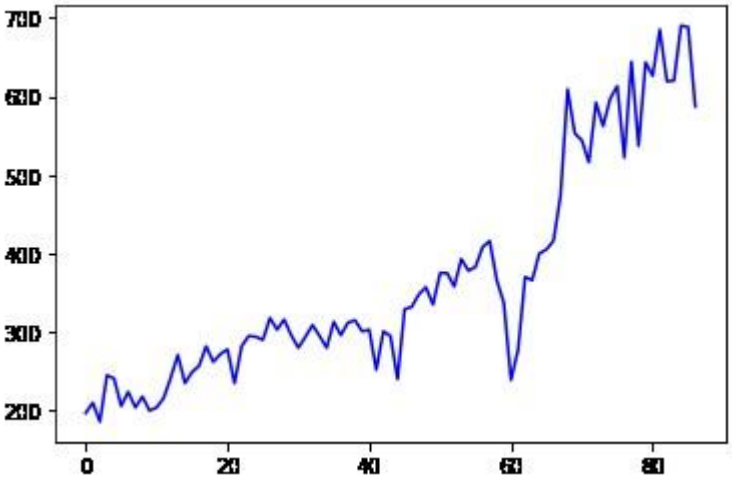
```
jun[4] = np.add(jun[2], temp)
```

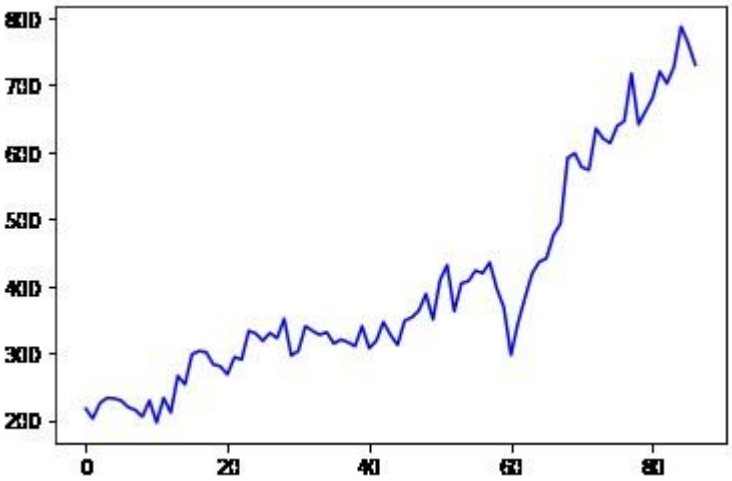
```
608 608 608 608
243.45977011494253
365.4942528735632
397.367816091954
394.0
384.632183908046
358.62068965517244
249.90697674418604
14.174342105263158
9.856907894736842
8.055921052631579
6.776315789473684
5.978618421052632
5.685855263157895
6.2368421052631575
7.550986842105263
9.057565789473685
11.429276315789474
15.004934210526315
17.1875
17.707236842105264
16.049342105263158
17.57236842105263
17.37171052631579
16.88157894736842
16.901315789473685
17.929276315789473
19.129934210526315
20.200657894736842
18.72203947368421
17.394736842105264
15.800986842105264
```

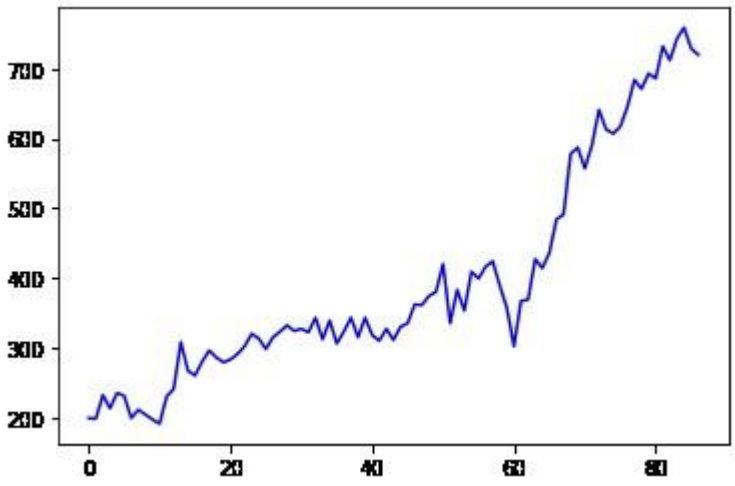
In [31]:

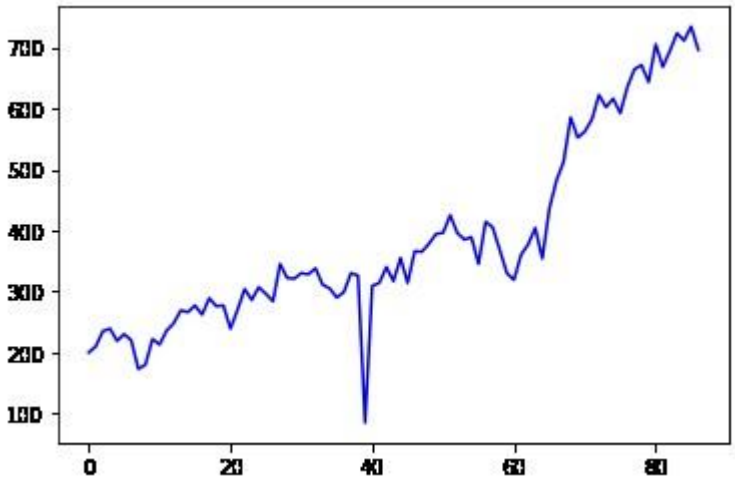
```
import matplotlib.pyplot as plt
for i in range(len(week)):
    plt.plot(week[i], 'blue')
    plt.savefig('f'+str(i)+'.png')
    plt.show()
```

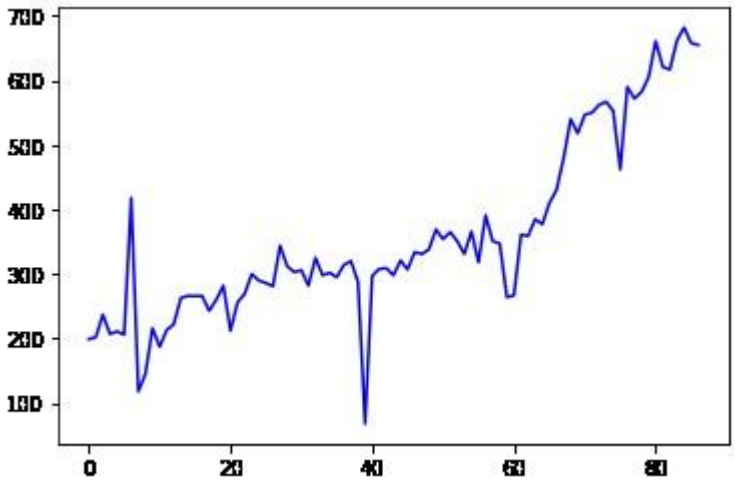

400
350
300
250
200
150
0
20
40
60
80

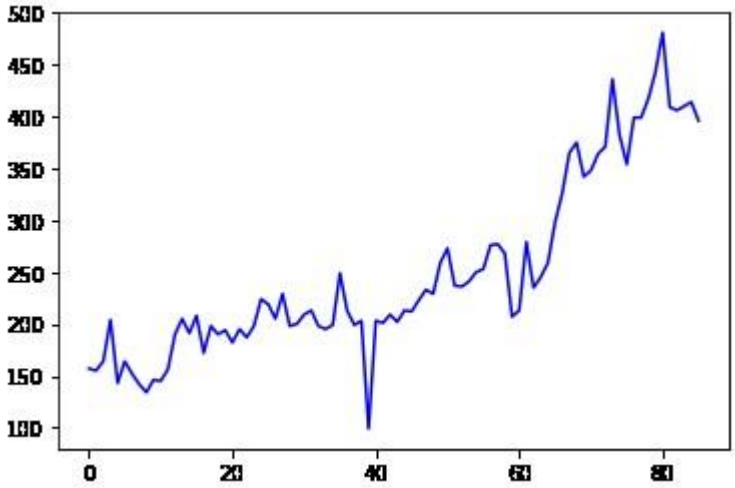

700
600
500
400
300
200
0
20
40
60
80


800
700
600
500
400
300
200
0
20
40
60
80


700
600
500
400
300
200
0
20
40
60
80

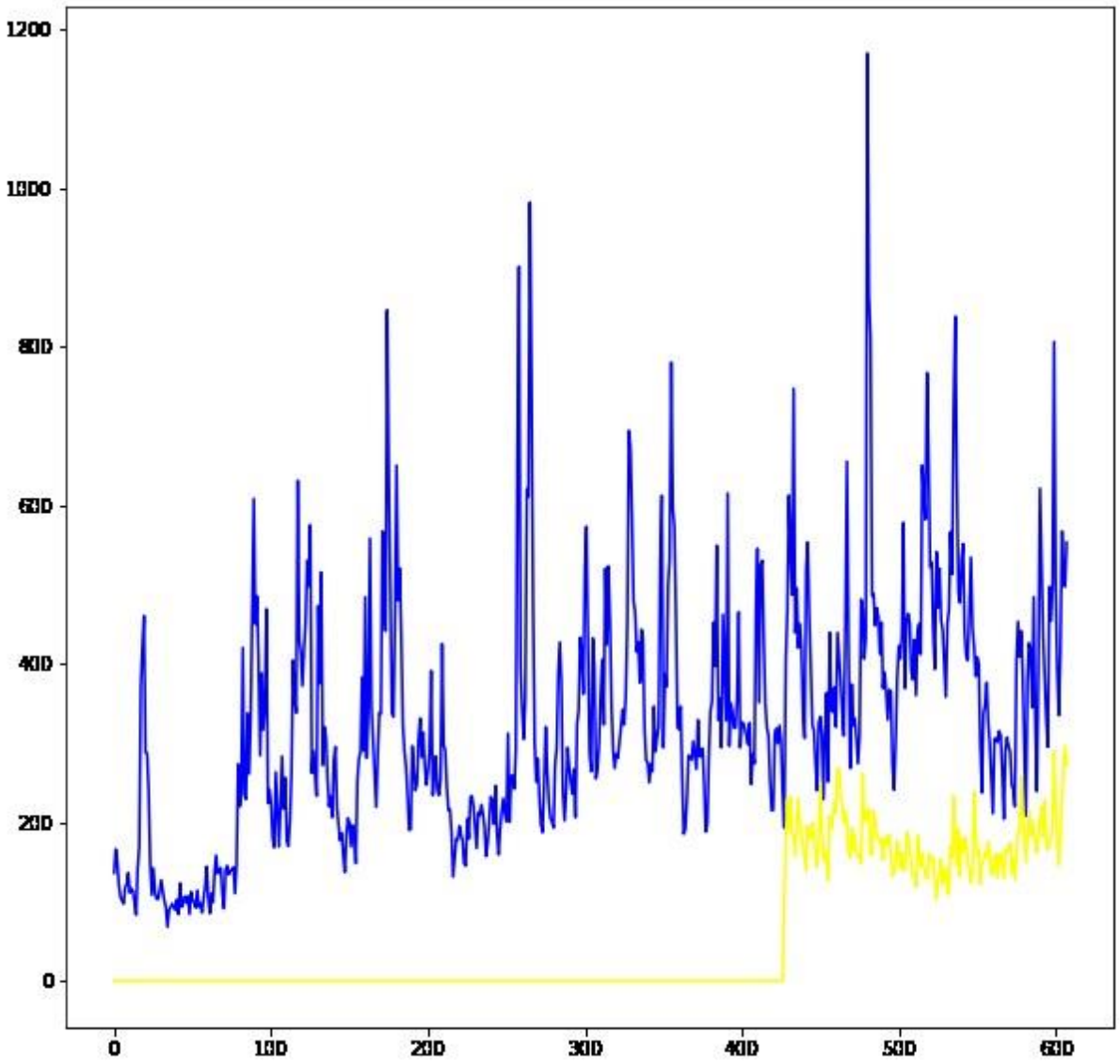

700
600
500
400
300
200
100
0
20
40
60
80

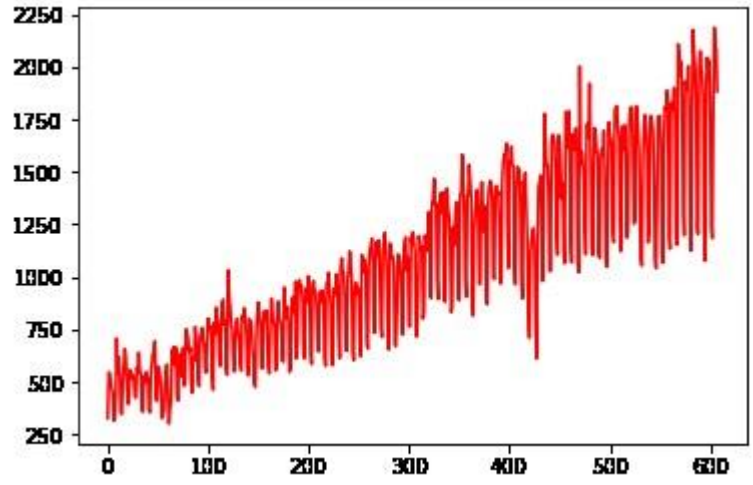


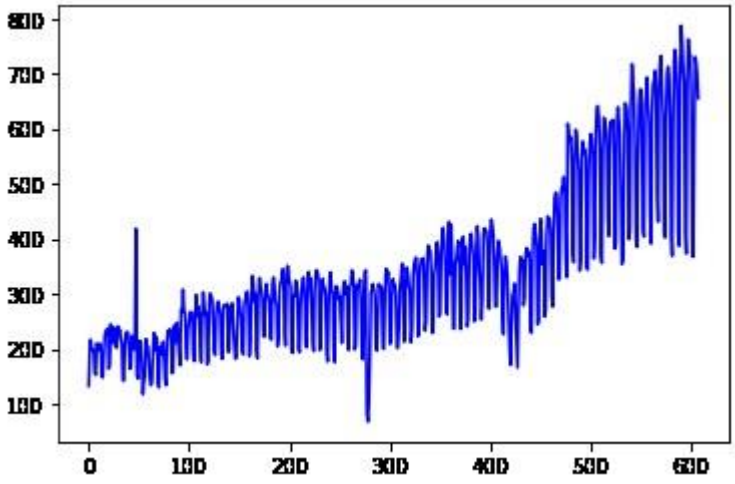


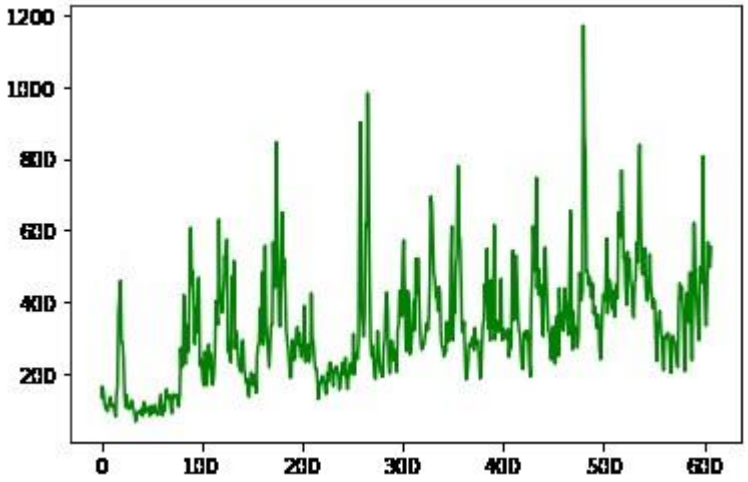
In [32]:

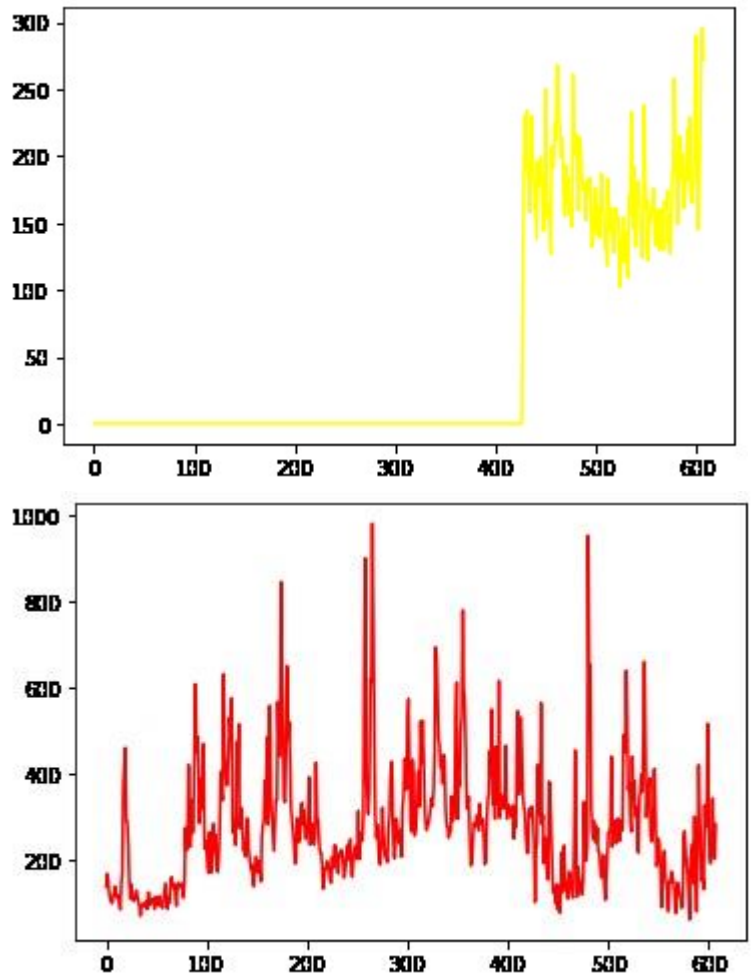
```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))

#plt.plot(jun[0], 'red')
plt.plot(jun[2], 'blue')
#plt.plot(jun[2], 'green')
plt.plot(jun[3], 'yellow')
plt.show()
plt.plot(jun[0], 'red')
plt.show()
plt.plot(jun[1], 'blue')
plt.show()
plt.plot(jun[2], 'green')
plt.show()
plt.plot(jun[3], 'yellow')
plt.show()
plt.plot(jun[4], 'red')
plt.show()
```


1200
1000
800
600
400
200
0
0
100
200
300
400
500
600


2250
2000
1750
1500
1250
1000
750
500
250
0
100
200
300
400
500
600


800
700
600
500
400
300
200
100
0
100
200
300
400
500
600


1200
1000
800
600
400
200
0
100
200
300
400
500
600



Random Forest Classifier

In [33]:

```
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=7)
from sklearn.ensemble import RandomForestClassifier
clf1 = RandomForestClassifier(criterion = 'entropy', min_samples_split = 150, min_
samples_leaf = 10, max_depth = 12, class_weight = 'balanced', n_estimators = 100)
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
from math import sqrt

'''
for train_index, test_index in skf.split(X, y):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    clf2 = clf1.fit(X_train, y_train)
    pred2 = clf1.predict(X_test)
    print pred1[:10], y_test[:10]
    rms = sqrt(mean_squared_error(y_test, pred2))
    print rms'''
```

Out[33]:

```
'\nfor train_index, test_index in skf.split(X, y):\n    X_train, X_test = X[tr
ain_index], X[test_index]\n    y_train, y_test = y[train_index], y[test_index]
\n    clf2 = clf1.fit(X_train, y_train)\n    pred2 = clf1.predict(X_test)\n
print pred1[:10], y_test[:10]\n    rms = sqrt(mean_squared_error(y_test, pred2
))\n    print rms'
```

```
In [34]:
clf1.fit(X, y)
pred = clf1.predict(X)
```

```
In [35]:
val1 = (accuracy_score(y, pred)*100)
print("*Accuracy score for RF: ", val1*5, "\n")
```

```
*Accuracy score for RF:  81.03699085619284
```

Decision Tree Classifier

```
In [36]:
linkcode
from sklearn import tree
DT = tree.DecisionTreeClassifier()
DT.fit(X, y)
predictions = DT.predict(X)
val2 = (accuracy_score(y, pred)*100)
print("*Accuracy score for DT: ", val2*5, "\n")
```

```
*Accuracy score for DT:  81.03699085619284
```

CONCLUSION

- In the face of burgeoning urbanization, Traffic Management Systems (TMS) stand as indispensable pillars of modern urban infrastructure. The challenges posed by escalating vehicular density, congestion, and environmental concerns necessitate intelligent, technology-driven solutions. Through this journey, we have explored the intricate web of sensors, data analytics, and real-time decision-making that constitute an effective TMS.
- One of the primary triumphs of TMS lies in its ability to alleviate traffic congestion. By harnessing the power of data analytics and predictive algorithms, traffic patterns can be analyzed and optimized in real-time. Adaptive traffic signals, intelligent route planning, and dynamic

lane management have become the norm, ensuring that vehicles move seamlessly through city arteries.

- Beyond mere congestion management, TMS plays a pivotal role in enhancing road safety. Swift incident detection, aided by surveillance cameras and AI algorithms, enables prompt emergency responses. Accurate data analysis also aids in identifying accident-prone zones, paving the way for targeted safety measures and education campaigns.
- In an era dominated by environmental concerns, TMS emerges as a beacon of eco-conscious transportation. By reducing traffic snarls and optimizing routes, these systems contribute significantly to minimizing carbon emissions. Additionally, the promotion of public transport integration and smart parking solutions encourages a shift toward more sustainable commuting practices.
- TMS not only streamline traffic for commuters but also empower them with knowledge. Real-time updates about traffic conditions, alternative routes, and public transportation schedules are disseminated via mobile apps and digital displays. This empowerment translates into informed decisions, reducing frustration and fostering a sense of control among commuters.
- Crucially, TMS doesn't just tackle current issues; it plays a pivotal role in shaping future urban landscapes. The data amassed by these systems serves as a treasure trove for urban planners, offering invaluable insights. Smart city development, eco-friendly infrastructure, and responsive traffic policies all find their roots in the data provided by TMS.

- In essence, Traffic Management Systems epitomize the harmonious fusion of technology, urban planning, and public welfare. As cities continue to evolve, the implementation and enhancement of these systems will be pivotal. They are not mere solutions; they are the keystones upon which future urban mobility, safety, and sustainability will be built. By embracing these innovations, societies pave the way for cities that are not just well-connected but are also safer, greener, and more livable for all.