# This Python 3 environment comes with many helpful analytics libraries installed

# It is defined by the google colab/python docker image:

# For example, here's several helpful packages to load in

importnumpyas np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

# Input data files are available in the "../input/" directory.

# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory

from subprocess importcheck\_output

# Any results you write to the current directory are saved as output.

import matplotlib.pyplot as plt

df\_train = pd.read\_csv('/content/train\_aWnotuB.csv',parse\_dates=[0],infer\_datetime\_format=True)

df\_test = pd.read\_csv('/content/test\_BdBKkAj.csv',parse\_dates=[0],infer\_datetime\_format=True)

print("Size of training set: "+ str(df\_train.shape))

print("Size of test set: "+ str(df\_test.shape))

print('\n')

print('columns in train: '+str(df\_train.columns.tolist()))

print('columns in test: '+str(df\_test.columns.tolist()))

df\_train.head()

df\_tmp = df\_train.set\_index(['Junction','DateTime'])

level\_values = df\_tmp.index.get\_level\_values

time\_targets = df\_tmp.groupby([level\_values(0)] + [pd.Grouper(freq='1M', level=-1)])['Vehicles'].sum()

time\_targets

del df\_tmp

del time\_targets

train = df\_train.pivot(index='DateTime', columns='Junction', values='Vehicles')

train

train.isnull().sum()

train.info()

value=round(train[4].max())/1.0

value

train = train.finlla(value)

defgen\_lag\_features(df, n\_in=1,n\_out=1,dropnan=True):

n\_vars = df.shape[1]

cols, names = list(), list()

#input sequence (t-n,.....t-1)

for i inrange(n\_in,0,-1):

cols.append(df.shift(i))

names+=[('Junction %d (H-%d)' %(j+1, i)) for j inrange(n\_vars)]

# forecast sequence (t,t+1,.....t+n)

for i inrange(0,n\_out):

cols.append(df.shift(-i))

if i == 0:

names+=[('Junction %d (H)' %(j+1)) for j inrange(n\_vars)]

else:

names+=[('Junction %d (H+%d)' %(j+1,i)) for j inrange(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

Xy\_train= gen\_lag\_features(train)

Xy\_train

from sklearn.preprocessing import MinMaxScaler, StandardScaler

scaler = MinMaxScaler(feature\_range=(0,1))

Xy\_train[Xy\_train.columns]= scaler.fit\_transform(Xy\_train[Xy\_train.columns])

Xy\_train

X\_train = Xy\_train[Xy\_train.index <'2017-04-01'].iloc[:,0:4]

X\_train

y\_train= Xy\_train[Xy\_train.index <'2017-04-01'].iloc[:,4:]

y\_train

print(X\_train.shape, y\_train.shape)

X\_train = np.expand\_dims(X\_train.values,axis=2)

print(X\_train.shape)

y\_train= y\_train.values

print(y\_train.shape)

fromkeras.modelsimport Sequential

from keras.layers import Dense

from keras.layers import LSTM

from keras.initializers import he\_normal

import keras.backend as K

defroot\_mean\_squared\_error(y\_true,y\_pred):

return K.sqrt(K.mean(K.square(y\_pred - y\_true), axis=-1))

# Initialising the RNN

regressor= Sequential()

#Adding the input layer and the LSTM layer

regressor.add(LSTM(units = 50,activation='relu',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)

# Fitting the RNN to the Training set

regressor.fit(X\_train,y\_train,batch\_size=120,epochs=100,verbose=1)

X\_valid = Xy\_train[Xy\_train.index >='2017-04-01'].iloc[:,0:4]

X\_valid

X\_valid=np.expand\_dims(X\_valid.values,axis=2)

y\_pred= regressor.predict(X\_valid)

# 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

y\_truth= train[train.index >= '2017-04-01']

y\_truth

# Visualising Result for the junctions

for junction inrange(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()

from sklearn.metrics import mean\_squared\_error

from math import sqrt

defrmse(y\_true,y\_pred):

return sqrt(mean\_squared\_error(y\_true, y\_pred))

rmse(y\_truth,y\_pred)

import pandas as pd

import numpy as np

trdf = pd.read\_csv('/content/train\_aWnotuB.csv')

trainMat= trdf.to\_numpy()

tedf = pd.read\_csv('/content/test\_BdBKkAj.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)

jun1=[]

jun2=[]

jun3=[]

jun4=[]

jun5=[]

jun=[jun1,jun2,jun3,jun4,jun5]

for i inrange(0,len(train),24):

ct=0

for j inrange(24):

ct+=target[i+j]

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 inrange(k)]

for i inrange(len(jun[1])):

week[i%k].append(jun[1][i])

for i inrange(k):

print(np.mean(week[i]))

hour=[[] for i inrange(24)]

for i inrange(len(jun[0])\*24+len(jun[1])\*24, len(jun[0])\*24+len(jun[1])\*24+len(jun[2])\*24):

hour[i%24].append(target[i])

for i inrange(24):

print(np.mean(hour[i]))

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

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

import matplotlib.pyplot as plt

for i inrange(len(week)):

plt.plot(week[i],'blue')

plt.show()

import matplotlib.pyplot as plt

plt.figure(figsize=(10,10))

plt.plot(jun[0],'yellow')

plt.show()

plt.plot(jun[1],'red')

plt.show()

plt.plot(jun[2],'green')

plt.show()

plt.plot(jun[3],'blue')

plt.show()

plt.plot(jun[4],'red')

plt.show()

from sklearn.model\_selection import StratifiedKFold

skf=StratifiedKFold(n\_splits=7)

from sklearn.ensemble import RandomForestClassifier

clf=RandomForestClassifier(criterion='entropy',min\_samples\_split=100,min\_samples\_leaf=10,max\_depth=12)

import numpy as np

from sklearn.metrics import accuracy\_score

from sklearn.metrics import mean\_squared\_error

from math import sqrtclf.fit(X,y)

pred = clf.predict(X)

val1=(accuracy\_score(y,pred)\*100)

print("Accuracy Score for Random Forest :",val1\*5)

from sklearn.metrics import mean\_squared\_error

from math import sqrt

defrmse1(y\_true,y\_pred):

return sqrt(mean\_squared\_error(y\_true, y\_pred))

from sklearn.tree import DecisionTreeClassifier

DT = DecisionTreeClassifier()

DT.fit(X,y)

pred2 = DT.predict(X)

val2= (accuracy\_score(y,pred2)\*100)

print("Accuracy score for Decision tree classifer : ",val2\*5)

from sklearn.svm import SVC

SVM = SVC(kernel='linear')

SVM.fit(X,y)

pred3 = SVM.predict(X)

val3= (accuracy\_score(y,pred3)\*100)

print("Accuracy score for SVM : ",val3\*5)

from xgboost import XGBClassifier

from sklearn.metrics import accuracy\_score

XGboost = XGBClassifier(use\_label\_encoder=False, objective="multi:softmax", eval\_metric="merror", n\_estimators=20, max\_depth=10, random\_state=42)

XGboost.fit(X, y)

pred4 = XGboost.predict(X)

val4 = accuracy\_score(y, pred4) \* 100

print("Accuracy score for XGBoosting: ", val4)

importnumpyas np

import pandas as pd

importmatplotlib.pyplotasplt

#Bring some raw data

frequencies =[val1,val2]

freq\_series = pd.Series.to\_xarray(frequencies)

x\_labels = ['Rf','DT','SVM','XGBoost']

#plot the figure

plt.figure(figsize=(12,8))

ax= freq\_series.plot(kind='bar')

ax.set\_title('Evaluation of ML & Dl')

ax.set\_xlabel('Classifier')

ax.set\_ylabel('Accuracy Range')

ax.set\_xticklabels(x\_labels)

defadd\_value\_labels(ax,spacing=5):

#for each bar: Place a label

for rect in ax.patches:

y\_value= rect.get\_height()

x\_value= rect.get\_x() + rect.get\_width()/2

space=spacing

va='bottom'

if y\_value<0:

space\*=-1

va='top'

label ="{:.1f}".format(y\_value)

#create annotation

ax.annotate(label,

(x\_value,y\_value),

xytext=(0,space),

textcoords="offset points",

ha='center',

va=va)

# Call the function above. All the magic happens there.

add\_value\_labels(ax)

plt.show()