# Agenda

1. General Data Properties
2. Access the quality of the data
   1. Checking incorrect value types
   2. Missing values and empty data & incorrect or invalid values
   3. Outliers and non relevant data
3. Data Wrangling & Analysis

## 1. General Data Properties

import pandas as pd  
import numpy as np   
import matplotlib.pyplot as plt  
  
  
from statsmodels.tsa.seasonal import seasonal\_decompose  
from statsmodels.graphics.tsaplots import month\_plot, quarter\_plot  
from statsmodels.tsa.stattools import adfuller  
  
%matplotlib inline

# Read in the given data and set the index column to the date column and parse the dates  
df=pd.read\_csv("../01\_data\_source/sickness\_table.csv", index\_col="date", parse\_dates=True )   
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')

df.head()

Unnamed: 0

n\_sick

calls

n\_duty

n\_sby

sby\_need

dafted

date

2016-04-01

0

73

8154.0

1700

90

4.0

0.0

2016-04-02

1

64

8526.0

1700

90

70.0

0.0

2016-04-03

2

68

8088.0

1700

90

0.0

0.0

2016-04-04

3

71

7044.0

1700

90

0.0

0.0

2016-04-05

4

63

7236.0

1700

90

0.0

0.0

df.describe()

Unnamed: 0

n\_sick

calls

n\_duty

n\_sby

sby\_need

dafted

count

1152.000000

1152.000000

1152.000000

1152.000000

1152.0

1152.000000

1152.000000

mean

575.500000

68.808160

7919.531250

1820.572917

90.0

34.718750

16.335938

std

332.698061

14.293942

1290.063571

80.086953

0.0

79.694251

53.394089

min

0.000000

36.000000

4074.000000

1700.000000

90.0

0.000000

0.000000

25%

287.750000

58.000000

6978.000000

1800.000000

90.0

0.000000

0.000000

50%

575.500000

68.000000

7932.000000

1800.000000

90.0

0.000000

0.000000

75%

863.250000

78.000000

8827.500000

1900.000000

90.0

12.250000

0.000000

max

1151.000000

119.000000

11850.000000

1900.000000

90.0

555.000000

465.000000

df.info()

<class 'pandas.core.frame.DataFrame'>  
DatetimeIndex: 1152 entries, 2016-04-01 to 2019-05-27  
Freq: D  
Data columns (total 7 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Unnamed: 0 1152 non-null int64   
 1 n\_sick 1152 non-null int64   
 2 calls 1152 non-null float64  
 3 n\_duty 1152 non-null int64   
 4 n\_sby 1152 non-null int64   
 5 sby\_need 1152 non-null float64  
 6 dafted 1152 non-null float64  
dtypes: float64(3), int64(4)  
memory usage: 72.0 KB

### 1.1 Columns descriptions

* date: entry date
* n\_sick: number of drivers called sick on duty
* calls: number of emergency call
* n\_duty: number of drivers on duty available
* n\_sby: number of standby resources available
* sby\_need: number of standbys, which are activated on a given day
* dafted: number of additional drivers needed due to not enough standbys

### 1.2 Additional informations

* Business claims, that having a daily fixed amount of standbys (n\_sby = 90) is not efficient because there are days with too many standbys followed by days with not enough standbys. The business aims at a more dynamical standby allocation, which takes seasonal patterns into account.
* Most important, the model should minimize dates with not enough standby drivers at hand!

## 2. Access the quality of the data

* Missing values and empty data
* Data imputation
* Incorrect types
* incorrect or invalid values
* outliers and non relevant data
* statistical sanitization

### A: Checking incorrect value types

# What kind of variable types are given in the data frame  
df.dtypes

Unnamed: 0 int64  
n\_sick int64  
calls float64  
n\_duty int64  
n\_sby int64  
sby\_need float64  
dafted float64  
dtype: object

It seems reaonable, that the number of sick drivers (n\_sick), the drivers on duty (n\_duty), number of standby resources available (n\_sby) are integer values since there are no half drivers. The columns calls, standby drivers needed (sby\_need) as well as the column dafter (number of additional drivers needed due to not enough standbys) are float values. In this use case no float values are needed. The float values will be transformed to int64:

df["calls"] = df["calls"].apply(np.int64)  
df["sby\_need"] = df["sby\_need"].apply(np.int64)  
df["dafted"] = df["dafted"].apply(np.int64)

df.dtypes

Unnamed: 0 int64  
n\_sick int64  
calls int64  
n\_duty int64  
n\_sby int64  
sby\_need int64  
dafted int64  
dtype: object

# How much data lines includes the dataframe  
print( f"The dataframe contains: {len(df)} datasets. One dataset is one day resulting in data for roundabout {len(df)/365} years")

The dataframe contains: 1152 datasets. One dataset is one day resulting in data for roundabout 3.1561643835616437 years

### B: Missing values and empty data & incorrect or invalid values

# Are values values missing:  
df.isnull().sum()

Unnamed: 0 0  
n\_sick 0  
calls 0  
n\_duty 0  
n\_sby 0  
sby\_need 0  
dafted 0  
dtype: int64

As displayed there are no missing values. And we have seen above all columns contain only integer value thus the values are valid. Thus no data imputation is needed

### C: Outliers and non relevant data

fig, (fig1, fig2, fig3, fig4, fig5, fig6) = plt.subplots(1, 6)  
fig.suptitle('Outliers analysis')  
fig.set\_figheight(8)  
fig.set\_figwidth(21)  
fig1.boxplot(df["calls" ])  
fig1.set\_title('calls')  
fig2.boxplot(df["n\_sick"])  
fig2.set\_title('n\_sick')  
fig3.boxplot(df["n\_duty"])  
fig3.set\_title('n\_duty')  
fig4.boxplot(df["n\_sby"])  
fig4.set\_title('n\_sby')  
fig5.boxplot(df["sby\_need"])  
fig5.set\_title('sby\_need')  
fig6.boxplot(df["dafted"])  
fig6.set\_title('dafted');

png

png

df["sby\_need"].plot.hist(bins=50,edgecolor="k", title="Histogram of standby drivers needed").autoscale(enable=True,axis="both", tight=True)

png

png

df["dafted"].plot.hist(bins=50,edgecolor="k", title="Histogram of further drivers needed [exceeding duty and standby]").autoscale(enable=True,axis="both", tight=True)

png

png

Column "calls": - The columns calls has two small outliers around 11800.This seems to me reasonable

Column "n\_sick": - The column n\_sick has 4 outliers around 110-120. This seems to me reasonable as well

Column "n\_duty": - no major outliers can be detected

column "n\_sby": - its a fixed number --> no outliers

columns sby\_need & dafted: - The column sby\_need describes the number of standbys activated on a given day. The column "dafted" describes the number of additional drivers needed due to not enough standbys. These values are equal zero when enough drivers are on duty and no additional drivers are needed. This is the case for the most times (see Histogram). In some cases the number of drivers on duty is not sufficient and thus sby\_needed is greater then 0 and if this number exeeds 90 the number duafted is greater then 0 as well. - The boxplots diagrams show a lot of outliers here. This is reasonable since just in a few cases standby drivers are needed as well as dafted drivers.

###### All together no unreasonable outliers are detected in the dataset

## 3. Data Wrangling & Analysis

### Data Wrangling

* Reshaping and transforming structures
* Indexing data for quick access (already done in chapter one)
* Merging, combining and joining data

### Analysis

* Exploration
* Visualization and representation
* Correlation vs Causation analysis
* Statistical analysis

### A: Merging, combining and joining data

* the column of the number of drivers working at the given date is not calculated yet
* drivers\_atwork= n\_duty + sby\_need

df["drivers\_atwork"]=df["n\_duty"]+df["sby\_need"]

### B: Visaulization and representation

#### Basic visualization

fig, ((fig1, fig2, fig3), (fig4, fig5, fig6))= plt.subplots(2, 3)  
fig.suptitle('Graphical representation of the given data')  
fig.set\_figheight(12)  
fig.set\_figwidth(27)  
fig1.plot(df["calls" ], "tab:orange")  
fig1.set\_title('calls')  
fig2.plot(df["n\_sick"], "tab:red")  
fig2.set\_title('n\_sick')  
fig3.plot(df["n\_duty"], "tab:green")  
fig3.set\_title('n\_duty')  
fig4.plot(df["drivers\_atwork"], "tab:grey")  
fig4.set\_title('drivers\_atwork')  
fig5.plot(df["sby\_need"], "tab:pink")  
fig5.set\_title('sby\_need')  
fig6.plot(df["dafted"], "tab:blue")  
fig6.set\_title('dafted');  
  
#n\_sby is not considered here since its a fixed value and thus boring

png

png

Column "calls": - the calls seems to have a trend (more calls with increasing times) as well as a seasonal (seasonal fluctuation) component - there is definitely some noice in the data - Analysis of trend and sesonality is needed!

Column "n\_sick": - the calls seems to have a weak trend (more calls with increasing times) as well as a weak seasonal (seasonal fluctuation) component --> closer look is necessary - Analysis of trend and sesonality is needed!

Column "n\_duty": - the number of planned duty drivers is jump-fixed

column "drivers\_atwork": - the buttom line is the number of drivers on duty - besides the base line, there are days where more drivers needs to be at work - the pattern of days where more drivers has to be at work seems to be seasonal --> closer look is needed - it seems that the amount of drivers at work is correlating with the number of calls --> closer look is needed

columns sby\_need & dafted: - base line is 0 - sby\_need and dafted are connected --> max(sby\_need-n\_sby;0) - The peaks of the data seems to be seasonal

#### Analyzing Trends & Seasonalities

Calls, n\_sick & drivers\_atwork seems to have a seasonal and a trend component. Further evaluation are conducted in the following.

def test\_adfuller(series,title=''):  
 """  
 A Augmented Dickey-Fuller Test-Report is created based on the given time series and an optional title  
 """  
 print(f'DF-Test: {title}')  
 result = adfuller(series.dropna(),autolag='AIC')  
 labels = ['ADF test statistic','p-value','# lags used','# observations']  
 out = pd.Series(result[0:4],index=labels)  
  
 for key,val in result[4].items():  
 out[f'critical value ({key})']=val  
 print(out.to\_string())   
   
 if result[1] <= 0.05:  
 print("Strong evidence against the null hypothesis")  
 print("Reject of the null hypothesis")  
 print("Data is stationary")  
 else:  
 print("Weak evidence against the null hypothesis")  
 print("Fail to reject the null hypothesis")  
 print("Data is non-stationary")

#calls  
results=seasonal\_decompose(df["calls"], model="add")  
results.plot();

png

png

results.seasonal.head(59)

date  
2016-04-01 -34.035301  
2016-04-02 -212.149504  
2016-04-03 -637.308136  
2016-04-04 312.846232  
2016-04-05 299.685953  
2016-04-06 201.328810  
2016-04-07 69.631946  
2016-04-08 -34.035301  
2016-04-09 -212.149504  
2016-04-10 -637.308136  
2016-04-11 312.846232  
2016-04-12 299.685953  
2016-04-13 201.328810  
2016-04-14 69.631946  
2016-04-15 -34.035301  
2016-04-16 -212.149504  
2016-04-17 -637.308136  
2016-04-18 312.846232  
2016-04-19 299.685953  
2016-04-20 201.328810  
2016-04-21 69.631946  
2016-04-22 -34.035301  
2016-04-23 -212.149504  
2016-04-24 -637.308136  
2016-04-25 312.846232  
2016-04-26 299.685953  
2016-04-27 201.328810  
2016-04-28 69.631946  
2016-04-29 -34.035301  
2016-04-30 -212.149504  
2016-05-01 -637.308136  
2016-05-02 312.846232  
2016-05-03 299.685953  
2016-05-04 201.328810  
2016-05-05 69.631946  
2016-05-06 -34.035301  
2016-05-07 -212.149504  
2016-05-08 -637.308136  
2016-05-09 312.846232  
2016-05-10 299.685953  
2016-05-11 201.328810  
2016-05-12 69.631946  
2016-05-13 -34.035301  
2016-05-14 -212.149504  
2016-05-15 -637.308136  
2016-05-16 312.846232  
2016-05-17 299.685953  
2016-05-18 201.328810  
2016-05-19 69.631946  
2016-05-20 -34.035301  
2016-05-21 -212.149504  
2016-05-22 -637.308136  
2016-05-23 312.846232  
2016-05-24 299.685953  
2016-05-25 201.328810  
2016-05-26 69.631946  
2016-05-27 -34.035301  
2016-05-28 -212.149504  
2016-05-29 -637.308136  
Freq: D, Name: seasonal, dtype: float64

results.seasonal.plot(figsize=(50,8));

png

png

test\_adfuller(df["calls"], "ADF Test for calls on a daily basis")

DF-Test: ADF Test for calls on a daily basis  
ADF test statistic -2.761936  
p-value 0.063920  
# lags used 19.000000  
# observations 1132.000000  
critical value (1%) -3.436140  
critical value (5%) -2.864097  
critical value (10%) -2.568131  
Weak evidence against the null hypothesis  
Fail to reject the null hypothesis  
Data is non-stationary

calls\_m =df["calls"].resample(rule="MS").mean()

results=seasonal\_decompose(calls\_m, model="add")  
results.plot();

png

png

month\_plot(calls\_m);

png

png

test\_adfuller(calls\_m, "ADF Test for calls on a month basis")

DF-Test: ADF Test for calls on a month basis  
ADF test statistic 0.399761  
p-value 0.981486  
# lags used 10.000000  
# observations 27.000000  
critical value (1%) -3.699608  
critical value (5%) -2.976430  
critical value (10%) -2.627601  
Weak evidence against the null hypothesis  
Fail to reject the null hypothesis  
Data is non-stationary

calls\_w =df["calls"].resample(rule="W").mean()  
results=seasonal\_decompose(calls\_w, model="add")  
results.plot();

png

png

test\_adfuller(calls\_w, "ADF Test for calls on a weekly basis")

DF-Test: ADF Test for calls on a weekly basis  
ADF test statistic -2.811003  
p-value 0.056719  
# lags used 10.000000  
# observations 155.000000  
critical value (1%) -3.473259  
critical value (5%) -2.880374  
critical value (10%) -2.576812  
Weak evidence against the null hypothesis  
Fail to reject the null hypothesis  
Data is non-stationary

# n\_sick  
results=seasonal\_decompose(df["n\_sick"], model="add")  
results.plot();

png

png

results.seasonal.plot(figsize=(50,8));

png

png

test\_adfuller(df["n\_sick"], "ADF Test for n\_sick on a daily basis")

DF-Test: ADF Test for n\_sick on a daily basis  
ADF test statistic -3.374317  
p-value 0.011866  
# lags used 21.000000  
# observations 1130.000000  
critical value (1%) -3.436150  
critical value (5%) -2.864101  
critical value (10%) -2.568134  
Strong evidence against the null hypothesis  
Reject of the null hypothesis  
Data is stationary

n\_sick\_m =df["n\_sick"].resample(rule="MS").mean()  
results=seasonal\_decompose(n\_sick\_m, model="add")  
results.plot();

png

png

month\_plot(n\_sick\_m);

png

png

test\_adfuller(n\_sick\_m, "ADF Test for n\_sick on a monthly basis")

DF-Test: ADF Test for n\_sick on a monthly basis  
ADF test statistic -1.795900  
p-value 0.382499  
# lags used 2.000000  
# observations 35.000000  
critical value (1%) -3.632743  
critical value (5%) -2.948510  
critical value (10%) -2.613017  
Weak evidence against the null hypothesis  
Fail to reject the null hypothesis  
Data is non-stationary

n\_sick\_w =df["n\_sick"].resample(rule="W").mean()  
results=seasonal\_decompose(n\_sick\_w, model="add")  
results.plot();

png

png

test\_adfuller(n\_sick\_w, "ADF Test for n\_sick on a weekly basis")

DF-Test: ADF Test for n\_sick on a weekly basis  
ADF test statistic -3.796061  
p-value 0.002948  
# lags used 1.000000  
# observations 164.000000  
critical value (1%) -3.470866  
critical value (5%) -2.879330  
critical value (10%) -2.576255  
Strong evidence against the null hypothesis  
Reject of the null hypothesis  
Data is stationary

#drivers\_atwork

results=seasonal\_decompose(df["drivers\_atwork"], model="add")  
results.plot();

png

png

results.seasonal.plot(figsize=(50,12))

<AxesSubplot:xlabel='date'>

png

png

test\_adfuller(df["drivers\_atwork"], "ADF Test for drivers\_atwork on a daily basis")

DF-Test: ADF Test for drivers\_atwork on a daily basis  
ADF test statistic -1.843787  
p-value 0.358949  
# lags used 21.000000  
# observations 1130.000000  
critical value (1%) -3.436150  
critical value (5%) -2.864101  
critical value (10%) -2.568134  
Weak evidence against the null hypothesis  
Fail to reject the null hypothesis  
Data is non-stationary

drivers\_atwork\_m=df["drivers\_atwork"].resample(rule="MS").mean()  
results=seasonal\_decompose(drivers\_atwork\_m, model="add")  
results.plot();

png

png

month\_plot(drivers\_atwork\_m);

png

png

test\_adfuller(drivers\_atwork\_m, "ADF Test for drivers\_atwork on a monthly basis")

DF-Test: ADF Test for drivers\_atwork on a monthly basis  
ADF test statistic -0.556377  
p-value 0.880538  
# lags used 10.000000  
# observations 27.000000  
critical value (1%) -3.699608  
critical value (5%) -2.976430  
critical value (10%) -2.627601  
Weak evidence against the null hypothesis  
Fail to reject the null hypothesis  
Data is non-stationary

drivers\_atwork\_w=df["drivers\_atwork"].resample(rule="W").mean()  
results=seasonal\_decompose(drivers\_atwork\_w, model="add")  
results.plot();

png

png

test\_adfuller(drivers\_atwork\_w, "ADF Test for drivers\_atwork on a weekly basis")

DF-Test: ADF Test for drivers\_atwork on a weekly basis  
ADF test statistic -1.183565  
p-value 0.680626  
# lags used 4.000000  
# observations 161.000000  
critical value (1%) -3.471633  
critical value (5%) -2.879665  
critical value (10%) -2.576434  
Weak evidence against the null hypothesis  
Fail to reject the null hypothesis  
Data is non-stationary

month\_plot(drivers\_atwork\_m);

png

png

#### Correlation and Causation analysis

df\_calls\_driveratwork=df[["calls", "drivers\_atwork"]]

df\_calls\_driveratwork.corr(method='pearson')

calls

drivers\_atwork

calls

1.000000

0.704728

drivers\_atwork

0.704728

1.000000

#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()

Unnamed: 0

n\_sick

calls

n\_duty

n\_sby

sby\_need

dafted

drivers\_atwork

date

2016-04-01

0

73

8154

1700

90

4

0

1704

2016-04-19

18

57

8676

1700

90

93

3

1793

2016-05-01

30

54

8400

1700

90

34

0

1734

2016-05-04

33

58

8340

1700

90

26

0

1726

2016-05-07

36

57

8868

1700

90

131

41

1831

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()  
  
#plt.figure(figsize=(30,30))  
#labels=["drivers\_atwork","calls divided py linear\_coef"]  
#plt.plot(df["drivers\_atwork"], label="drivers\_atwork")  
#plt.plot(df["calls"]/x\_Value, label="calls divided py linear\_coef")  
#plt.legend()  
#plt.show()

Min\_MSEError 196.22728826407905  
Best linear\_coef value: 4.822822822822823

png

png

plt.hist(df\_adjusted["sby\_need"])

(array([88., 76., 43., 33., 29., 20., 4., 3., 3., 2.]),  
 array([ 2. , 57.3, 112.6, 167.9, 223.2, 278.5, 333.8, 389.1, 444.4,  
 499.7, 555. ]),  
 <BarContainer object of 10 artists>)

png

png

## Calculating the base line success KPIs to compare the model with status quo

* Percentage of standbys being activated is higher than in the current approach of keeping 90 drivers on hold
* Situations with not enough standbys should occur less often than in the current approach.

#### A: Whole dataset

# Percentage of standbys being activated is higher than in the current approach of keeping 90 drivers on hold  
df["sby\_activated"]=df["sby\_need"]-df["dafted"]  
df["percentage\_sby\_activated"]=df["sby\_activated"]/df["n\_sby"]  
percentage\_standby\_activated=df["percentage\_sby\_activated"].mean()  
print(f'{np.round(percentage\_standby\_activated\*100,2)} % of the standby drivers are activated over the whole dataset before optimisation')

20.43 % of the standby drivers are activated over the whole dataset before optimisation

# Situations with not enough standbys should occur less often than in the current approach.  
print(f'In {np.round(len(df[df["sby\_need"]>90])/len(df)\*100,2)} % of the days the amount of 90 standby drivers were exceeded')

In 14.84 % of the days the amount of 90 standby drivers were exceeded

#### B: The last 365 to compare with test data set

# Percentage of standbys being activated is higher than in the current approach of keeping 90 drivers on hold  
df["sby\_activated"]=df["sby\_need"]-df["dafted"]  
df["percentage\_sby\_activated"]=df["sby\_activated"]/df["n\_sby"]  
percentage\_standby\_activated=df["percentage\_sby\_activated"].iloc[len(df)-365:].mean()  
print(f'{np.round(percentage\_standby\_activated\*100,2)} % of the standby drivers are activated over the whole dataset before optimisation')

28.61 % of the standby drivers are activated over the whole dataset before optimisation

df["datfted\_for\_calc"]=df["dafted"].apply(lambda x:x if x>0 else np.nan)  
print(f'In {np.round(df["datfted\_for\_calc"].iloc[len(df)-365:].count()/365\*100,2)} % of the days the amount of 90 standby drivers were exceeded')

In 22.47 % of the days the amount of 90 standby drivers were exceeded

df.head()

Unnamed: 0

n\_sick

calls

n\_duty

n\_sby

sby\_need

dafted

drivers\_atwork

sby\_activated

percentage\_sby\_activated

datfted\_for\_calc

date

2016-04-01

0

73

8154

1700

90

4

0

1704

4

0.044444

NaN

2016-04-02

1

64

8526

1700

90

70

0

1770

70

0.777778

NaN

2016-04-03

2

68

8088

1700

90

0

0

1700

0

0.000000

NaN

2016-04-04

3

71

7044

1700

90

0

0

1700

0

0.000000

NaN

2016-04-05

4

63

7236

1700

90

0

0

1700

0

0.000000

NaN

##### C: Calculating the consumed FTE (full time equivilant) shifts as a third comparison parameter (not the leading indicator ... but never the less interesting)

print(f'In the last 365 days {df["n\_duty"].iloc[len(df)-365:].sum() + df["sby\_need"].iloc[len(df)-365:].sum()} FTE shifts were needed')  
print(f'In the last 365 days {df["n\_sby"].iloc[len(df)-365:].sum() - df["sby\_activated"].iloc[len(df)-365:].sum()} FTE shifts were hold on standby and were not needed')

In the last 365 days 713377 FTE shifts were needed  
In the last 365 days 23451 FTE shifts were hold on standby and were not needed

np.round(df["datfted\_for\_calc"].iloc[len(df)-365:].count(),2)

82