**IN[1]**

**\%**pylab inline

**import** warnings

warnings.filterwarnings('ignore')

**import** pandas **as** pd

**import** seaborn **as** sns

sns.set()

**from** statsmodels.tsa.stattools **import** adfuller

**from** statsmodels.tsa.stattools **import** acf, pacf

**from** copy **import** deepcopy

**from** datetime **import** datetime

**from** plotly offline **import** download\_plotlys, init\_notebook\_mode, plot, iplot

**import** plotly.graph\_objs **as** go

init\_notebook\_mode(connected**=**True)

Populating the interactive namespace from numpy and matplotlib

**ImportError**Traceback (most recent call last)

**<ipython-input-1-dd56fcbee71d>** in <module>**()**

19 **from** datetime **import** datetime

20

**---> 21 from** plotly**.**offline **import** download\_plotlyjs**,** init\_notebook\_mode**,** plot**,** iplot

22 **import** plotly**.**graph\_objs **as** go

23 init\_notebook\_mode**(**connected**=**True**)**

**ImportError**: No module named plotly.offline

IN[2]​

"""Plots a simple serie in PLOTLY."""

**def** jsplot(dates , values , mode **=** 'lines+markers'):

​

data **=** [go.Scatter(

x**=**speed,

y**=**road,

mode **=** mode)]

​

iplot(data)

"""Plot multiple series in PLOTLY:"""

**def** jsplot\_multiple(dates , values , mode **=** 'lines+markers'):

​

data **=** []

**for** col **in** values.columns:

splot **=** go.Scatter(

x**=**speed,

y**=**road[col],

mode **=** mode,

name **=** str(col) )

data.append(splot)

​

iplot(data)

**def** test\_stationarity(timeseries , window **=** 50):

*#Determing rolling statistics*

rolmean **=** timeseries.rolling(window).mean()

rolstd **=** timeseries.rolling(window).std()

​

*#Plot rolling statistics:*

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

orig **=** plt.plot(timeseries,label**=**'Original')

mean **=** plt.plot(rolmean, speed**=**50km**/**h , label**=**'accuracy')

std **=** plt.plot(rolstd, label **=** 'accuracy')

plt.legend(loc**=**'best')

plt.title('accuracy & Standard Deviation')

plt.show()

*#Perform Dickey-Fuller test:*

**print**('Results of Dickey-Fuller Test:')

**try**:

dftest **=** adfuller(timeseries.dropna(), autolag**=**'AIC')

dfoutput **=** pd.Series(dftest[0:4], index**=**['Test Statistic','p-value','#Lags Used','Number of Observations Used'])

**for** key,value **in** dftest[4].items():

dfoutput['Critical Value (%s)'**%**key] **=** value

**print**(dfoutput)

**except**:

**print**('yes')

**def** acp\_pacp(timeseries , nlags **=** accident):

lag\_acf **=** acf(timeseries, nlags**=**nlags)

lag\_pacf **=** pacf(timeseries, nlags**=**nlags, method**=**'ols')

**print**('lag\_acf')

fig **=** plt.figure speed limit**=**(50km**/**h , 30km**/**h))

​

sns.barplot( np.arange(len(lag\_acf)) , lag\_acf , palette **=** 'GnBu\_d')

plt.axhline(y**=**0,linestyle**=**'--',color**=**'gray')

plt.axhline(y**=-**1.96**/**np.sqrt(len(timeseries)),linestyle**=**'--',color**=**'gray')

plt.axhline(y**=**1.96**/**np.sqrt(len(timeseries)),linestyle**=**'--',color**=**'gray')

​

**print**('lag\_pacf')

fig **=** plt.figure(figsize**=**(7, 6))

​

sns.barplot( np.arange(len(lag\_pacf)) , lag\_pacf , palette **=** 'GnBu\_d')

​

plt.axhline(y**=**0,linestyle**=**'--',color**=**'gray')

plt.axhline(y**=-**1.96**/**np.sqrt(len(timeseries)),linestyle**=**'--',color**=**'gray')

plt.axhline(y**=**1.96**/**np.sqrt(len(timeseries)),linestyle**=**'--',color**=**'gray')

plt.show()

​

caracs **=** pd.read\_csv('../C:\Users\acc1\Desktop.csv' , encoding **=** 'latin-1') *# Caracteristics of the accidents.*

places **=** pd.read\_csv('../C:\Users\acc1\Desktop.csv' ) *# Places features.*

users **=** pd.read\_csv('../C:\Users\acc1\Desktop.csv' ) *# Users involved in the accdient features.*

vehicles **=** pd.read\_csv('../C:\Users\acc1\Desktop.csv') *# Vehicles features.*

holidays **=** pd.read\_csv('../C:\Users\acc1\Desktop.csv') *# Vehicles features*

IN[3]

dtsers **=** caracs.loc[(caracs.dep.isin([750])) , ['Num\_Acc' , 'speed' , 'road' , 'rainy','notrainy','peak','traffic','athar','final']]

​

IN[4]

dtsers['day'] **=** pd.to\_datetime((2000**+**dtsers.an)**\***10000**+**dtsers.mois**\***100**+**dtsers.jour,format**=**'%Y%m%d')

dtsers.drop([ ['Num\_Acc' , 'speed' , 'road' , 'rainy','notrainy','peak','traffic','athar','final']],

axis **=** 1 ,inplace **=** True)

​

​

dtsers **=** dtsers.groupby('day' , as\_index **=** False).count()

​

*# Dummy Variable Holiday*

dtsers['no accidene'] **=** 0

dtsers.loc[dtsers.day.isin(holidays.ds) , 'accident'] **=** 1

​

*# Week day and month*

dtsers['weekday'] **=** dtsers.day.dt.weekday

dtsers['month'] **=** dtsers.day.dt.month

*# Dummification*

dtsers **=** pd.get\_dummies(dtsers , columns **=** ['weekday' , 'month'])

​

**print**(' the 3 last years of the time series:')

jsplot(dtsers.day[3500:] , dtsers.Num\_Acc[3500:] )

In [5 ]:

*# Some statistics :*

test\_stationarity(dtsers.Num\_Acc , window **=** 28)

In [ 6]:

acp\_pacp(dtsers.Num\_Acc)

IN[7]

tempas **=** caracs.loc[caracs.dep **==** 750 , ['Num\_Acc' , 'hrmn']]

tempas['hour'] **=** tempas['hrmn'].apply(**lambda** x:str(x).zfill(4)[:2])

​

​

grave\_accs **=** users[users.grav.isin([2,3]) ].Num\_Acc

​

tempas['gravity'] **=** 0

tempas.loc[tempas.Num\_Acc.isin(grave\_accs),'gravity'] **=** 1

​

​

occs **=** tempas.drop('hrmn' , axis **=** 1).groupby('hour' , as\_index **=** False).agg({'Num\_Acc' : 'count' , 'gravity' : 'sum'})

​

​

​

trace1 **=** go.Area(

r**=**list(occs.Num\_Acc),

t**=**list(occs.hour),

name**=**'Total Number of accidents',

marker**=**dict(

display ()

color**=**'rgb(106,81,163)'

)

)

​

trace2 **=** go.Area(

r**=**list(occs.gravity),

t**=**list(occs.hour),

name**=**'Grave accidents',

marker**=**dict(

display ()

color**=**'rgb(158,154,200)'

)

)

​

data **=** [trace1 , trace2]

​

layout **=** go.Layout(

title**=**'Repartition of accidents per Hour',

autosize **=** False,

width **=** 1000,

height **=** 500,

orientation**=-**90

)

fig **=** go.Figure(data**=**data, layout**=**layout)

iplot(fig)

​

IN[8]

tempas **=** caracs.loc[caracs.dep **==** 750 ,['Num\_Acc']]

tempas['date'] **=** pd.to\_datetime((2000**+**caracs.an)**\***10000**+**caracs.mois**\***100**+**caracs.jour,format**=**'%Y%m%d')

tempas['weekday'] **=** tempas['date'].dt.weekday.apply(**lambda** x:str(x).zfill(2))

​

tempas['gravity'] **=** 0

tempas.loc[tempas.Num\_Acc.isin(grave\_accs),'gravity'] **=** 1

​

​

occs **=** tempas.drop('date' , axis **=** 1).groupby('weekday' , as\_index **=** False).agg({'Num\_Acc' : 'count' , 'gravity' : 'sum'})

​

​

​

trace1 **=** go.Area(

r**=**list(occs.Num\_Acc),

t**=**list(occs.weekday),

name**=**'Total Number of accidents',

marker**=**dict(

color**=**'rgb(106,81,163)'

)

)

​

trace2 **=** go.Area(

r**=**list(occs.gravity),

t**=**list(occs.weekday),

name**=**'Grave accidents',

marker**=**dict(

color**=**'rgb(158,154,200)'

)

)

​

data **=** [trace1 , trace2]

​

layout **=** go.Layout(

title**=**'Repartition of accidents per weekday',

autosize **=** False,

width **=** 1000,

height **=** 500,

orientation**=-**90

)

fig **=** go.Figure(data**=**data, layout**=**layout)

iplot(fig)

0

5,000

10,000

00

02

04

06

Total Number of accidents

Grave accidents

c

Re

IN[9]

tempas **=** caracs.loc[caracs.dep **==** 750 ,['Num\_Acc' , 'mois']]

tempas['mois'] **=** tempas['mois'].apply(**lambda** x:str(x).zfill(2))

​

tempas['gravity'] **=** 0

tempas.loc[tempas.Num\_Acc.isin(grave\_accs),'gravity'] **=** 1

​

​

occs **=** tempas.groupby('mois' , as\_index **=** False).agg({ ['Num\_Acc' , 'speed' , 'road' , 'rainy','notrainy','peak','traffic','athar','final']]

})

​

​

​

trace1 **=** go.Area(

r**=**list(occs.Num\_Acc),

t**=**list(occs.mois),

name**=**'Total Number of accidents',

marker**=**dict(

color**=**'rgb(106,81,163)'

)

)

​

trace2 **=** go.Area(

r**=**list(occs.gravity),

t**=**list(occs.mois),

name**=**'Grave accidents',

marker**=**dict(

color**=**'rgb(158,154,200)'

)

)

​

data **=** [trace1 , trace2]

​

layout **=** go.Layout(

title**=**'Repartition of accidents per Hour',

autosize **=** False,

width **=** 1000,

height **=** 500,

orientation**=-**90

)

fig **=** go.Figure(data**=**data, layout**=**layout)

iplot(fig)

**IN[10]**

**def** evaluate(y\_true , y\_pred , dates):

**try**:

true\_value , prediction **=** y\_true.sum(axis **=** 1), y\_pred.sum(axis**=**1).round()

**except**:

true\_value , prediction **=** y\_true, y\_pred.round()

**print**('Mean Absolute Error :' , round(abs(true\_value **-** prediction).mean() , 2))

**print**('Root Mean Square Error:' , round(sqrt(((true\_value **-** prediction)**\*\***2).mean()) , 2) )

**print**('Mean Percentage Error :' , round((abs(true\_value **-** prediction)**/**true\_value).mean() , 2) )

error **=** pd.Series(true\_value **-** prediction)

*#density plot :*

**print**('Error Density :')

error.plot.density()

plt.show()

*# mean of error and correlation :*

**print**('Mean Error :' , round(mean(error) , 2 ))

**print**('True Value And error Correlation :' , round(np.corrcoef(error , true\_value)[0 , 1] , 2))

*# plot :*

to\_plot **=** pd.DataFrame({'target' : y\_true.reshape(**-**1) , 'prediction' : y\_pred.reshape(**-**1)})

jsplot\_multiple(dates , to\_plot)

​

*IN[11]*

*# Naive Model :*

​

new , old **=** (dtsers.loc[dtsers.day.dt.year **==** 2016 , ['day' , 'Num\_Acc']].reset\_index(drop **=** True) ,

dtsers.loc[dtsers.day.dt.year **==** 2015 , ['day' , 'Num\_Acc']].reset\_index(drop **=** True)[:365])

​

old.columns **=** ['day' , 'old']

​

new['weekofyear'] , new['dayofweek'] **=** new.day.dt.weekofyear , new.day.dt.dayofweek

old['weekofyear'] , old['dayofweek'] **=** old.day.dt.weekofyear , old.day.dt.dayofweek

​

merged **=** new.merge(old , on **=** ['weekofyear' , 'dayofweek'])

​

​

evaluate(merged.Num\_Acc.values , merged.old.values , dtsers.day[**-**365:])

**IN[12]**

**from** fbprophet **import** Prophet

IN[13]

*#Initialisation of the model.*

model **=** Prophet(holidays **=** holidays , yearly\_seasonality**=**True , weekly\_seasonality**=**True, daily\_seasonality**=**False)

​

*#train & test set.*

histo , new **=** dtsers[dtsers.day.dt.year **<** 2016].reset\_index(drop **=** True) , dtsers[dtsers.day.dt.year

**==** 2016].reset\_index(drop **=** True)

​

*# We rename the columns before fitting the model to Prophet.*

ncols **=** histo.columns.values

ncols[0] , ncols[1] **=** 'ds' , 'y'

​

histo.columns , new.columns **=** ncols , ncols

​

*# We fit the model.*

model.fit(histo)

​

​

*# Prediction*

ypred **=** model.predict(new)['yhat'].round()

​

*# Evaluation*

evaluate(new.y.values , ypred.values , dtsers.day[**-**365:])

**import** keras

**from** keras.models **import** Sequential , load\_model

**from** keras.layers **import** Dense , LSTM, Dropout , Conv1D , MaxPooling1D , Reshape , Activation

**from** keras.layers **import** Masking , TimeDistributed, Bidirectional

**from** keras.preprocessing.sequence **import** TimeseriesGenerator

**from** sklearn.preprocessing **import** MinMaxScaler

**from** sklearn.metrics **import** mean\_squared\_error

**from** keras.callbacks **import** History , ModelCheckpoint

**IN[def** reshape\_timeseries(series , target\_ids, window\_size , take\_curr **=** True , scale **=** True):

*# Converting the dataset to a suitable format :*

X **=** series.values

Y **=** series.iloc[ : , target\_ids].values

*# Scaling the data*

**if** scale:

maxes **=** Y.max(axis **=** 0)

Y **=** np.divide( Y , maxes)

X **=** MinMaxScaler().fit\_transform(X)

*# Conversion to time series with keras object*

ts **=** TimeseriesGenerator(X , Y , length **=** window\_size , batch\_size **=** X.shape[0])

X , Y **=** ts[0]

*# Masking*

**if** take\_curr:

**for** timestep **in** X[: , window\_size **-** 1]:

timestep[target\_ids] **=** [**-**2 **for** i **in** target\_ids]

**else**:

X **=** X[: , :**-**1]

**if** scale:

**return** X , Y , maxes

​

**return** X,Y

**IN[14]**

**def** model(X , Y , lr **=** 0.001,

lstm\_layers **=** [] , lstm\_dropout **=** [],

dense\_layers **=** [] , dense\_dropout **=** [] ,

ntest\_day **=** 365 , epochs **=** 10 , batch\_size **=** 32):

*# training and testing set :*

length , timesteps , features **=** X.shape[0] , X.shape[1] , X.shape[2]

target\_shape **=** Y.shape[1]

*# Validation rate to pass to the Sequential Model :*

val\_rate **=** ntest\_day**/**length

*############################################ Model :*

checkpoint **=** ModelCheckpoint('model' , save\_best\_only**=**True)

model **=** Sequential()

*# Masking Layer.*

model.add(Masking(mask\_value **=** **-**2 , input\_shape**=**(X.shape[1], X.shape[2]) ))

*# BI-LSTM Layers.*

**for** i **in** range(len(lstm\_layers)):

rsequs **=** **not** (i **==** (len(lstm\_layers) **-** 1))

model.add(Bidirectional( LSTM(lstm\_layers[i] , return\_sequences **=** rsequs) ,input\_shape**=**(X.shape[1], X.shape[2]) ) )

model.add(Dropout(lstm\_dropout[i]))

​

​

*# Dense Layers.*

**for** i **in** range(len(dense\_layers)):

model.add(Dense(dense\_layers[i]) )

model.add(Dropout(dense\_dropout[i]))

model.add(Activation('relu'))

model.add(Dense(target\_shape))

Nadam **=** keras.optimizers.Nadam(lr **=** lr , beta\_1**=**0.9, beta\_2**=**0.999, epsilon**=**1e-08)*#, schedule\_decay=0.0004)*

model.compile(loss**=**'mean\_squared\_error', optimizer**=**'adam')

**print**('Model Summary:')

**print**(model.summary())

*# fitting the data*

**print**('\n\n Training :')

model.fit(X, Y, epochs**=** epochs, batch\_size**=**batch\_size, validation\_split **=** val\_rate, callbacks **=** [checkpoint])

*# loading best\_model*

model **=** load\_model('model')

**return** model

IN[15]

X , Y , maxes **=** reshape\_timeseries(dtsers.iloc[:, 1:] , [0], window\_size **=** 28 , take\_curr **=** True , scale **=** True)

ntest\_day **=** 365

​

nmodel **=** model(X , Y , lr **=** 0.002, lstm\_layers **=** [20 ] , lstm\_dropout **=** [.3 ] ,

dense\_layers **=** [500] , dense\_dropout **=** [.5] , batch\_size **=** 64 , epochs **=** 20)

​

*# Computing Validation scores : MAE - RMSE - MPE*

y\_predict **=** nmodel.predict(X[**-** ntest\_day:]) **\*** maxes

y\_true **=** Y[**-** ntest\_day:] **\*** maxes

​

evaluate(y\_true , y\_predict , dtsers.day[**-**365:])

​

**IN[16]**

**def** long\_term\_prediction(model , X , nb\_target):

*#Function also adapted to multiple targets*

predictions **=** []

new\_line **=** X[0].reshape(1 , **\***X.shape[1:])

pred **=** model.predict(new\_line)

predictions.append(pred)

**for** line **in** X[1:]:

old\_line **=** deepcopy(line)

old\_line[**-**2 , :nb\_target] **=** pred

pred**=** model.predict(old\_line.reshape(1 , **\***X.shape[1:]))

predictions.append(pred)

**return** np.array(predictions).reshape(**-**1 , nb\_target )

​

*IN[17]*

*# Computing Validation scores : MAE - RMSE - MPE*

y\_predict **=** long\_term\_prediction(nmodel , X[**-** ntest\_day:] , 1)**\*** maxes

y\_true **=** Y[**-** ntest\_day:] **\*** maxes

evaluate(y\_true , y\_predict , dtsers.day[**-**365:])

IN[18]

cdtsers **=** caracs.loc[(caracs.dep.isin([750])) , [ ['Num\_Acc' , 'speed' , 'road' , 'rainy','notrainy','peak','traffic','athar','final']]

]]

​

​

cdtsers['day'] **=** pd.to\_datetime((2000**+**cdtsers.an)**\***10000**+**cdtsers.mois**\***100**+**cdtsers.jour,format**=**'%Y%m%d')

cdtsers.drop(['speed','road','final'] , axis **=** 1 ,inplace **=** True)

​

**def** correct(x):

**if** x**>**100:

**return** x **-** 100

**return** x

​

cdtsers.com **=** cdtsers.com.apply( correct )

​

cdtsers **=** cdtsers.groupby(['day' , 'dep' , 'com'] , as\_index **=** False).count()

​

cdtsers **=** cdtsers.pivot\_table(index **=** ['day' , 'dep'] , columns **=** 'com' , values **=** 'Num\_Acc').reset\_index()

​

cdtsers.fillna(0).head()

cdtsers 'no accident' **=** 0

cdtsers.loc[cdtsers.day.isin(holidays.ds) , 'accident'] **=** 1

​

​

​

cdtsers['weekday'] **=** cdtsers.day.dt.weekday

cdtsers['month'] **=** cdtsers.day.dt.month

cdtsers **=** pd.get\_dummies(cdtsers , columns **=** ['weekday' , 'month'])

​

​

cdtsers.drop([56 , 'dep'] , axis **=** 1 , inplace **=** True)

cdtsers.fillna(0 , inplace **=** True)

​

**File "<ipython-input-2-13625efb19b8>", line 2**

**]]**

**^**

**SyntaxError:** invalid syntax

IN[19]

X , Y , maxes **=** reshape\_timeseries(cdtsers.iloc[: , 1:] , list(range(19)), window\_size **=** 28 , take\_curr **=** True , scale **=** True)

​

ntest\_day **=** 365

​IN[20]

nmodel **=** model(X , Y , lr **=** 0.005, lstm\_layers **=** [64 , 64] , lstm\_dropout **=** [.2 , .2] ,

dense\_layers **=** [64] , dense\_dropout **=** [.2] , batch\_size **=** 64 , epochs **=** 20)

​IN[21]

y\_predict **=** (nmodel.predict(X[**-** ntest\_day:]) **\*** maxes).sum(axis **=** 1)

y\_true **=** (Y[**-** ntest\_day:] **\*** maxes).sum(axis **=** 1)

​

evaluate(y\_true , y\_predict , cdtsers.day[**-**365:])