import pandas as pd  
import numpy as np  
from fbprophet import Prophet  
import matplotlib.pyplot as plt  
from statsmodels.tools.eval\_measures import rmse  
from fbprophet.diagnostics import cross\_validation,performance\_metrics  
from fbprophet.plot import plot\_cross\_validation\_metric

# Read in the given data and set the index column to the date column and parse the dates  
df=pd.read\_csv("../../02\_data\_acquisition\_understanding/01\_data\_source/sickness\_table.csv", index\_col="date", parse\_dates=True )   
df["drivers\_atwork"]=df["n\_duty"]+df["sby\_need"]  
df.index.freq="D" #Set the frequence to Daily  
df.index

DatetimeIndex(['2016-04-01', '2016-04-02', '2016-04-03', '2016-04-04',  
 '2016-04-05', '2016-04-06', '2016-04-07', '2016-04-08',  
 '2016-04-09', '2016-04-10',  
 ...  
 '2019-05-18', '2019-05-19', '2019-05-20', '2019-05-21',  
 '2019-05-22', '2019-05-23', '2019-05-24', '2019-05-25',  
 '2019-05-26', '2019-05-27'],  
 dtype='datetime64[ns]', name='date', length=1152, freq='D')

# Prepare the Data for Prophet  
# Create the column "date"  
df["date"]=df.index  
# Create new DataFrames with the column "date"  
df\_new=pd.DataFrame(df["date"])  
# Create the column y="calls"  
df\_new["y"]=df["calls"]  
# Delete the index  
df\_new.reset\_index(drop=True, inplace=True)  
# Rename the columns to the Prophet specifics  
df\_new.rename(columns={"date": "ds", "y": "y"}, inplace=True)  
# Show the head of the DataFrame  
df\_new.head()

ds

y

0

2016-04-01

8154.0

1

2016-04-02

8526.0

2

2016-04-03

8088.0

3

2016-04-04

7044.0

4

2016-04-05

7236.0

# Split the Data in a test and a split data set  
train = df\_new.iloc[:len(df\_new)-365]  
test = df\_new.iloc[len(df\_new)-365:]  
test.set\_index(test["ds"], inplace=True)  
# Get the time window for the predicted times   
df\_new.iloc[len(df\_new)-365:]

ds

y

787

2018-05-28

8862.0

788

2018-05-29

8226.0

789

2018-05-30

8064.0

790

2018-05-31

7392.0

791

2018-06-01

10752.0

...

...

...

1147

2019-05-23

8544.0

1148

2019-05-24

8814.0

1149

2019-05-25

9846.0

1150

2019-05-26

9882.0

1151

2019-05-27

8790.0

365 rows × 2 columns

test

ds

y

ds

2018-05-28

2018-05-28

8862.0

2018-05-29

2018-05-29

8226.0

2018-05-30

2018-05-30

8064.0

2018-05-31

2018-05-31

7392.0

2018-06-01

2018-06-01

10752.0

...

...

...

2019-05-23

2019-05-23

8544.0

2019-05-24

2019-05-24

8814.0

2019-05-25

2019-05-25

9846.0

2019-05-26

2019-05-26

9882.0

2019-05-27

2019-05-27

8790.0

365 rows × 2 columns

#Create the Prohpet-Model,with an interval\_width of 97,5% (97,5% of the train values are in the predicted range)   
m = Prophet(interval\_width=0.975, daily\_seasonality=True,seasonality\_mode="multiplicative")  
# Train the Prophet-Model  
m.fit(train)  
# Create future datafame with xx days (periods & freq) as basis for prediction  
future = m.make\_future\_dataframe(periods=365,freq='D')  
# Make predictions for the created future dataframe  
forecast = m.predict(future)  
#plot the predictions   
m.plot(forecast, figsize=(20, 6));  
forecast.set\_index(forecast["ds"], inplace=True)  
forecast["n\_RealCalls"]=df["calls"]

png

png

fig = plt.figure(figsize=(50,12))  
ax = fig.add\_subplot(1, 1, 1)  
ax.plot(forecast["ds"].iloc[len(forecast)-365:], forecast["yhat"].iloc[len(forecast)-365:], label='Medium prediction calls', c="blue")  
ax.plot(forecast["ds"].iloc[len(forecast)-365:], forecast["yhat\_upper"].iloc[len(forecast)-365:], label='Saftey prediction of calls', c="red")  
ax.plot(forecast["ds"].iloc[len(forecast)-365:],forecast["n\_RealCalls"].iloc[len(forecast)-365:],label="Real amount of Calls", c="black")  
ax.legend(loc='upper left')

<matplotlib.legend.Legend at 0x24e571128c8>

png

png

#ratio calls/ drivers:   
#adjust the dataframe by the values where the base line is hit  
df\_adjusted = df.loc[df["drivers\_atwork"] != 1770.0]  
df\_adjusted = df\_adjusted.loc[df\_adjusted["drivers\_atwork"] != 1700.0]  
df\_adjusted = df\_adjusted.loc[df\_adjusted["drivers\_atwork"] != 1800.0]  
df\_adjusted = df\_adjusted.loc[df\_adjusted["drivers\_atwork"] != 1900.0]  
df\_adjusted.head()  
  
Min\_MSEError=10\*\*1000  
x\_Value=0  
for x in np.linspace(2, 6, num=1000):  
 ErrorSUM=0  
 #print(x)  
 for counter1 in range(0,len(df\_adjusted)):  
 a=float(df\_adjusted.iloc[counter1, 7]) # drivers\_atwork  
 b=float(df\_adjusted.iloc[counter1, 2]) # calls  
 ErrorSUM=ErrorSUM + (a-(b/x))\*\*2  
 if Min\_MSEError > (ErrorSUM/len(df\_adjusted)):  
 Min\_MSEError=ErrorSUM/len(df\_adjusted)  
 x\_Value=x  
   
print ("Min\_MSEError", Min\_MSEError)  
print("Best linear\_coef value: ", x\_Value)  
  
  
fig = plt.figure(figsize=(30,20))  
ax = fig.add\_subplot(111)  
ax.set\_title('calls vs drivers\_atwork')  
plt.plot(df["drivers\_atwork"],color='black',label='drivers\_atwork', lw=4)  
plt.plot(df["calls"]/x\_Value,color='orange',label="calls divided py linear\_coef", alpha=0.7, lw=3)  
ax.set\_xlabel('Date')  
ax.set\_ylabel('Drivers; calls/linear\_coef')  
ax.legend(loc='upper left')  
plt.show()

Min\_MSEError 196.22728826407905  
Best linear\_coef value: 4.822822822822823

png

png

#List of interval\_width to be evaluated  
liste=[0.7,0.75,0.80, 0.85, 0.9,0.95,0.975, 0.99]  
  
data=[]  
for i in liste:  
 #Create the Prohpet-Model,with an interval\_width of 95% (95% of the train values are in the predicted range)   
 m = Prophet(interval\_width=i, daily\_seasonality=True,seasonality\_mode="multiplicative")  
 # Train the Prophet-Model  
 m.fit(train)  
 # Create future datafame with xx days (periods & freq) as basis for prediction  
 future = m.make\_future\_dataframe(periods=365,freq='D')  
 # Make predictions for the created future dataframe  
 forecast = m.predict(future)  
 #plot the predictions   
 #m.plot(forecast, figsize=(20, 6));  
 forecast.set\_index(forecast["ds"], inplace=True)  
 forecast["n\_RealCalls"]=df["calls"]  
  
  
 df\_eval=pd.DataFrame(forecast["ds"].iloc[len(forecast)-365:])  
 df\_eval["pred\_n\_duty"]=forecast["yhat"].iloc[len(forecast)-365:]/x\_Value  
 df\_eval["pred\_n\_sby"]=forecast["yhat\_upper"].iloc[len(forecast)-365:]/x\_Value-forecast["yhat"].iloc[len(forecast)-365:]/x\_Value  
 df\_eval["real\_drivers\_atwork"]=df["drivers\_atwork"].iloc[len(forecast)-365:]  
 df\_eval["Delta"]=df\_eval["pred\_n\_duty"]-df\_eval["real\_drivers\_atwork"]  
 df\_eval["Delta2"]=df\_eval["pred\_n\_duty"] + df\_eval["pred\_n\_sby"] -df\_eval["real\_drivers\_atwork"]  
  
 df\_eval["pred\_sbyNeed"]= df\_eval["Delta"].apply(lambda x: -x if x < 0 else 0)  
 df\_eval["pred\_dafted"]= df\_eval["Delta2"].apply(lambda x: -x if x < 0 else np.nan)  
 df\_eval["pred\_utilizationSby"]=df\_eval["pred\_sbyNeed"]/df\_eval["pred\_n\_sby"]  
 df\_eval["pred\_utilizationSby"]= df\_eval["pred\_utilizationSby"].apply(lambda x: x if x < 1 else 1)  
  
 df\_eval["real\_n\_sby"]=df["n\_sby"].iloc[len(forecast)-365:]  
 df\_eval["real\_sby\_need"]=df["sby\_need"].iloc[len(forecast)-365:]  
 df\_eval["real\_dafted"]=df["dafted"].iloc[len(forecast)-365:]  
 utilization=df\_eval["pred\_utilizationSby"].mean()  
 Overexceed=df\_eval["pred\_dafted"].count()/365   
 FTE\_shifte\_needed=df\_eval["pred\_n\_duty"].sum() + df\_eval["pred\_sbyNeed"].sum()  
 FTE\_standbyShifts\_withoutDuty\_needed=df\_eval["pred\_n\_sby"].sum() - df\_eval["pred\_sbyNeed"].sum() + df\_eval["pred\_dafted"].iloc[len(df)-365:].sum()  
   
   
   
 data.append([i, utilization, Overexceed,FTE\_shifte\_needed,FTE\_standbyShifts\_withoutDuty\_needed])  
  
   
 #print(f'Schritt-{i}:In the last 365 days {df\_eval["pred\_n\_duty"].sum() + df\_eval["pred\_sbyNeed"].sum()} FTE shifts we needed')  
 #print(f'Schritt-{i}:In the last 365 days {df\_eval["pred\_n\_sby"].sum() - df\_eval["pred\_sbyNeed"].sum() + df\_eval["pred\_dafted"].iloc[len(df)-365:].sum()} FTE shifts were hold on standby and were not needed')  
 #print(f"Schritt-{i}: The utilization of the standby drivers is: {utilization}, and the number of standby drivers is exeeded in: {Overexceed} 5 over one year")  
  
cols=['interval\_width', 'Utilization\_ofStandby\_%', 'Standby\_exceeded\_%', 'FTE\_shifte\_needed', 'FTE\_standbyShifts\_withoutDuty\_needed']  
df\_result= pd.DataFrame(data, columns=cols)  
df\_result.set\_index("interval\_width", inplace=True)  
df\_result

Utilization\_ofStandby\_%

Standby\_exceeded\_%

FTE\_shifte\_needed

FTE\_standbyShifts\_withoutDuty\_needed

interval\_width

0.700

0.681972

0.454795

714844.426749

2016.720380

0.750

0.659824

0.432877

714844.426749

10340.131240

0.800

0.630474

0.358904

714844.426749

19844.493298

0.850

0.599755

0.306849

714844.426749

30698.786910

0.900

0.555172

0.230137

714844.426749

45843.909901

0.950

0.495772

0.117808

714844.426749

67919.806416

0.975

0.442822

0.065753

714844.426749

88130.147749

0.990

0.393779

0.030137

714844.426749

111117.289235

#df\_eval.to\_csv("PredictionResults.csv", sep=";", decimal=",")  
#df\_eval

#df\_result["Utilization\_ofStandby\_%"].plot(figsize=(12,8), grid =True)  
#df\_result["Standby\_exceeded\_%"].plot()

#fig, ax1 = plt.subplots()  
#ax2 = ax1.twinx()  
#ax1.plot(df\_result.index,df\_result["Utilization\_ofStandby\_%"], 'g-')  
#ax1.plot( df\_result.index,df\_result["Standby\_exceeded\_%"], 'g-')  
#ax2.plot( df\_result.index,df\_result["FTE\_standbyShifts\_withoutDuty\_needed"], 'r-')  
#  
#ax1.set\_xlabel('X data')  
#ax1.set\_ylabel('Y1 data', color='g')  
#ax2.set\_ylabel('Y2 data', color='b')  
#plt.show()

fig, ax1 = plt.subplots()  
ax1.plot(df\_result.index,df\_result["Utilization\_ofStandby\_%"], 'g-', label="Sby\_utilization")  
ax1.plot( df\_result.index,df\_result["Standby\_exceeded\_%"], 'b-', label="Sby\_exceeded")  
  
ax1.set\_xlabel('interval\_width')  
ax1.set\_ylabel('Percentage', color='black')  
plt.legend()  
plt.show()

png

png

#Create the Prohpet-Model,with an interval\_width of 97,5% (97,5% of the train values are in the predicted range)   
m = Prophet(interval\_width=0.975, daily\_seasonality=True,seasonality\_mode="multiplicative")  
# Train the Prophet-Model  
m.fit(train)  
# Create future datafame with xx days (periods & freq) as basis for prediction  
future = m.make\_future\_dataframe(periods=365,freq='D')  
# Make predictions for the created future dataframe  
forecast = m.predict(future)  
#plot the predictions   
forecast.set\_index(forecast["ds"], inplace=True)  
forecast["n\_RealCalls"]=df["calls"]  
forecast["drivers\_atwork\_old"]=df[["drivers\_atwork"]]  
  
fig = plt.figure(figsize=(50,12))  
ax = fig.add\_subplot(1, 1, 1)  
ax.plot(forecast["ds"], forecast["yhat"]/x\_Value, label='Medium prediction drivers[duty]', c="blue")  
ax.plot(forecast["ds"], forecast["yhat\_upper"]/x\_Value, label='Saftey prediction of drivers [duty + standby]', c="red")  
ax.plot(forecast["ds"],forecast["drivers\_atwork\_old"],label="Numbers of drivers at work in old model", c="black")  
ax.legend(loc='upper left')

<matplotlib.legend.Legend at 0x24e56f6d7c8>

png

png

#### Calculation of the root mean square error

pred=forecast["yhat"].iloc[len(forecast)-365:].values  
try:  
 rmse\_value=rmse(pred,test['y'].values)  
except:   
 rmse\_value=0  
print(f' The root mean square error is: {rmse\_value}')

The root mean square error is: 1012.0483883515906

#### Prophets Diagnostic Crossvalidation

# Initial 1 years training period  
initial = 1 \* 365.25  
initial = str(initial) + ' days'  
# Fold every year  
period = 1 \* 365  
period = str(period) + ' days'  
# Forecast 1 year into the future  
horizon = 365  
horizon = str(horizon) + ' days'  
  
df\_cv = cross\_validation(m, initial=initial, period=period, horizon = horizon)  
plot\_cross\_validation\_metric(df\_cv, metric='mape');

INFO:fbprophet:Making 1 forecasts with cutoffs between 2017-05-27 00:00:00 and 2017-05-27 00:00:00  
  
  
  
 0%| | 0/1 [00:00<?, ?it/s]  
  
  
C:\Users\niels\anaconda3\envs\tsa\_course\_env\_V3\lib\site-packages\fbprophet\plot.py:526: FutureWarning: casting timedelta64[ns] values to int64 with .astype(...) is deprecated and will raise in a future version. Use .view(...) instead.  
 x\_plt = df\_none['horizon'].astype('timedelta64[ns]').astype(np.int64) / float(dt\_conversions[i])  
C:\Users\niels\anaconda3\envs\tsa\_course\_env\_V3\lib\site-packages\fbprophet\plot.py:527: FutureWarning: casting timedelta64[ns] values to int64 with .astype(...) is deprecated and will raise in a future version. Use .view(...) instead.  
 x\_plt\_h = df\_h['horizon'].astype('timedelta64[ns]').astype(np.int64) / float(dt\_conversions[i])

png

png