Welcome to the Southern Water Corp Python Case Study!

While working on the Financial unit, you used Microsoft Excel's data analytics capabilities to analyze Southern Water Corp's data.

Now, Joanna Luez — Southern Water Corp's Lead Scientist — has requested that you convert your earlier analysis in Excel to Python Code. After all, with all the formulas in Excel, it can be tricky for others with less experience in Excel to follow.

Excel is an excellent tool for adhoc analysis, but Python is an invaluable tool thanks to its advanced data analysis capabilities that only take a few lines of code to complete.

Please note that this case study is composed of two parts — once you have completed part 1, which involves descriptive statistics, please submit your work and discuss it with your mentor before moving on to part 2.

Let's get started!

Part I: Descriptive Statistics

Step 1: Import Libraries

Import the libraries you'll need for your analysis. You will need the following libraries:

Matplotlib - This is Python's basic plotting library. You'll use the pyplot and dates function collections from matplotlib throughout this case study so we encourage you to important these two specific libraries with their own aliases. Also, include the line **'%matplotlib inline'** so that your graphs are easily included in your notebook. You will need to import DateFormatter from matplotlib as well.

Seaborn - This library will enable you to create aesthetically pleasing plots.

Pandas - This library will enable you to view and manipulate your data in a tabular format.

statsmodel.api - This library will enable you to create statistical models. You will need this library when perfroming regession analysis in Part 2 of this case study.

Place your code here

In [5]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib as mpl
from matplotlib.dates import DateFormatter
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
%matplotlib inline
import matplotlib.dates as md
```

Step 2: Descriptive Statistics

Unfortunately, the data you've received from Southern Water Corp has been split into three files: Desalination_Unit_File 001, Desalination_Unit_File_002, and Desalination_Unit_File_003. You'll need to merge them into a complete dataframe for your analysis. To do this, follow the steps below:

i. Import each of the three separate files and merge them into one dataframe. Suggested names: **(dataframe_1, dataframe_2, dataframe_3)**. Don't forget to use the **header** argument to ensure your columns have meaningful names!

ii. Print descriptive statistics on your combined dataframe using **.describe()** and **.info()**

iii. Set "TIMEFRAME" as the index on your combined dataframe.

In [ ]:
```

```

In [6]:
```
dataframe_1 = pd.read_csv('/Users/LuckyDog/Desktop/Python case study_stu
dents /Desalination_Unit_File_001.csv', header=1)
dataframe_2 = pd.read_csv('/Users/LuckyDog/Desktop/Python case study_stu
dents /Desalination_Unit_File_002.csv', header=1)
dataframe_3 = pd.read_csv('/Users/LuckyDog/Desktop/Python case study_stu
dents /Desalination_Unit_File_003.csv', header=1)
```

In [7]:
```
timeframe=pd.concat([dataframe_1,dataframe_2,dataframe_3])
```

In [8]:
```
timeframe.describe()
```

Out[8]:

	SURJEK_FLOW_METER_1	SURJEK_FLOW_METER_2	ROTATIONAL_PUMP_RPM	SURJEK_PU
count	6998.000000	6998.000000	6998.000000	
mean	5.946164	5.157796	6.607023	
std	20.351494	24.444442	20.843080	
min	-0.527344	-9.118652	-1.000000	
25%	0.000000	-4.766639	-0.687240	
50%	0.313672	-0.351562	-0.013326	
75%	0.704162	0.981540	0.000000	
max	127.221700	313.989300	99.000000	

In [9]:
```
timeframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6998 entries, 0 to 2001
Data columns (total 10 columns):
SURJEK_FLOW_METER_1            6998 non-null float64
SURJEK_FLOW_METER_2            6998 non-null float64
ROTATIONAL_PUMP_RPM            6998 non-null float64
SURJEK_PUMP_TORQUE             6998 non-null float64
MAXIMUM_DAILY_PUMP_TORQUE      6998 non-null float64
SURJEK_AMMONIA_FLOW_RATE       6998 non-null int64
SURJEK_TUBE_PRESSURE           6998 non-null float64
SURJEK_ESTIMATED_EFFICIENCY    6998 non-null float64
PUMP FAILURE (1 or 0)          6997 non-null float64
TIMEFRAME                      6998 non-null object
dtypes: float64(8), int64(1), object(1)
memory usage: 601.4+ KB
```

Step 3: Create a Boxplot

When you look at your dataframe, you should now be able to see the upper and lower quartiles for each row of data. You should now also have a rough sense of the number of entires in each dataset. However, just as you learned when using Excel, creating a visualization of the data using Python is often more informative than viewing the table statistics. Next up — convert the tables you created into a boxplot by following these instructions:

i) Create a boxplot from your combined dataframe using **matplotlib and seaborn** with all the variables plotted out. Note: do any particular variables stand out to you? Title your visualization **"Boxplot for all attributes"** and set the boxplot size to 25 x 5.

Please put your code here

In [10]:
```
timeframe.head()
```

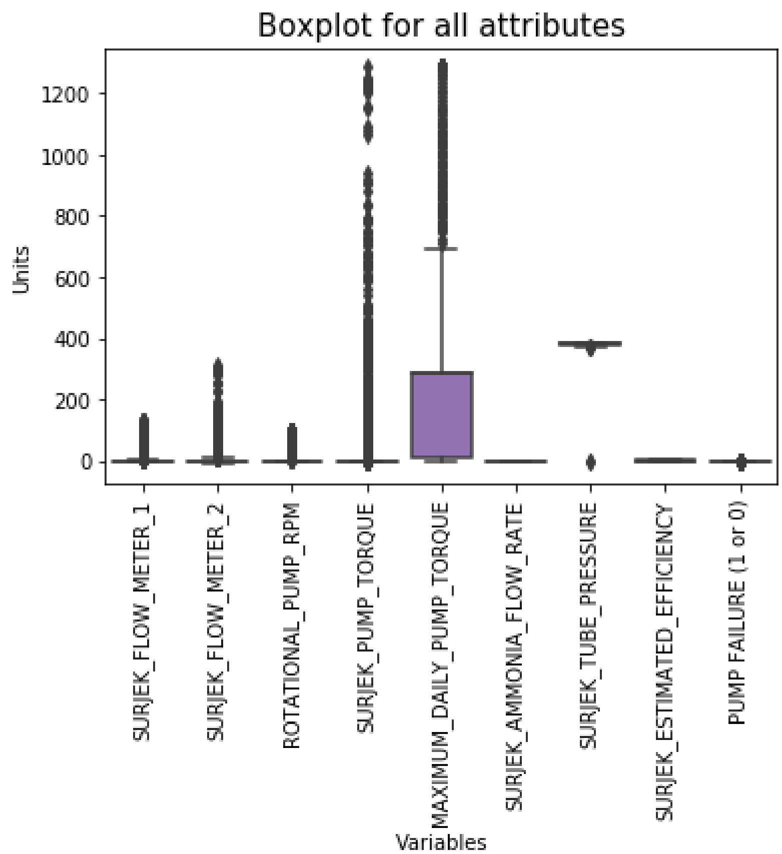
Out[10]:

	SURJEK_FLOW_METER_1	SURJEK_FLOW_METER_2	ROTATIONAL_PUMP_RPM	SURJEK_PUMP_
0	0.0	-4.768066	0.0	
1	0.0	-4.855957	0.0	
2	0.0	-7.447938	0.0	
3	0.0	-8.745117	0.0	
4	0.0	-6.877441	0.0	

In [11]:
```
timeframe=timeframe.dropna()
timeframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6997 entries, 0 to 2001
Data columns (total 10 columns):
SURJEK_FLOW_METER_1            6997 non-null float64
SURJEK_FLOW_METER_2            6997 non-null float64
ROTATIONAL_PUMP_RPM            6997 non-null float64
SURJEK_PUMP_TORQUE             6997 non-null float64
MAXIMUM_DAILY_PUMP_TORQUE      6997 non-null float64
SURJEK_AMMONIA_FLOW_RATE       6997 non-null int64
SURJEK_TUBE_PRESSURE           6997 non-null float64
SURJEK_ESTIMATED_EFFICIENCY    6997 non-null float64
PUMP FAILURE (1 or 0)          6997 non-null float64
TIMEFRAME                      6997 non-null object
dtypes: float64(8), int64(1), object(1)
memory usage: 601.3+ KB
```

In [12]:
```
_=sns.boxplot(data=timeframe)
plt.title('Boxplot for all attributes', fontsize=15)
plt.xlabel('Variables')
plt.ylabel('Units')
plt.rcParams['figure.figsize'] = (25,5)
plt.xticks(rotation=90)
plt.show()
```



Splitting the data into tow different sets, indicating normal behavior and abnormal behavior , show us the the tree variables that were shown by a boxplot , which indicate that METER 2, PUMP TORQUE AND MAXIMUM TORQUE , are correlated to the pump failure!

You would probably note that it might seem that some variables, due to their range and size of values, dwarfs some of the other variables which makes the variation difficult to see.

Perhaps, we should remove these variables and look at the box plot again?

Step 4: Create a Filtered Boxplot

i) Create the same boxplot from Step 3, but this time, filter out SURJEK_PUMP_TORQUE and MAXIMUM_DAILY_PUMP_TORQUE. Create a new dataframe and apply a filter named **'dataframe_filt'.** Title this boxplot 'Boxplot without Pump Torque, or Max Daily Pump Torque'. We have provided the filter list for you.

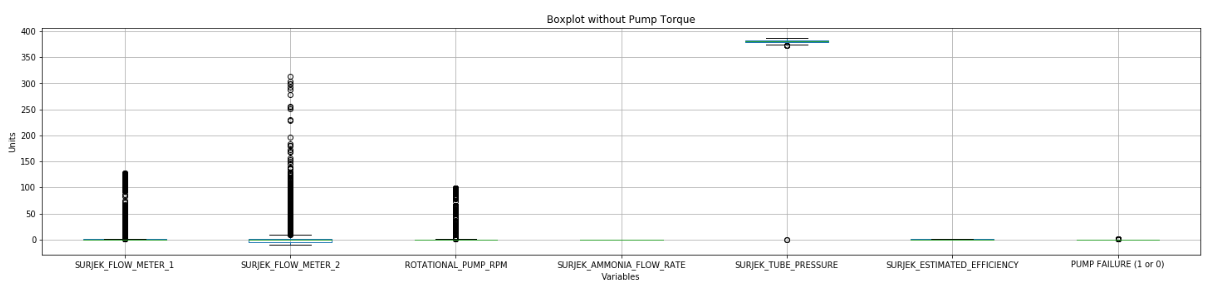
Open-ended question:

Beyond pump torque and max daily pump torque, do any other attributes seem to 'stand out'?

Please put your code here

In [13]:

```
#Below is the first part of the code
#filt = ['SURJEK_FLOW_METER_1', 'SURJEK_FLOW_METER_2', 'ROTATIONAL_PUMP_RPM',
       #'SURJEK_AMMONIA_FLOW_RATE', 'SURJEK_TUBE_PRESSURE',
       #'SURJEK_ESTIMATED_EFFICIENCY', 'PUMP FAILURE (1 or 0)']
#plt.rcParams['figure.figsize'] = (25,5)
#--write your code below------
#_=sns.boxplot(data=timeframe)

_ = timeframe.boxplot(column=['SURJEK_FLOW_METER_1', 'SURJEK_FLOW_METER_2', 'ROTATIONAL_PUMP_RPM',
       'SURJEK_AMMONIA_FLOW_RATE', 'SURJEK_TUBE_PRESSURE',
       'SURJEK_ESTIMATED_EFFICIENCY', 'PUMP FAILURE (1 or 0)'])
plt.title('Boxplot without Pump Torque')
plt.xlabel('Variables')
plt.ylabel('Units')
plt.show()
```



Sujek_Meter_2 show a correlation to the Pump Failure

Step 5: Filter Your Boxplot by Column Value

i) Using the whole dataset, create another boxplot using the whole dataset but this time, compare the distributions for when Pump Failure is 1 (The Pump has failed) and 0 (Pump is in normal operations). You will be creating two boxplots using the 'PUMP FAILURE (1 or 0)' column in the dataset. We have provided a few lines of code to get you started. Once complete, you should be able to see how much quicker it is to apply filters in Python than it is in Excel.

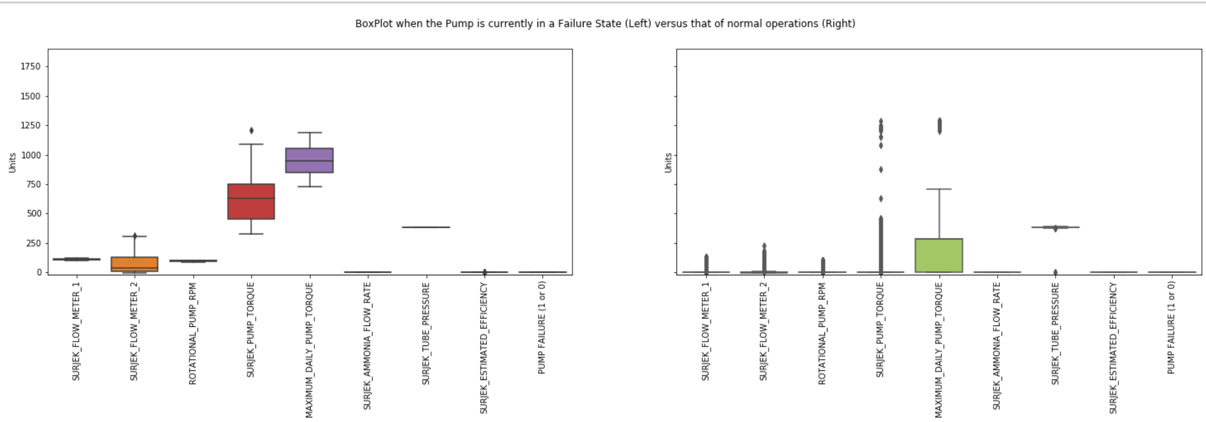
Note: Please display the two boxplots side-by-side. You can do this by creating a shared X axis or by creating two axes and looping through them while using the pyplot command.

Open-ended Question:

What variables seem to have the largest variation when the Pump has failed?

Please put your code here

In [14]:
```
f, axes = plt.subplots(1,2, sharey=True)
f.suptitle("BoxPlot when the Pump is currently in a Failure State (Left) versus that of normal operations (Right)")

df_1 = timeframe[timeframe['PUMP FAILURE (1 or 0)']==1]
sns.boxplot(data=df_1,orient='v', ax = axes[0], )
plt.ylabel('Units')

df_2=timeframe[timeframe['PUMP FAILURE (1 or 0)']==0]
sns.boxplot(data=df_2, orient='v',palette='Set2', ax=axes[1])
plt.ylabel('Units')

for ax in f.axes:
    mpl.pyplot.sca(ax)
    plt.ylim(-20,1900)
    plt.xticks(rotation=90)
    plt.ylabel('Units')
plt.show()
```



Meter_2, Pump_Torque, Maximum_daily seems to have the largest variation when the Pump has failed

From analysing the boxplots, you'll notice that there seem to be a number of outliers.

When you did this work in Excel, you used the interquartile ranges to remove the outliers from each column. Happily, Python allows you to do this same process more quickly and efficiently, as you'll see when working on Step 6.

Step 6: Create Quartiles

i) Create two new variables called Q1 and Q3. q1 should contain the 25th percentile for all columns in the dataframe while Q3 should contain the 75th percentile for all the columns in the dataframe.

ii) Calculate the interquartile range **(IQR = Q3 - Q1)** for all columns in the dataframe and print it to the screen.

Please put your code here

In [15]:

```
Q1=timeframe.quantile(0.25)
Median=timeframe.quantile(0.50)
Q3=timeframe.quantile(0.75)
IQR=Q3-Q1
lower=Q1*1.5
upper=Q3*1.5
print(IQR)
```

```
SURJEK_FLOW_METER_1              0.704173
SURJEK_FLOW_METER_2              5.746893
ROTATIONAL_PUMP_RPM              0.687126
SURJEK_PUMP_TORQUE               0.349459
MAXIMUM_DAILY_PUMP_TORQUE      276.209717
SURJEK_AMMONIA_FLOW_RATE         0.000000
SURJEK_TUBE_PRESSURE             3.662100
SURJEK_ESTIMATED_EFFICIENCY      1.240822
PUMP FAILURE (1 or 0)            0.000000
dtype: float64
```

Step 7: Identify Outliers

How many outliers do you have? What will happen to your dataset if you remove them all? Let's find out!

i) Calculate how many entries you currently have in the original dataframe.

ii) Using the quartiles and IQR previously calculated, identify the number of entries you'd have if you were to remove the outliers.

ii) Find the proportion of outliers that exist in the dataset.

Ensure your dataframe doesn't include the attribute TIMEFRAME - if it does, please drop this attribute for now.

Please put your code here

In [16]:

```
#i) Calculate how many entries you currently have in the original datafr
ame.
entries=len(timeframe)
entries
```

Out[16]:

```
6997
```

In [17]:

```
df = timeframe.drop('TIMEFRAME', axis=1)
```

In [18]:
```
remove_out = (df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))
print(remove_out)
```

```
      SURJEK_FLOW_METER_1  SURJEK_FLOW_METER_2  ROTATIONAL_PUMP_RPM  \
0                   False                False                False
1                   False                False                False
2                   False                False                False
3                   False                False                False
4                   False                False                False
...                   ...                  ...                  ...
1997                False                False                False
1998                False                 True                False
1999                False                False                False
2000                False                False                False
2001                False                False                False

      SURJEK_PUMP_TORQUE  MAXIMUM_DAILY_PUMP_TORQUE  SURJEK_AMMONIA_FLO
W_RATE  \
0                   True                      False
False
1                   True                      False
False
2                   True                      False
False
3                   True                      False
False
4                   True                      False
False
...                  ...                        ...
...
1997               False                       True
False
1998               False                       True
False
1999               False                       True
False
2000               False                       True
False
2001               False                       True
False

      SURJEK_TUBE_PRESSURE  SURJEK_ESTIMATED_EFFICIENCY  PUMP FAILURE
(1 or 0)
0                     True                        False
False
1                     True                        False
False
2                    False                        False
False
3                    False                        False
False
4                    False                        False
False
...                    ...                          ...
...
1997                 False                        False
False
1998                 False                        False
False
1999                 False                        False
```

```
False
2000                False                          False
False
2001                False                          False
False

[6997 rows x 9 columns]
```

In [19]:

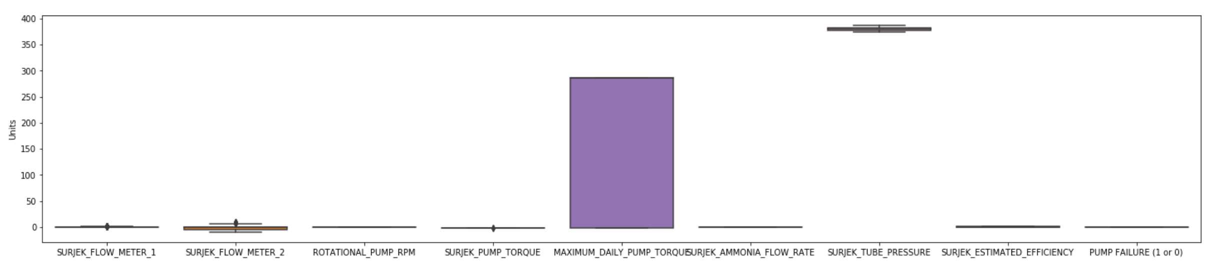
```
#ii) Using the quartiles and IQR previously calculated, identify the num
ber of entries you'd have
#if you were to remove the outliers. 3854 observations of 9 variables.
df_out = df[~((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis
=1)]
print(df_out.shape)
```

```
(3854, 9)
```

In [20]:

```
df_out.info()
sns.boxplot(data=df_out)
plt.ylabel('Units')


plt.show()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3854 entries, 413 to 178
Data columns (total 9 columns):
SURJEK_FLOW_METER_1            3854 non-null float64
SURJEK_FLOW_METER_2            3854 non-null float64
ROTATIONAL_PUMP_RPM            3854 non-null float64
SURJEK_PUMP_TORQUE             3854 non-null float64
MAXIMUM_DAILY_PUMP_TORQUE      3854 non-null float64
SURJEK_AMMONIA_FLOW_RATE       3854 non-null int64
SURJEK_TUBE_PRESSURE           3854 non-null float64
SURJEK_ESTIMATED_EFFICIENCY    3854 non-null float64
PUMP FAILURE (1 or 0)          3854 non-null float64
dtypes: float64(8), int64(1)
memory usage: 301.1 KB
```



In [21]:

```
proportion = (3854/6997)*100
proportion
```

Out[21]:

```
55.080748892382445
```

In [22]:
```
# ---write your code below----------------

#We have provided the print line, you need to provide the calculation af
ter the quoted text:

print ("When we have not removed any outliers from the dataset, we have
 " + str(entries) + " entries")
print ("When we were to removed any outliers from the dataset, we have "
+ str(df_out.shape) + " entries")
print ("The proportion of outliers which exist when compared to the data
frame are: " + str(proportion))
```

```
When we have not removed any outliers from the dataset, we have 6997 en
tries
When we were to removed any outliers from the dataset, we have (3854,
9) entries
The proportion of outliers which exist when compared to the dataframe a
re: 55.080748892382445
```

Step 8: Create a Boxplot without Outliers

With the dataset now stripped of outliers, create the following boxplots:

i) A boxplot when PUMP FAILURE is 1

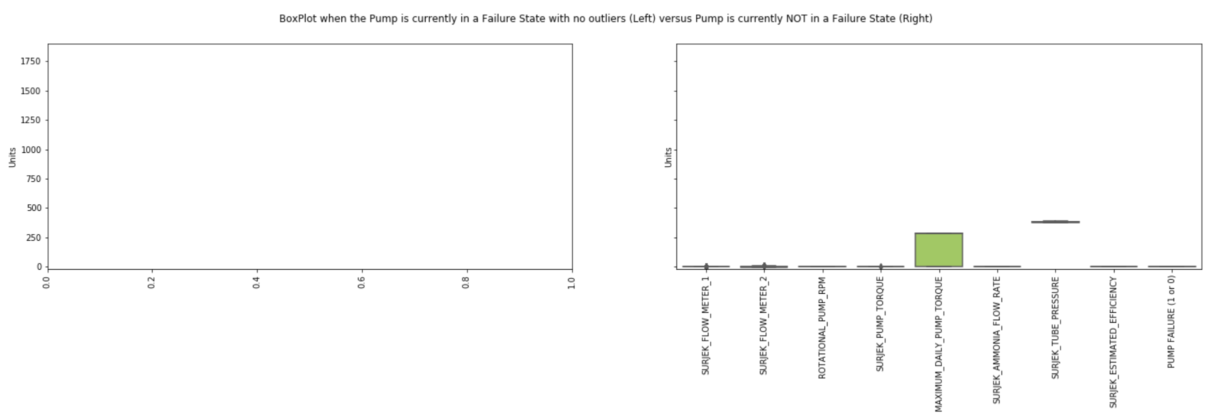
ii) A boxplot when PUMP FAILURE is 0

Note 1: Removing outliers is very situational and specific. Outliers can skew the dataset unfavourably; however, if you are doing a failure analysis, it is likely those outliers actually contain valuable insights you will want to keep as they represent a deviation from the norm that you'll need to understand.

Note 2: Please display the two boxplots side-by-side. You can do this by creating a shared X axis or by creating two axes and looping through them while using the pyplot command.

Please put your code here

In [23]:
```
#Below is the first part of the code
f, axes = plt.subplots(1, 2, sharey=True)
f.suptitle("BoxPlot when the Pump is currently in a Failure State with n
o outliers (Left) versus Pump is currently NOT in a Failure State (Righ
t)")
mpl.rcParams['figure.figsize'] = (15,5)
#---write your code below-------------

df_2 = df_out[df_out['PUMP FAILURE (1 or 0)']==1]
sns.boxplot(data=df_2, orient='v',palette='Set2', ax=axes[1])
plt.ylabel('Units')


df_3 = df_out[df_out['PUMP FAILURE (1 or 0)']==0]
sns.boxplot(data= df_3, orient='v',palette='Set2', ax=axes[1])


for ax in f.axes:
    mpl.pyplot.sca(ax)
    plt.ylim(-20,1900)
    plt.xticks(rotation=90)
    plt.ylabel('Units')
plt.show()
```



Based on the boxplots you've created, you've likely come to the conclusion that, for this case study, you actually _shouldn't_ remove the outliers, as you are attempting to understand the Pump Failure Behavior.

Step 9: Plot and Examine Each Column

We have provided a filtered column list for you.

Using a loop, iterate through each of the Column Names and plot the data. (You can either make your X-axis the Timeframe variable or you can leave it blank and use the row numbers as an index).

Find the minimum (min) and maximum (max) time in the dataframe. Use Tight_layout. Include a title with min and max time.

Note: For each plot, ensure that you have a dual axis set up so you can see the Pump Behaviour (0 or 1) on the second Y-axis, and the attribute (e.g. SURJEK_FLOW_METER_1) on the first Y-Axis. It might be helpful to give the failureState it's own color and add a legend to the axis to make it easier to view.

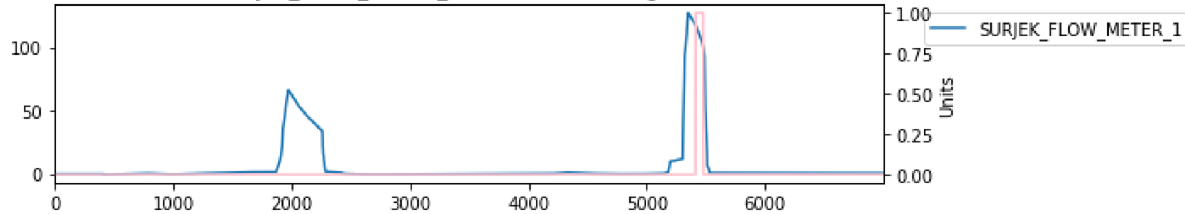
Check out this link to learn how to do this: https://matplotlib.org/gallery/api/two_scales.html (https://matplotlib.org/gallery/api/two_scales.html)

Note: Please ensure that the dataframe you are plotting contains all the outliers and that the Pump Failure Behaviour includes both the 0 and 1 State.

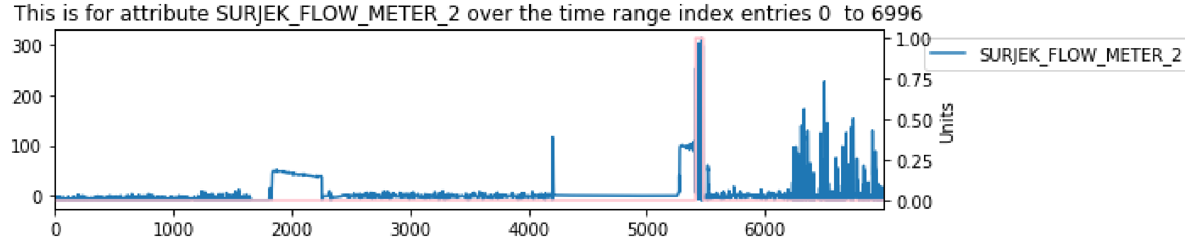
Please put your code here

In [24]:
```
#Below is the first part of the code
timeframe=timeframe.dropna()
timeframe=timeframe.reset_index()
filt = ['SURJEK_FLOW_METER_1', 'SURJEK_FLOW_METER_2', 'ROTATIONAL_PUMP_R
PM',
       'SURJEK_PUMP_TORQUE', 'MAXIMUM_DAILY_PUMP_TORQUE',
       'SURJEK_AMMONIA_FLOW_RATE', 'SURJEK_TUBE_PRESSURE',
       'SURJEK_ESTIMATED_EFFICIENCY']
filt2 = ['PUMP FAILURE (1 or 0)']
colList = timeframe[filt].columns
mpl.rcParams['figure.figsize'] = (10,2)
#---write your code below-------

for i in colList:
    failureState = timeframe[filt2] #
    ax = timeframe[i].plot()
    ax2 = ax.twinx()
    ax2.plot(failureState, 'pink')
    ax.legend(bbox_to_anchor=(1.04,1), loc="upper left")

    minTime = timeframe.index.min()
    maxTime= timeframe.index.max()
    plt.tight_layout()
    plt.title("This is for attribute " + i + " over the time range index
entries " + str(minTime) + " " +  " to " + str(maxTime))
    plt.ylabel('Units')
    #---To Here------
    plt.show()
```

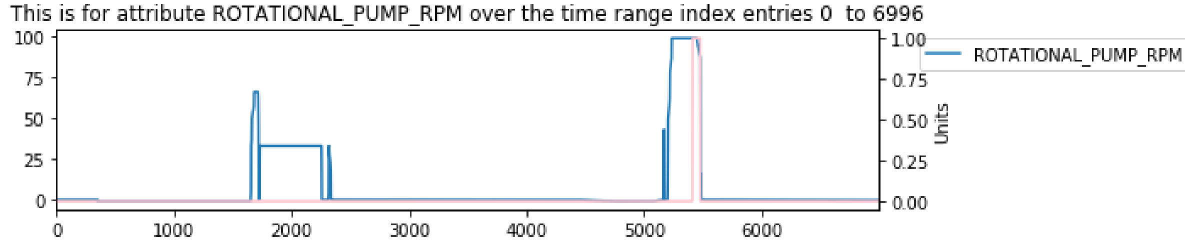
This is for attribute SURJEK_FLOW_METER_1 over the time range index entries 0 to 6996



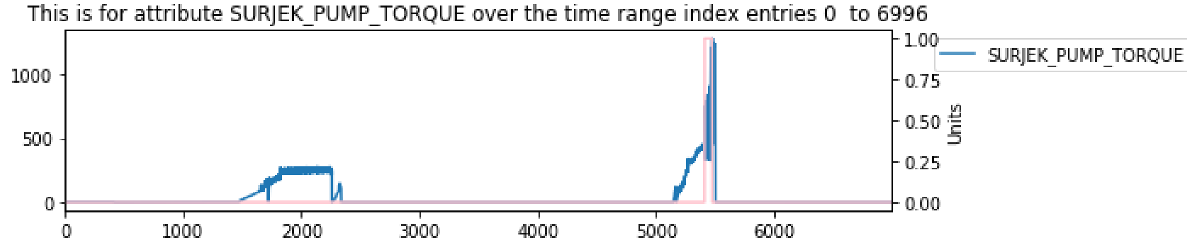
This is for attribute SURJEK_FLOW_METER_2 over the time range index entries 0 to 6996



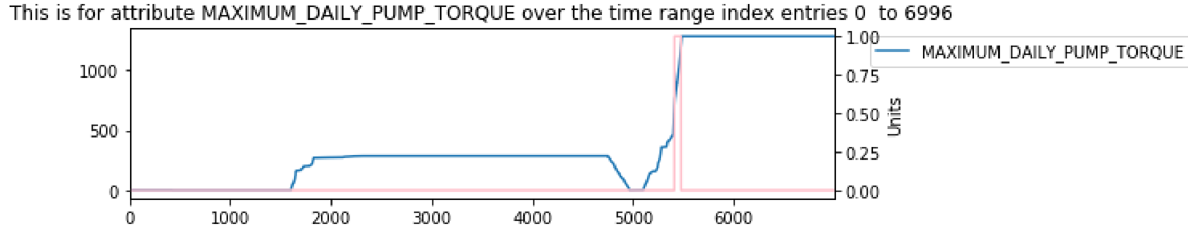
This is for attribute ROTATIONAL_PUMP_RPM over the time range index entries 0 to 6996



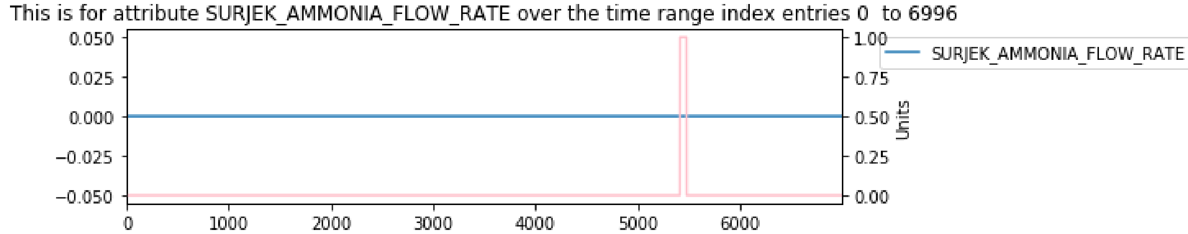
This is for attribute SURJEK_PUMP_TORQUE over the time range index entries 0 to 6996



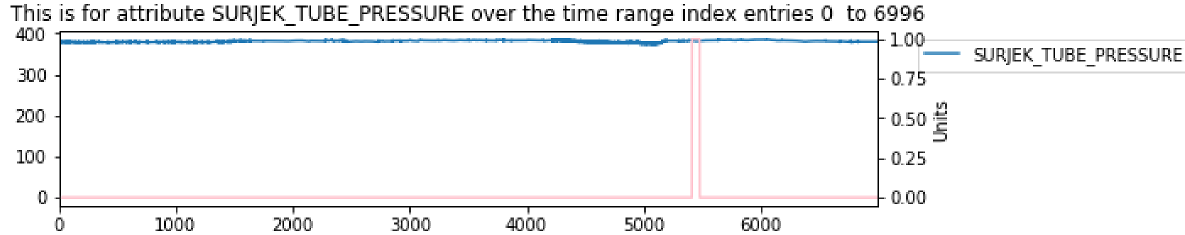
This is for attribute MAXIMUM_DAILY_PUMP_TORQUE over the time range index entries 0 to 6996

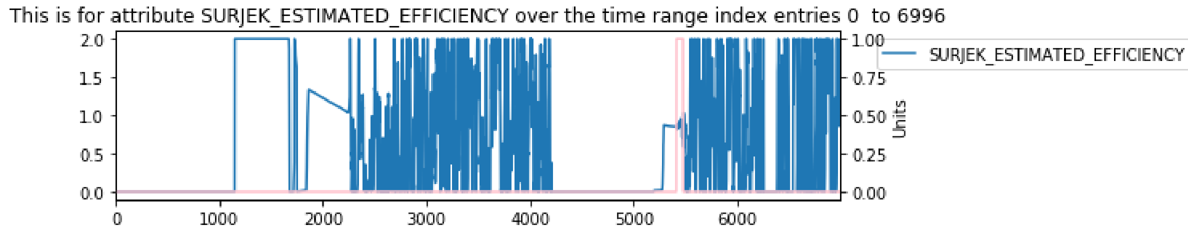


This is for attribute SURJEK_AMMONIA_FLOW_RATE over the time range index entries 0 to 6996



This is for attribute SURJEK_TUBE_PRESSURE over the time range index entries 0 to 6996





Of course, given that all the attributes have varying units, you might need more than one plot to make sense of all this data. For this next step, let's view the information by comparing the **ROLILNG DEVIATIONS** over a 30-point period.

As the deviations will likely be a lot lower, the scale should be much simpler to view on one plot. Make sure that you include the 'PUMP FAILURE 1 or 0' attribute on the secondary Y-axis.

Hint: Remember to make use of the Dual-Axis plot trick you learned in the previous exercise!

Step 10: Create a Plot for Pump Failures Over a Rolling Time Period

i) Apply a rolling standard deviation to the dataframe using a rolling window size of '30'.

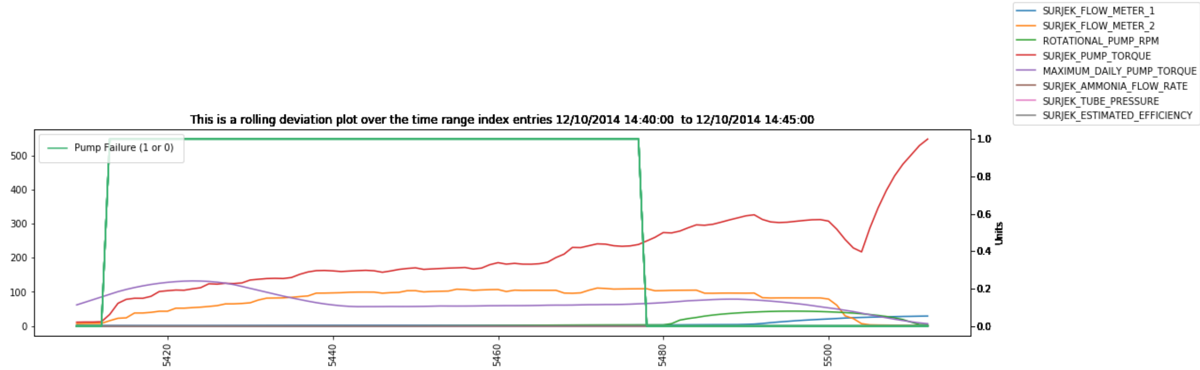
ii) Re-plot all variables for the time period 10/12/2014 14:40 to 10/12/2014 14:45, focusing specifically on the first Pump “Failure”.

Open-ended Question: Do any particular variables seem to move in relation to the failure event?

Please put your code here

In [25]:

```
#Below is the first part of the code

from datetime import datetime
timeframe=pd.concat([dataframe_1,dataframe_2,dataframe_3])
timeframe['TIMEFRAME'] = pd.to_datetime(timeframe['TIMEFRAME']).apply(la
mbda x: x.strftime('%d/%m/%Y %H:%M:%S')if not pd.isnull(x) else '')
filt = ['SURJEK_FLOW_METER_1', 'SURJEK_FLOW_METER_2', 'ROTATIONAL_PUMP_R
PM',
       'SURJEK_PUMP_TORQUE', 'MAXIMUM_DAILY_PUMP_TORQUE',
       'SURJEK_AMMONIA_FLOW_RATE', 'SURJEK_TUBE_PRESSURE',
       'SURJEK_ESTIMATED_EFFICIENCY','PUMP FAILURE (1 or 0)', 'TIMEFRAM
E']
filt2 = ['PUMP FAILURE (1 or 0)']
filt3 = ['SURJEK_FLOW_METER_1', 'SURJEK_FLOW_METER_2', 'ROTATIONAL_PUMP_
RPM',
       'SURJEK_PUMP_TORQUE', 'MAXIMUM_DAILY_PUMP_TORQUE',
       'SURJEK_AMMONIA_FLOW_RATE', 'SURJEK_TUBE_PRESSURE',
       'SURJEK_ESTIMATED_EFFICIENCY']
colList = timeframe[filt].columns
mpl.rcParams['figure.figsize'] = (15,4)
#timeframe.set_index('TIMEFRAME', inplace=True)
timeframe.reset_index(drop=True,inplace=True)
#----write your code below-------

rollingDF = timeframe[filt3].rolling(30).std()
rollingDF = rollingDF.join(timeframe[['PUMP FAILURE (1 or 0)', 'TIMEFRAM
E']], how='inner')
rollingDF_filter = rollingDF.loc[(rollingDF['TIMEFRAME'] >= "12/10/2014
 14:40:00")&(rollingDF['TIMEFRAME'] <="12/10/2014 14:45:00")]

#Ryan moved filt3 from here up to provided code section****************
**********

fig = plt.figure()
ax = plt.axes()

#Loop through the Plot
for i in filt3:
    ax.plot(rollingDF_filter.index, rollingDF_filter[i], label=i)
    ax2 = ax.twinx()
    ax2.plot(rollingDF_filter[filt2], 'mediumseagreen', label='Pump Fail
ure (1 or 0)')

    ax.xaxis.set_tick_params(rotation=90)

    plt.tight_layout()
    minTime = rollingDF_filter['TIMEFRAME'].min()
    maxTime= rollingDF_filter['TIMEFRAME'].max()
    plt.title("This is a rolling deviation plot over the time range inde
x entries " + str(minTime) + " " +  " to " + str(maxTime))
    plt.ylabel('Units')

ax.legend(bbox_to_anchor=(1.04,1), loc="lower left")
ax2.legend(loc='upper left', borderpad=1)
plt.show()
```



The particular variables Surjek 2,Pump_Torque and Maximum seems to move in relation to the failure even

In [26]:
```
rollingDF.tail()
```

Out[26]:

	SURJEK_FLOW_METER_1	SURJEK_FLOW_METER_2	ROTATIONAL_PUMP_RPM	SURJEK_PU
6993	0.001272	6.659791	0.001726	
6994	0.001299	6.651137	0.001763	
6995	0.001312	6.558681	0.001780	
6996	0.001344	6.561854	0.001824	
6997	0.001376	6.558285	0.001868	

Part II: Inferential Statistical Analysis

When you performed inferential statistics for Southern Water Corp using Excel, you made use of the data analysis package to create a heatmap using the correlation function. The heatmap showed the attributes that strongly correlated to Pump Failure.

Now, you'll create a heatmap using Seaborn's heatmap function — another testament to the fact that having Matplotlib and Seaborn in your toolbox will allow you to quickly create beautiful graphics that provide key insights.

Step 11: Create a Heatmap

i) Using Seaborn's heatmap function, create a heatmap that clearly shows the correlations (including R Squared) for all variables (excluding those with consistent 0 values such as Ammonia Flow Rate).

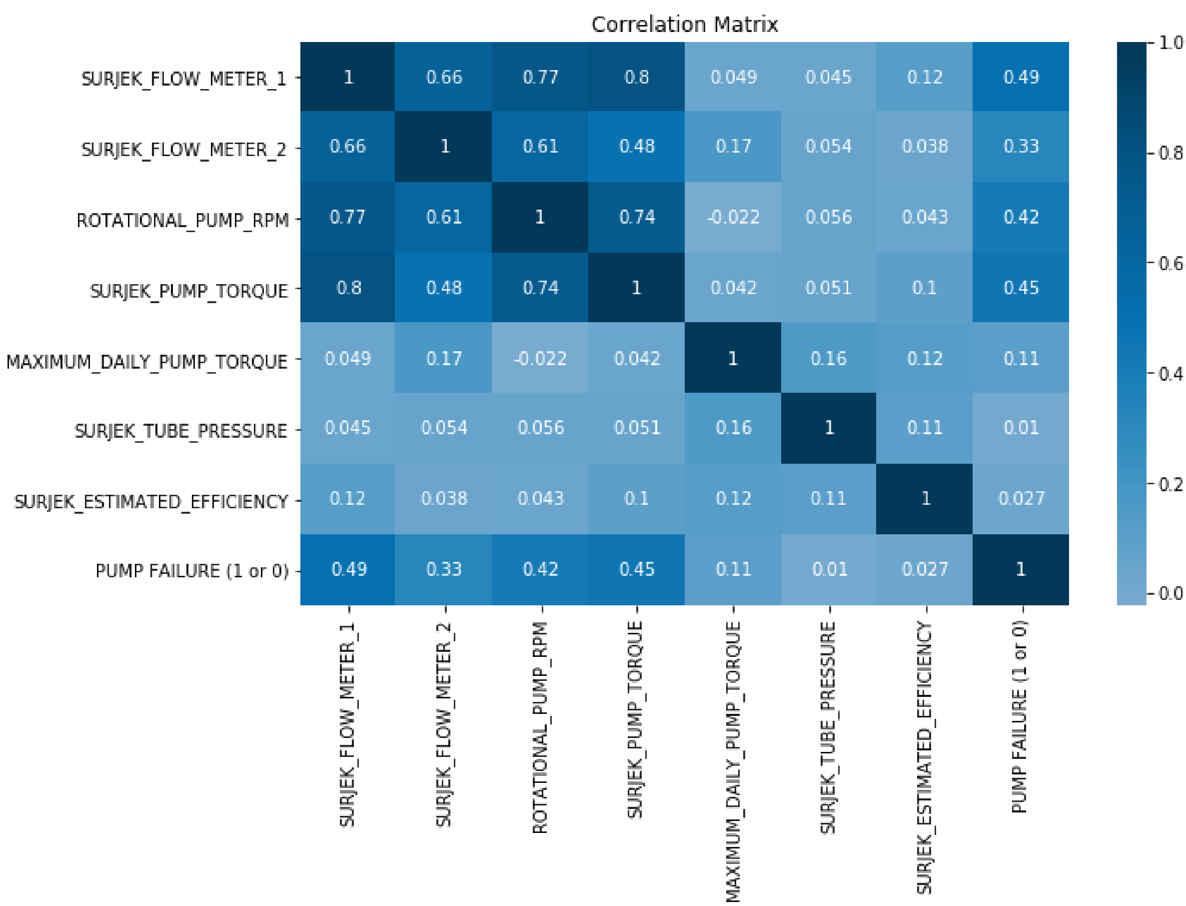
Note: We have provided the filter list and created the dataframe for you.

Link: (https://seaborn.pydata.org/generated/seaborn.heatmap.html (https://seaborn.pydata.org/generated/seaborn.heatmap.html))

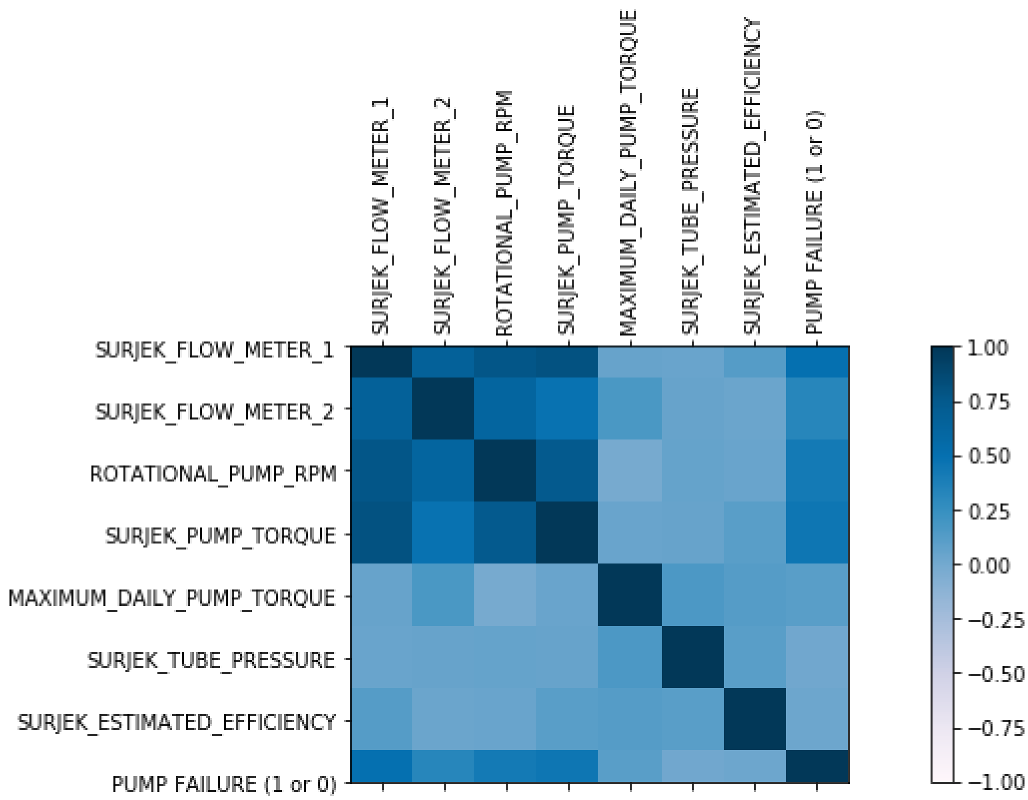
Please put your code here

In [27]:
```
#Below is the first part of the code
from datetime import datetime
dataframe =pd.concat([dataframe_1,dataframe_2,dataframe_3])
dataframe['TIMEFRAME'] = pd.to_datetime(dataframe['TIMEFRAME'], format="
%d/%m/%Y %H:%M:%S", infer_datetime_format=True )
dataframe.set_index('TIMEFRAME', inplace=True)

filt = ['SURJEK_FLOW_METER_1', 'SURJEK_FLOW_METER_2', 'ROTATIONAL_PUMP_R
PM',
        'SURJEK_PUMP_TORQUE', 'MAXIMUM_DAILY_PUMP_TORQUE',
        'SURJEK_TUBE_PRESSURE',
        'SURJEK_ESTIMATED_EFFICIENCY','PUMP FAILURE (1 or 0)']
dataframe = dataframe[filt]
#----write your code below--------

fig, ax = plt.subplots(figsize=(10,6))
sns.heatmap(dataframe.corr(), center=0, cmap='PuBu',annot=True)
ax.set_title("Correlation Matrix")
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

Out[27]: (8.0, 0.0)



In [28]:
```
corr = dataframe.corr()
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(corr,cmap='PuBu', vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = np.arange(0,len(dataframe.columns),1)
ax.set_xticks(ticks)
plt.xticks(rotation=90)
ax.set_yticks(ticks)
ax.set_xticklabels(dataframe.columns)
ax.set_yticklabels(dataframe.columns)
plt.show()
```



Open-ended Question:

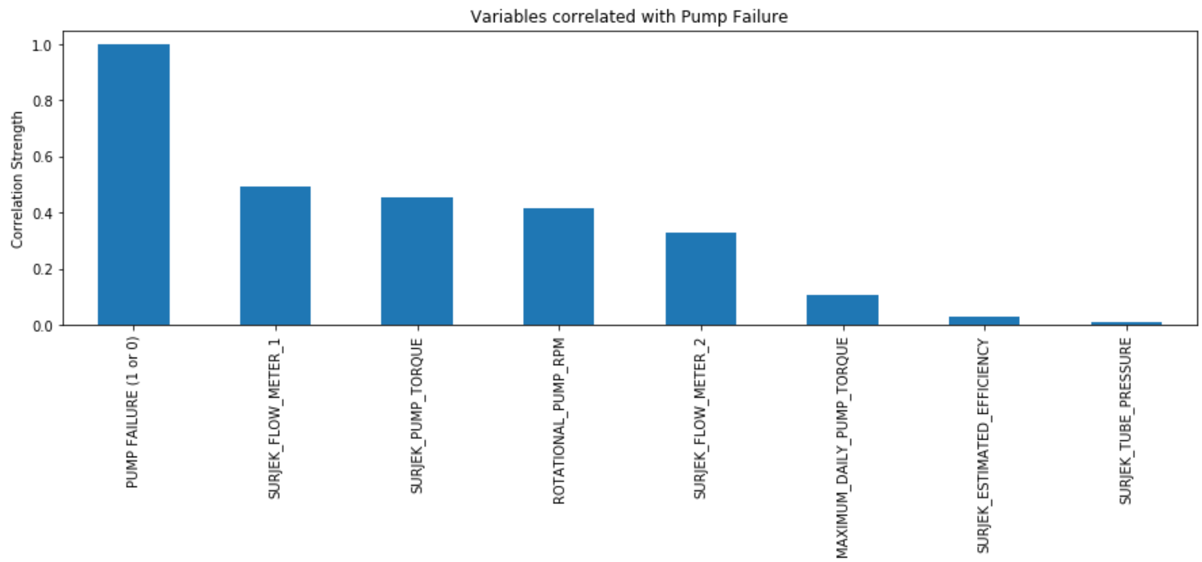
Which variables seem to correlate with Pump Failure? Surjek 2,Pump_Torque and Maximum seems to correlate with the Pump Failure

Step 12: Create a Barplot of Correlated Features

Create a barplot that shows the correlated features against PUMP FAILURE (1 or 0), in descending order.

Please put your code here

In [29]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

corr = corr.sort_values("PUMP FAILURE (1 or 0)", ascending=False)
corr['PUMP FAILURE (1 or 0)'].plot(kind='bar')
plt.title("Variables correlated with Pump Failure")
plt.ylabel("Correlation Strength")
plt.show()
```



Step 13: Create a Rolling Standard Deviation Heatmap

Previously, you created a correlation matrix using 'raw' variables. This time, you'll transform 'raw' variables using a rolling standard deviation.

i) Apply a rolling standard deviation to the dataframe using a rolling window size of '30'.

ii) Using the newly created rolling standard deviation dataframe, use the Seaborn heatmap function to replot this dataframe into a heatmap.

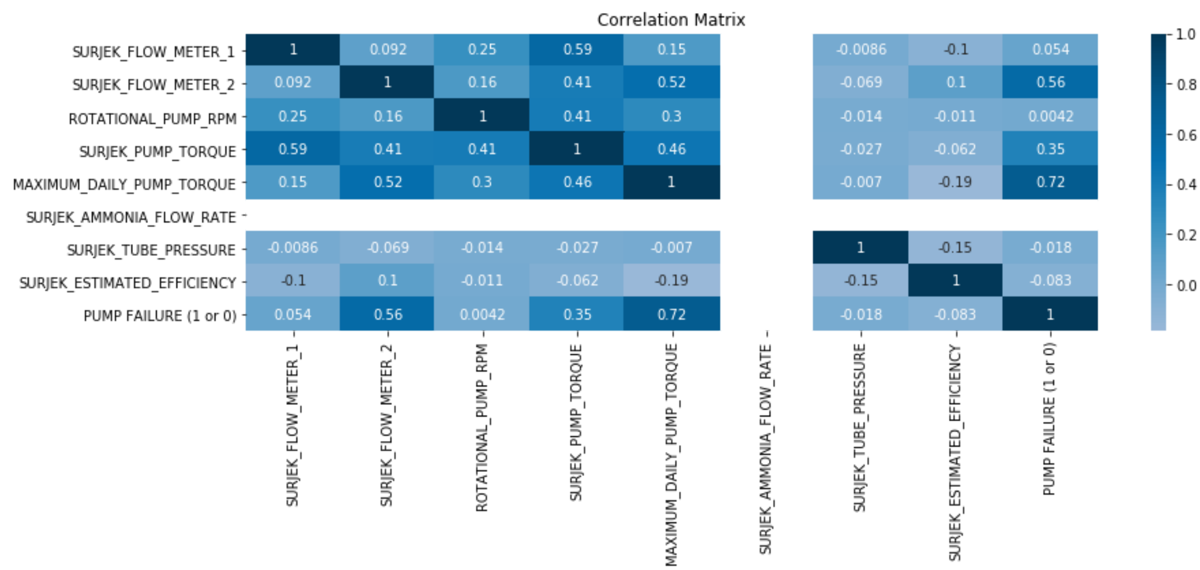
Do any variables stand out? If yes, list these out below your heatmap. Surjek 2,Pump_Torque and Maximum

Note: We have provided the initial dataframe and filters.

Please put your code here

In [30]:
```
#Below is the first part of the code
dataframe =pd.concat([dataframe_1,dataframe_2,dataframe_3])
dataframe['TIMEFRAME'] = pd.to_datetime(dataframe['TIMEFRAME'], format="
%d/%m/%Y %H:%M:%S", infer_datetime_format=True )
dataframe.set_index('TIMEFRAME', inplace=True)
filt = ['SURJEK_FLOW_METER_1', 'SURJEK_FLOW_METER_2', 'ROTATIONAL_PUMP_R
PM',
        'SURJEK_PUMP_TORQUE', 'MAXIMUM_DAILY_PUMP_TORQUE',
        'SURJEK_TUBE_PRESSURE',
        'SURJEK_ESTIMATED_EFFICIENCY','PUMP FAILURE (1 or 0)']
#----write your code below------
fig, ax = plt.subplots(figsize=(14,4))
ax.set_ylim(sorted(ax.get_xlim(), reverse=True))
sns.heatmap(rollingDF.corr(), center=0, cmap='PuBu',annot=True)
ax.set_title("Correlation Matrix")
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

Out[30]: (9.0, 0.0)



Creating a Multivariate Regression Model

When you worked on this case study in Excel, you went through the tricky process of using the rolling standard deviation variables to generate a regression equation. Happily, this process is much simpler in Python.

For this step, you'll be using the statsmodel.api library you imported earlier and calling the Ordinary Least Squares Regression to create a multivariate regression model (which is a linear regression model with more than one independent variable).

Step 14: Use OLS Regression

i) Using the OLS Regression Model in the statsmodel.api library, create a regression equation that models the Pump Failure (Y-Variable) against all your independent variables, which include every other variable that is not PUMP FAILURE (1 or 0). What is the R Squared for the model and what does this signify?

ii) Repeat i) but this time use the rolling standard deviation variables you created previously. What is the R Squared for the model and what does this signify?

Open-ended Question:

Which linear regression model seems to be a better fit? OLS Regression model fits better on this Analyses, its easier to visualizes the variable over the equation aftyer it is plotted in a graffic.

Note: We have provided the initial dataframe and filter list.

Please put your code here

In [31]:
```
#Answer for step i):
#Below is the first part of the code
dataframe =pd.concat([dataframe_1,dataframe_2,dataframe_3])
dependentVar = dataframe['PUMP FAILURE (1 or 0)']
filt = ['SURJEK_FLOW_METER_1', 'SURJEK_FLOW_METER_2', 'ROTATIONAL_PUMP_R
PM',
       'SURJEK_PUMP_TORQUE', 'MAXIMUM_DAILY_PUMP_TORQUE',
       'SURJEK_AMMONIA_FLOW_RATE', 'SURJEK_TUBE_PRESSURE',
       'SURJEK_ESTIMATED_EFFICIENCY', 'PUMP FAILURE (1 or 0)']
#----write your code below------

dataframe_two = dataframe[filt]
dataframe_two['PumpFailure'] = dependentVar
dataframe_two = dataframe_two.fillna(0)
X = dataframe_two.drop(['PUMP FAILURE (1 or 0)', 'PumpFailure'],axis=1)
X = sm.add_constant(X)
y = dataframe_two['PumpFailure']
OLSmodel = sm.OLS(y, X)
OLSmodelResult = OLSmodel.fit()
OLSmodelResult.summary()
```

```
/opt/anaconda3/lib/python3.7/site-packages/numpy/core/fromnumeric.py:24
95: FutureWarning: Method .ptp is deprecated and will be removed in a f
uture version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
/opt/anaconda3/lib/python3.7/site-packages/statsmodels/base/model.py:12
94: RuntimeWarning: invalid value encountered in true_divide
  return self.params / self.bse
/opt/anaconda3/lib/python3.7/site-packages/scipy/stats/_distn_infrastru
cture.py:901: RuntimeWarning: invalid value encountered in greater
  return (a < x) & (x < b)
/opt/anaconda3/lib/python3.7/site-packages/scipy/stats/_distn_infrastru
cture.py:901: RuntimeWarning: invalid value encountered in less
  return (a < x) & (x < b)
/opt/anaconda3/lib/python3.7/site-packages/scipy/stats/_distn_infrastru
cture.py:1892: RuntimeWarning: invalid value encountered in less_equal
  cond2 = cond0 & (x <= _a)
```

Out[31]:

OLS Regression Results

Dep. Variable:	PumpFailure	**R-squared:**	0.264
Model:	OLS	**Adj. R-squared:**	0.264
Method:	Least Squares	**F-statistic:**	358.8
Date:	Thu, 30 Jul 2020	**Prob (F-statistic):**	0.00
Time:	09:28:11	**Log-Likelihood:**	7548.9
No. Observations:	6998	**AIC:**	-1.508e+04
Df Residuals:	6990	**BIC:**	-1.503e+04
Df Model:	7		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	0.1293	0.056	2.318	0.020	0.020	0.239
SURJEK_FLOW_METER_1	0.0017	9.98e-05	16.797	0.000	0.001	0.002
SURJEK_FLOW_METER_2	-0.0001	5.77e-05	-2.085	0.037	-0.000	-7.17e-06
ROTATIONAL_PUMP_RPM	0.0003	8.17e-05	4.110	0.000	0.000	0.000
SURJEK_PUMP_TORQUE	0.0001	1.43e-05	7.011	0.000	7.23e-05	0.000
MAXIMUM_DAILY_PUMP_TORQUE	2.036e-05	2.18e-06	9.321	0.000	1.61e-05	2.46e-05
SURJEK_AMMONIA_FLOW_RATE	0	0	nan	nan	0	0
SURJEK_TUBE_PRESSURE	-0.0004	0.000	-2.516	0.012	-0.001	-8.17e-05
SURJEK_ESTIMATED_EFFICIENCY	-0.0052	0.001	-3.866	0.000	-0.008	-0.003

Omnibus:	7983.490	**Durbin-Watson:**	0.045
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	972591.134
Skew:	5.871	**Prob(JB):**	0.00
Kurtosis:	59.548	**Cond. No.**	5.34e+19

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.19e-30. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

In [32]:
```
#answer for step ii):
#Below is the first part of the code
dataframe_two =pd.concat([dataframe_1,dataframe_2,dataframe_3])
dependentVar = dataframe_two['PUMP FAILURE (1 or 0)']
filt = ['SURJEK_FLOW_METER_1', 'SURJEK_FLOW_METER_2', 'ROTATIONAL_PUMP_R
PM',
       'SURJEK_PUMP_TORQUE', 'MAXIMUM_DAILY_PUMP_TORQUE',
       'SURJEK_AMMONIA_FLOW_RATE', 'SURJEK_TUBE_PRESSURE',
       'SURJEK_ESTIMATED_EFFICIENCY', 'PUMP FAILURE (1 or 0)']

#----write your code below------

dataframe_two = dataframe_two[filt].rolling(30).std()
dataframe_two['PumpFailure'] = dependentVar
#dataframe_two = dataframe_two.fillna(0)
dataframe_two = dataframe_two.dropna()
dataframe_two = dataframe_two.reset_index(drop=True)
X = dataframe_two.drop(['PUMP FAILURE (1 or 0)', 'PumpFailure'],axis=1)
X = sm.add_constant(X)
y = dataframe_two['PumpFailure']
OLSmodel = sm.OLS(y, X)
OLSmodelResult = OLSmodel.fit()
OLSmodelResult.summary()
```

Out[32]:

OLS Regression Results

Dep. Variable:	PumpFailure	**R-squared:**	0.626
Model:	OLS	**Adj. R-squared:**	0.625
Method:	Least Squares	**F-statistic:**	1655.
Date:	Thu, 30 Jul 2020	**Prob (F-statistic):**	0.00
Time:	09:28:11	**Log-Likelihood:**	9800.5
No. Observations:	6939	**AIC:**	-1.958e+04
Df Residuals:	6931	**BIC:**	-1.953e+04
Df Model:	7		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-0.0123	0.001	-11.193	0.000	-0.015	-0.010
SURJEK_FLOW_METER_1	-0.0024	0.000	-7.262	0.000	-0.003	-0.002
SURJEK_FLOW_METER_2	0.0016	6.43e-05	25.547	0.000	0.002	0.002
ROTATIONAL_PUMP_RPM	-0.0065	0.000	-30.602	0.000	-0.007	-0.006
SURJEK_PUMP_TORQUE	0.0003	2.55e-05	10.066	0.000	0.000	0.000
MAXIMUM_DAILY_PUMP_TORQUE	0.0059	8.84e-05	66.749	0.000	0.006	0.006
SURJEK_AMMONIA_FLOW_RATE	-1.459e-19	2.88e-19	-0.506	0.613	-7.11e-19	4.19e-19
SURJEK_TUBE_PRESSURE	0.0002	0.000	0.396	0.692	-0.001	0.001
SURJEK_ESTIMATED_EFFICIENCY	0.0028	0.002	1.171	0.242	-0.002	0.007

Omnibus:	2595.796	**Durbin-Watson:**	0.084
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	400181.596
Skew:	0.704	**Prob(JB):**	0.00
Kurtosis:	40.177	**Cond. No.**	8.93e+17

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.61e-29. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

Great job creating those regressive equations! You've reached the final step of this case study!

Step 15: Validate Predictions

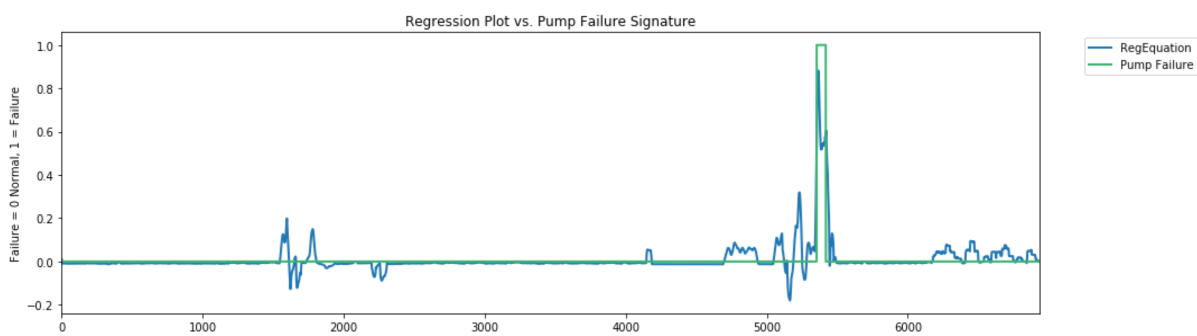
i) Use the regression equation you created in the previous step and apply the .predict() function to the dataframe to see whether or not your model 'picks' up the Pump Failure Event.

ii) Plot the rolling linear regression equation against the attribute 'PUMP FAILURE (1 or 0)'

Note: Please ensure all axes are clearly labelled and ensure that you use Dual Axes to plot this. Make the line widths wider than 1 so the plots are easier to see. We have provided the initial figure size.

Please put your code here

In [33]:
```
#Below is the first part of the code
mpl.rcParams['figure.figsize'] = (15,4)
#----write your code below------
ax = OLSmodelResult.predict(X).plot(linewidth=2, marker='', label="RegEq
uation")
ax.set_ylabel("Regression Equation Prediction of Failre signal")
plt.ylabel("Failure = 0 Normal, 1 = Failure")
ax.plot(dataframe_two.PumpFailure, 'mediumseagreen', linewidth=2, marker
='',label='Pump Failure')
ax.legend(bbox_to_anchor=(1.04,1), loc="upper left")
plt.ylabel("Failure = 0 Normal, 1 = Failure")
plt.tight_layout()


plt.title("Regression Plot vs. Pump Failure Signature")
plt.show()
```



You've made it to the end of this challenging case study — well done! You've now converted all of the analysis you did for Southern Water Corp using Excel into Python. You created visualizations using Seaborn, manipulated datasets with pandas, and so much more! This case study was designed to give you practice using Python to analyze datasets both large and small — you can now apply these skills to work you do throughout your career as a data analyst.

Great job! Being able to complete this case study means that you're now fluent in Python for data analysis! Congratulations!