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 [1]:

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.dates as md
from matplotlib.dates import DateFormatter
%matplotlib inline
import numpy as np
import seaborn as sbn
import pandas as pd
import statsmodels.api as sm
```

In [2]:

```
pip install seaborn
```

```
Requirement already satisfied: seaborn in /opt/anaconda3/lib/python3.
7/site-packages (0.10.0)
Requirement already satisfied: pandas>=0.22.0 in /opt/anaconda3/lib/p
ython3.7/site-packages (from seaborn) (1.0.1)
Requirement already satisfied: matplotlib>=2.1.2 in /opt/anaconda3/li
b/python3.7/site-packages (from seaborn) (3.1.3)
Requirement already satisfied: scipy>=1.0.1 in /opt/anaconda3/lib/pyt
hon3.7/site-packages (from seaborn) (1.4.1)
Requirement already satisfied: numpy>=1.13.3 in /opt/anaconda3/lib/py
thon3.7/site-packages (from seaborn) (1.18.1)
Requirement already satisfied: pytz>=2017.2 in /opt/anaconda3/lib/pyt
hon3.7/site-packages (from pandas>=0.22.0->seaborn) (2019.3)
Requirement already satisfied: python-dateutil>=2.6.1 in /opt/anacond
a3/lib/python3.7/site-packages (from pandas>=0.22.0->seaborn) (2.8.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0
.1 in /opt/anaconda3/lib/python3.7/site-packages (from matplotlib>=2.
1.2->seaborn) (2.4.6)
Requirement already satisfied: cycler>=0.10 in /opt/anaconda3/lib/pyt
hon3.7/site-packages (from matplotlib>=2.1.2->seaborn) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/anaconda3/li
b/python3.7/site-packages (from matplotlib>=2.1.2->seaborn) (1.1.0)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/lib/python3
.7/site-packages (from python-dateutil>=2.6.1->pandas>=0.22.0->seabor
n) (1.14.0)
Requirement already satisfied: setuptools in /opt/anaconda3/lib/pytho
n3.7/site-packages (from kiwisolver>=1.0.1->matplotlib>=2.1.2->seabor
n) (46.0.0.post20200309)
Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```
conda install seaborn
```

```
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /opt/anaconda3

  added / updated specs:
    - seaborn


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    conda-4.8.3                |           py37_0         2.8 MB
    ------------------------------------------------------------
                                           Total:         2.8 MB

The following packages will be UPDATED:

  conda                                       4.8.2-py37_0 --> 4.8.3
-py37_0



Downloading and Extracting Packages
conda-4.8.3          | 2.8 MB    | ##################################
### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done

Note: you may need to restart the kernel to use updated packages.
```

In [4]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib.dates as mdates
import pandas as pd
import numpy as np

```

In [5]:
```
import seaborn as sns
from statsmodels.formula.api import ols
```

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 [6]:

```
dataframe_1 = pd.read_csv('Desalination_Unit_File_001.csv')
dataframe_1 = pd.DataFrame(dataframe_1)
dataframe_2 = pd.read_csv('Desalination_Unit_File_002.csv')
dataframe_2 = pd.DataFrame(dataframe_2)
dataframe_3 = pd.read_csv('Desalination_Unit_File_003.csv')
dataframe_2 = pd.DataFrame(dataframe_2)
```

In [7]:

```
merge = pd.concat([dataframe_1,dataframe_2, dataframe_3])
```

In [8]:

```
merge = pd.DataFrame(merge)
merge.set_index('TIMEFRAME', inplace = True)
merge.describe()
```

Out[8]:

	SURJEK_FLOW_METER_1	SURJEK_FLOW_METER_2	ROTATIONAL_PUMP_RPM	SURJEK
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]:
```
merge.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 6998 entries, 9/12/14 0:00 to 10/12/14 16:52
Data columns (total 9 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   SURJEK_FLOW_METER_1           6998 non-null   float64
 1   SURJEK_FLOW_METER_2           6998 non-null   float64
 2   ROTATIONAL_PUMP_RPM           6998 non-null   float64
 3   SURJEK_PUMP_TORQUE            6998 non-null   float64
 4   MAXIMUM_DAILY_PUMP_TORQUE     6998 non-null   float64
 5   SURJEK_AMMONIA_FLOW_RATE      6998 non-null   int64
 6   SURJEK_TUBE_PRESSURE          6998 non-null   float64
 7   SURJEK_ESTIMATED_EFFICIENCY   6998 non-null   float64
 8   PUMP FAILURE (1 or 0)         6997 non-null   float64
dtypes: float64(8), int64(1)
memory usage: 546.7+ 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]:
```
merge.head()
```

Out[10]:

	SURJEK_FLOW_METER_1	SURJEK_FLOW_METER_2	ROTATIONAL_PUMP_RPM	SI
TIMEFRAME				
9/12/14 0:00	0.0	-4.768066	0.0	
9/12/14 0:01	0.0	-4.855957	0.0	
9/12/14 0:01	0.0	-7.447938	0.0	
9/12/14 0:01	0.0	-8.745117	0.0	
9/12/14 0:02	0.0	-6.877441	0.0	

In [11]:
```
plt.figure(figsize=(25,5))
```

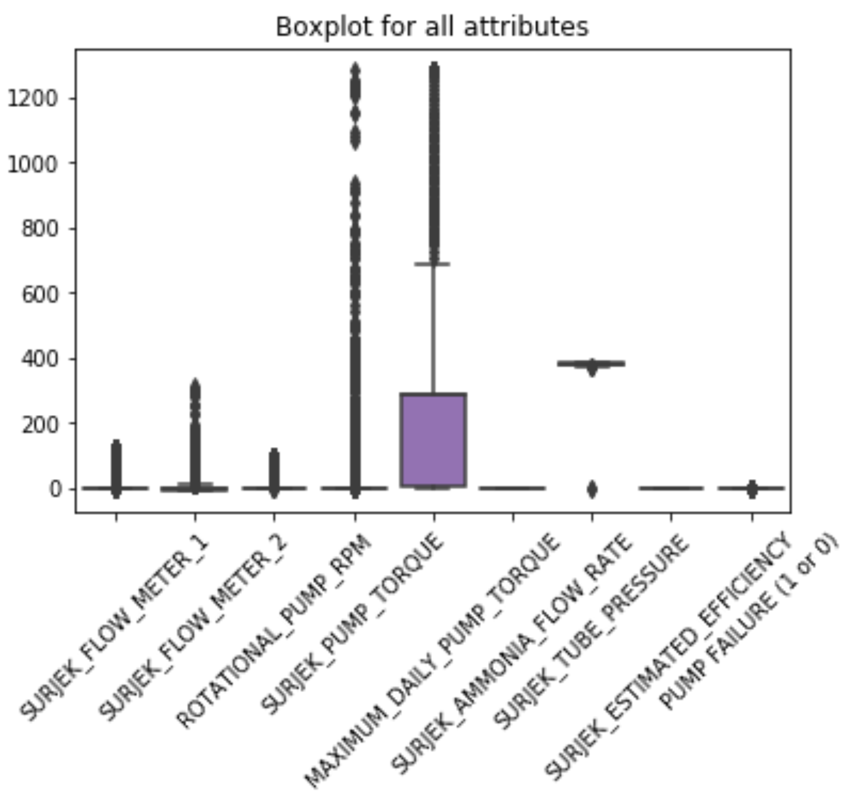
Out[11]:
```
<Figure size 1800x360 with 0 Axes>

<Figure size 1800x360 with 0 Axes>
```

In [12]:

```python

fig1 = sns.boxplot(data=merge)
fig1.set_title("Boxplot for all attributes")
fig1.set_xticklabels(fig1.get_xticklabels(),rotation=45)
```

Out[12]:

```
[Text(0, 0, 'SURJEK_FLOW_METER_1'),
 Text(0, 0, 'SURJEK_FLOW_METER_2'),
 Text(0, 0, 'ROTATIONAL_PUMP_RPM'),
 Text(0, 0, 'SURJEK_PUMP_TORQUE'),
 Text(0, 0, 'MAXIMUM_DAILY_PUMP_TORQUE'),
 Text(0, 0, 'SURJEK_AMMONIA_FLOW_RATE'),
 Text(0, 0, 'SURJEK_TUBE_PRESSURE'),
 Text(0, 0, 'SURJEK_ESTIMATED_EFFICIENCY'),
 Text(0, 0, 'PUMP FAILURE (1 or 0)')]
```



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'?

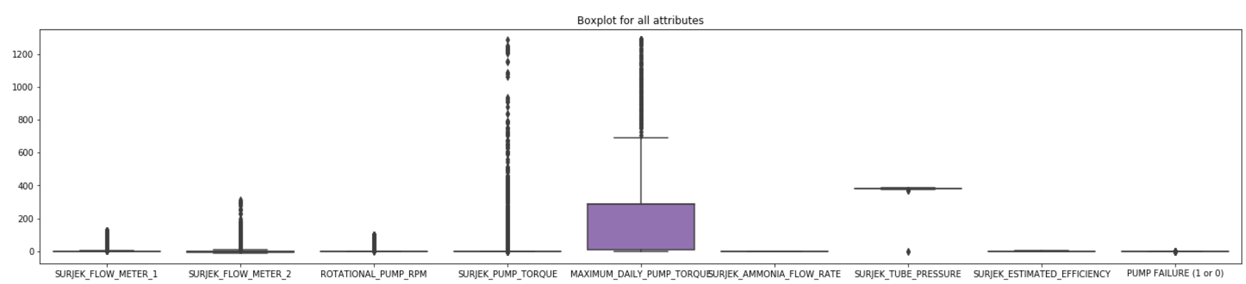
Please put your code here

In [13]:

```
#Below is the first part of the code
filt = ['SURJEK_FLOW_METER_1', 'SURJEK_FLOW_METER_2', 'ROTATIONAL_
        '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------
fig2 = sns.boxplot(data=merge)
fig2.set_title("Boxplot for all attributes")
```

Out[13]:

```
Text(0.5, 1.0, 'Boxplot for all attributes')
```



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?

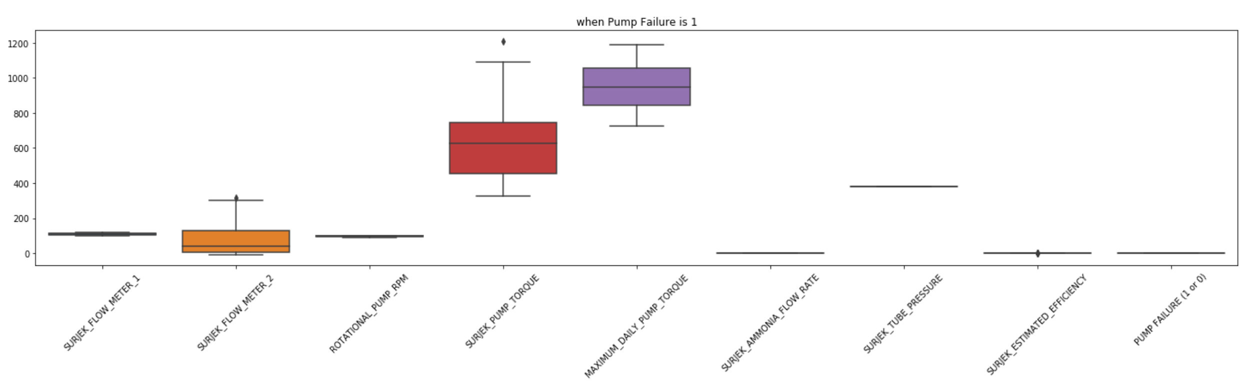
Please put your code here

In [14]:

```
# when Pump Failure is 1 The Pump has failed
filt1 =merge[merge['PUMP FAILURE (1 or 0)'] == 1.0]
filt1.head(5)
fig3 = sns.boxplot(data=filt1)
fig3.set_title("when Pump Failure is 1")
fig3.set_xticklabels(fig3.get_xticklabels(),rotation=45)
```

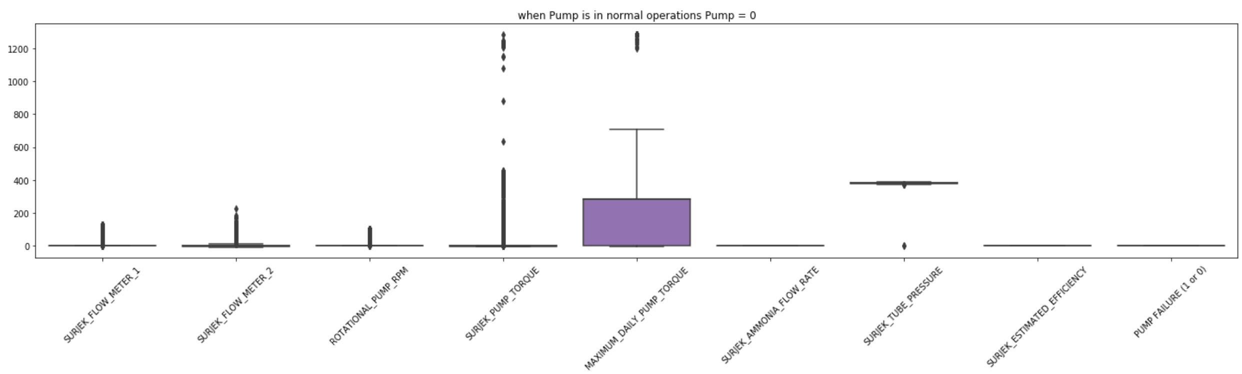
Out[14]:

```
[Text(0, 0, 'SURJEK_FLOW_METER_1'),
 Text(0, 0, 'SURJEK_FLOW_METER_2'),
 Text(0, 0, 'ROTATIONAL_PUMP_RPM'),
 Text(0, 0, 'SURJEK_PUMP_TORQUE'),
 Text(0, 0, 'MAXIMUM_DAILY_PUMP_TORQUE'),
 Text(0, 0, 'SURJEK_AMMONIA_FLOW_RATE'),
 Text(0, 0, 'SURJEK_TUBE_PRESSURE'),
 Text(0, 0, 'SURJEK_ESTIMATED_EFFICIENCY'),
 Text(0, 0, 'PUMP FAILURE (1 or 0)')]
```



In [15]:
```python
# when Pump is in normal operations Pump = 0
filt2 =merge[merge['PUMP FAILURE (1 or 0)'] == 0.0]
filt2.head(5)
fig4 = sns.boxplot(data=filt2)
fig4.set_title("when Pump is in normal operations Pump = 0")
fig4.set_xticklabels(fig4.get_xticklabels(),rotation=45)
```

Out[15]:
```
[Text(0, 0, 'SURJEK_FLOW_METER_1'),
 Text(0, 0, 'SURJEK_FLOW_METER_2'),
 Text(0, 0, 'ROTATIONAL_PUMP_RPM'),
 Text(0, 0, 'SURJEK_PUMP_TORQUE'),
 Text(0, 0, 'MAXIMUM_DAILY_PUMP_TORQUE'),
 Text(0, 0, 'SURJEK_AMMONIA_FLOW_RATE'),
 Text(0, 0, 'SURJEK_TUBE_PRESSURE'),
 Text(0, 0, 'SURJEK_ESTIMATED_EFFICIENCY'),
 Text(0, 0, 'PUMP FAILURE (1 or 0)')]
```



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 [16]:

```
Q1 = merge.quantile(0.25)
Q3 = merge.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
SURJEK_FLOW_METER_1               0.704162
SURJEK_FLOW_METER_2               5.748178
ROTATIONAL_PUMP_RPM               0.687240
SURJEK_PUMP_TORQUE                0.350032
MAXIMUM_DAILY_PUMP_TORQUE       276.315522
SURJEK_AMMONIA_FLOW_RATE          0.000000
SURJEK_TUBE_PRESSURE              3.662100
SURJEK_ESTIMATED_EFFICIENCY       1.240724
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 [17]:

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


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

#We have provided the print line, you need to provide the calculati

noout = merge[~((merge < (Q1 - 1.5 * IQR)) |(merge > (Q3 + 1.5 * IQ
print ("When we have removed outliers from the dataset, we have " +
#We have provided the print line, you need to provide the calculati
print ("When we have not removed any outliers from the dataset, we
print ("The proportion of outliers which exist when compared to the
```

```
When we have removed outliers from the dataset, we have 3855 entries
When we have not removed any outliers from the dataset, we have 6998
entries
The proportion of outliers which exist when compared to the dataframe
are: 0.5508716776221778
```

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.

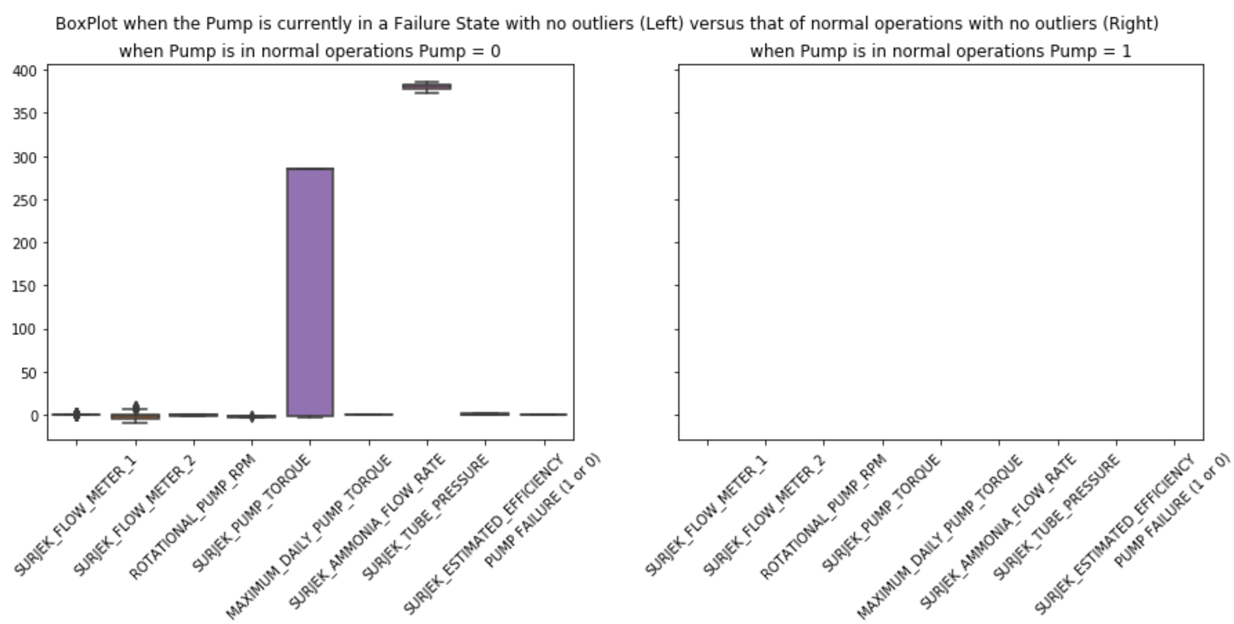
Please put your code here

In [18]:

```python
#Below is the first part of the code
f, axes = plt.subplots(1, 2, sharey=True, figsize = (15,5))
f.suptitle("BoxPlot when the Pump is currently in a Failure State
#plt.rcParams['figure.figsize'] = (15,5)

#---write your code below-------------
normal =noout[noout['PUMP FAILURE (1 or 0)'] == 0.0]
normal.head(5)
fig5 = sns.boxplot(data=normal, ax = axes[0])
fig5.set_title("when Pump is in normal operations Pump = 0")
fig5.set_xticklabels(fig5.get_xticklabels(),rotation=45)
fail =noout[noout['PUMP FAILURE (1 or 0)'] == 1.0]
fail.head(5)
fig6 = sns.boxplot(data=fail, ax = axes[1])
fig6.set_title("when Pump is in normal operations Pump = 1")
fig6.set_xticklabels(fig6.get_xticklabels(),rotation=45)
```

Out[18]:

```
[Text(0, 0, 'SURJEK_FLOW_METER_1'),
 Text(0, 0, 'SURJEK_FLOW_METER_2'),
 Text(0, 0, 'ROTATIONAL_PUMP_RPM'),
 Text(0, 0, 'SURJEK_PUMP_TORQUE'),
 Text(0, 0, 'MAXIMUM_DAILY_PUMP_TORQUE'),
 Text(0, 0, 'SURJEK_AMMONIA_FLOW_RATE'),
 Text(0, 0, 'SURJEK_TUBE_PRESSURE'),
 Text(0, 0, 'SURJEK_ESTIMATED_EFFICIENCY'),
 Text(0, 0, 'PUMP FAILURE (1 or 0)')]
```



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)](https://matplotlib.org/gallery/api/two_scales.html)

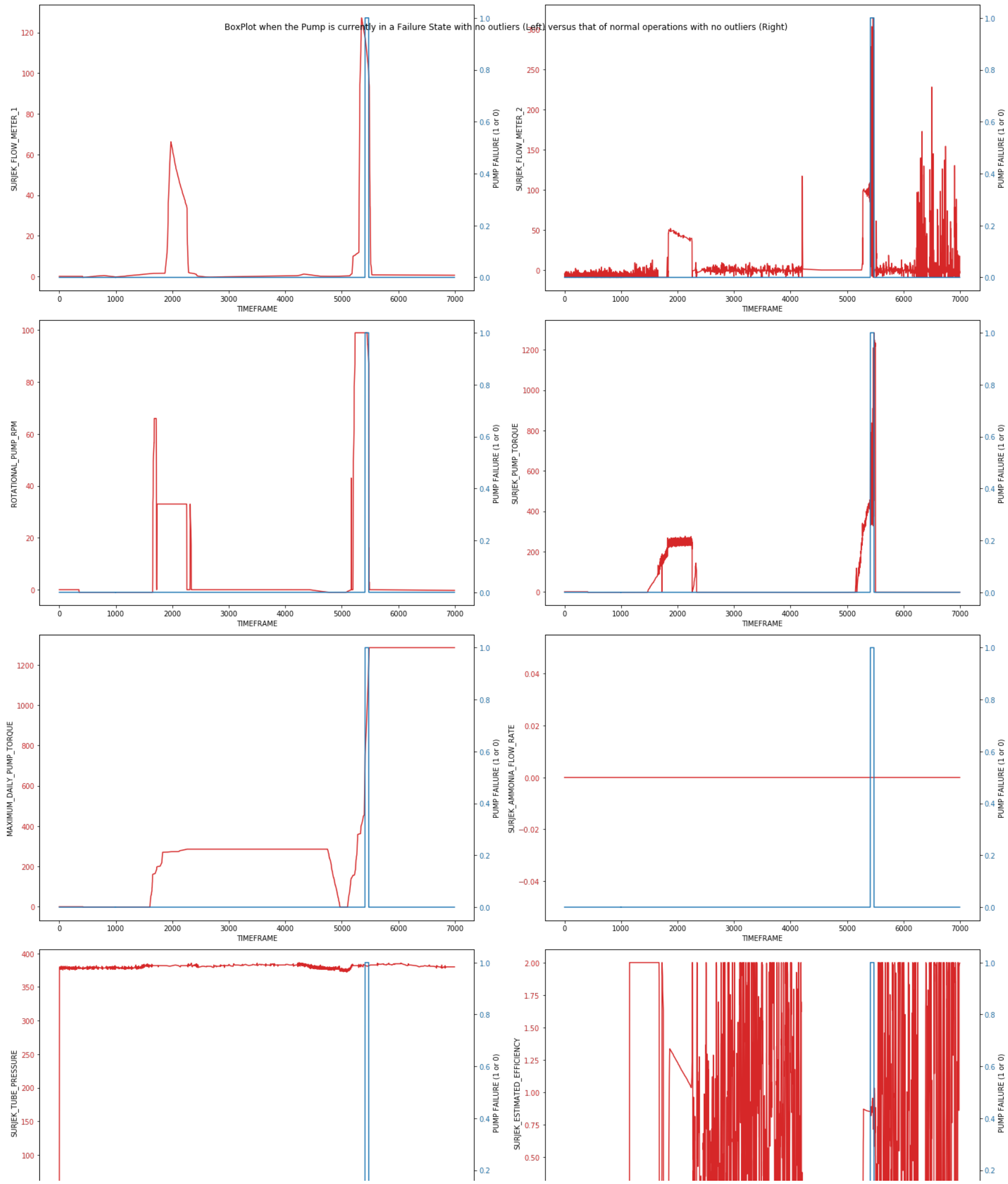
Note: Please ensure that the dataframe you are plotting contains all the outliers and that the Pump Failure Behaviour includes both the 0 and 1 State.

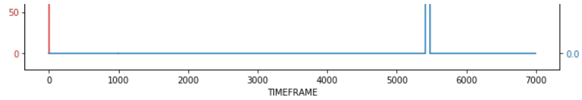
Please put your code here

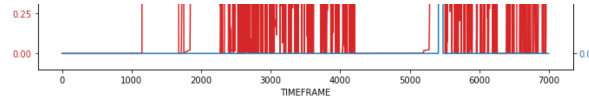
In [19]:

```
merge2 = merge.copy()
merge2.reset_index(drop = True, inplace = True)
#Below is the first part of the code
filt = ['SURJEK_FLOW_METER_1', 'SURJEK_FLOW_METER_2', 'ROTATIONAL_
       'SURJEK_PUMP_TORQUE', 'MAXIMUM_DAILY_PUMP_TORQUE',
       'SURJEK_AMMONIA_FLOW_RATE', 'SURJEK_TUBE_PRESSURE',
       'SURJEK_ESTIMATED_EFFICIENCY']
filt2 = ['PUMP FAILURE (1 or 0)']
colList = merge2[filt].columns

#---write your code below-------
f, axes = plt.subplots(4, 2, figsize = (20,25))
axes = axes.flatten()
f.suptitle("BoxPlot when the Pump is currently in a Failure State

for i,name in enumerate(filt):

    ax1 = axes[i]
    color = 'tab:red'
    ax1.set_xlabel('TIMEFRAME')
    ax1.set_ylabel(name)
```

```
    ax1.plot(merge2[name], color=color)
    ax1.tick_params(axis='y', labelcolor=color)
    ax2 = ax1.twinx()
    color = 'tab:blue'
    ax2.set_ylabel('PUMP FAILURE (1 or 0)')
    ax2.plot(merge2['PUMP FAILURE (1 or 0)'], color=color)
    ax2.tick_params(axis='y', labelcolor=color)
    f.tight_layout()


#---To Here------
plt.show()

```







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”.

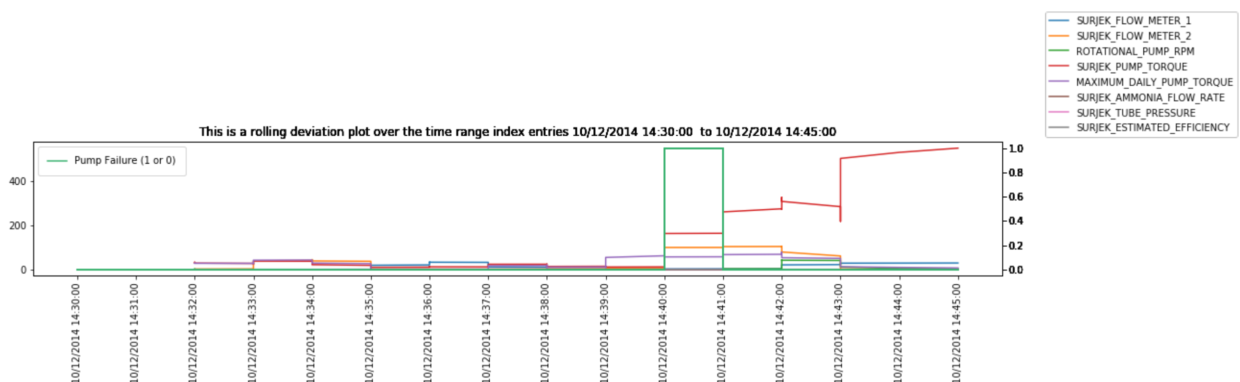
Open-ended Question: Do any particular variables seem to move in relation to the failure event?

Please put your code here

In [20]:

```
#Below is the first part of the code
from datetime import datetime
dataframe_1 = pd.read_csv("Desalination_Unit_File_001.csv", header=1
dataframe_2 = pd.read_excel("Desalination_Unit_File_002.xlsx", heade
dataframe_3 = pd.read_excel("Desalination_Unit_File_003.xlsx", heade
combine =[dataframe_1, dataframe_2, dataframe_3]
dataframe = pd.concat(combine)
dataframe['TIMEFRAME'] = pd.to_datetime(dataframe['TIMEFRAME']).appl
filt = ['SURJEK_FLOW_METER_1', 'SURJEK_FLOW_METER_2', 'ROTATIONAL_PU
       'SURJEK_PUMP_TORQUE', 'MAXIMUM_DAILY_PUMP_TORQUE',
       'SURJEK_AMMONIA_FLOW_RATE', 'SURJEK_TUBE_PRESSURE',
       'SURJEK_ESTIMATED_EFFICIENCY','PUMP FAILURE (1 or 0)', 'TIMEF
filt2 = ['PUMP FAILURE (1 or 0)']
filt3 = ['SURJEK_FLOW_METER_1', 'SURJEK_FLOW_METER_2', 'ROTATIONAL_F
       'SURJEK_PUMP_TORQUE', 'MAXIMUM_DAILY_PUMP_TORQUE',
       'SURJEK_AMMONIA_FLOW_RATE', 'SURJEK_TUBE_PRESSURE',
       'SURJEK_ESTIMATED_EFFICIENCY']
```

```
colList = dataframe[filt].columns
mpl.rcParams['figure.figsize'] = (15,4)
dataframe.set_index('TIMEFRAME', inplace=True)
#----write your code below-------

dataframe = dataframe[(dataframe.index >= "10/12/2014 14:30:00")&(da
rollingDF = dataframe.rolling(30).std()

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

colList = rollingDF[filt3].columns
rollingDF['PUMP FAILURE (1 or 0)'] = dataframe['PUMP FAILURE (1 or 0

fig = plt.figure()
ax = plt.axes()
date_form = DateFormatter("%d/%m/%Y %H:%M:%S")
ax.xaxis.set_major_formatter(date_form)
#Loop through the Plot
for i in colList:
    ax.plot(rollingDF.index, rollingDF[i], label=i)
    ax2 = ax.twinx()
    ax2.plot(dataframe[filt2], 'mediumseagreen', label='Pump Failure

    ax.xaxis.set_tick_params(rotation=90)

    plt.tight_layout()
    minTime = rollingDF.index.min()
    maxTime= rollingDF.index.max()
    plt.title("This is a rolling deviation plot over the time range

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



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).

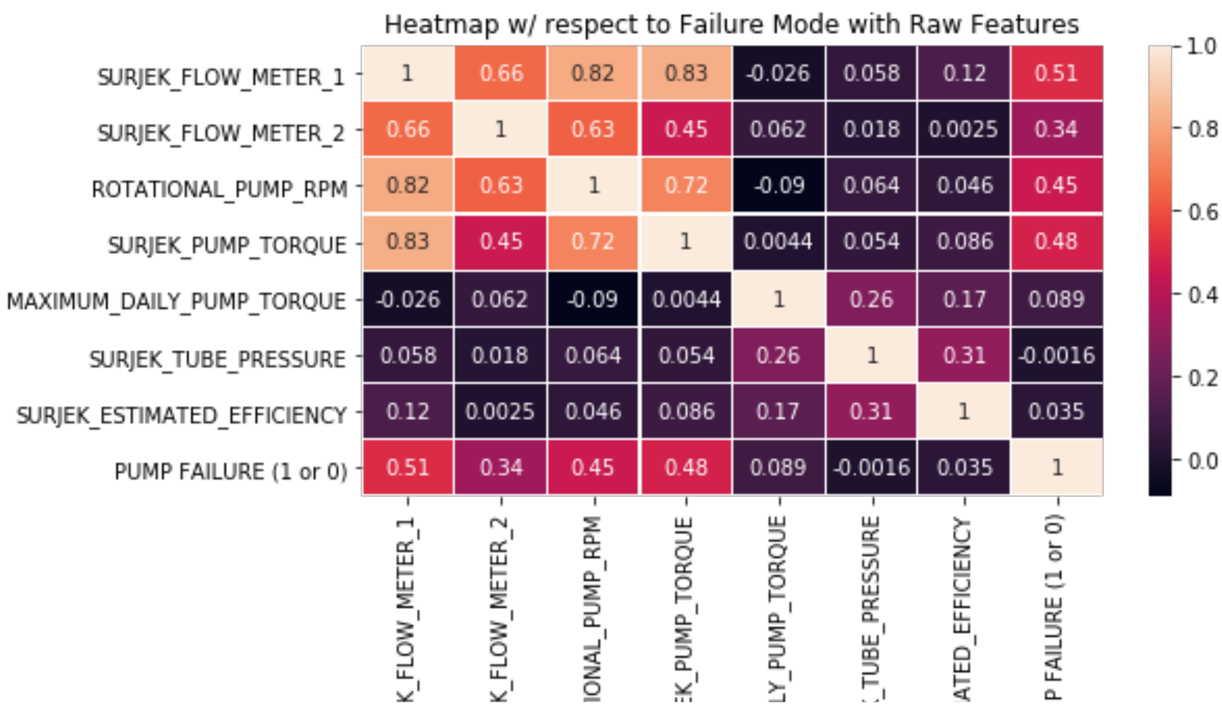
Note: We have provided the filter list and created the dataframe for you.

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

Please put your code here

In [21]:

```
#Below is the first part of the code
from datetime import datetime
dataframe = pd.concat(combine)
dataframe['TIMEFRAME'] = pd.to_datetime(dataframe['TIMEFRAME'], fo
dataframe.set_index('TIMEFRAME', inplace=True)

filt = ['SURJEK_FLOW_METER_1', 'SURJEK_FLOW_METER_2', 'ROTATIONAL_
        '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--------

corr = dataframe.corr()
fig, ax = plt.subplots(figsize=(8,4))
sbn.heatmap(corr, annot=True,linewidths=.1,
            xticklabels=corr.columns.values,
            yticklabels=corr.columns.values,
            ax=ax)
plt.title("Heatmap w/ respect to Failure Mode with Raw Features")
plt.show()
```



Open-ended Question:

Which variables seem to correlate with 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.

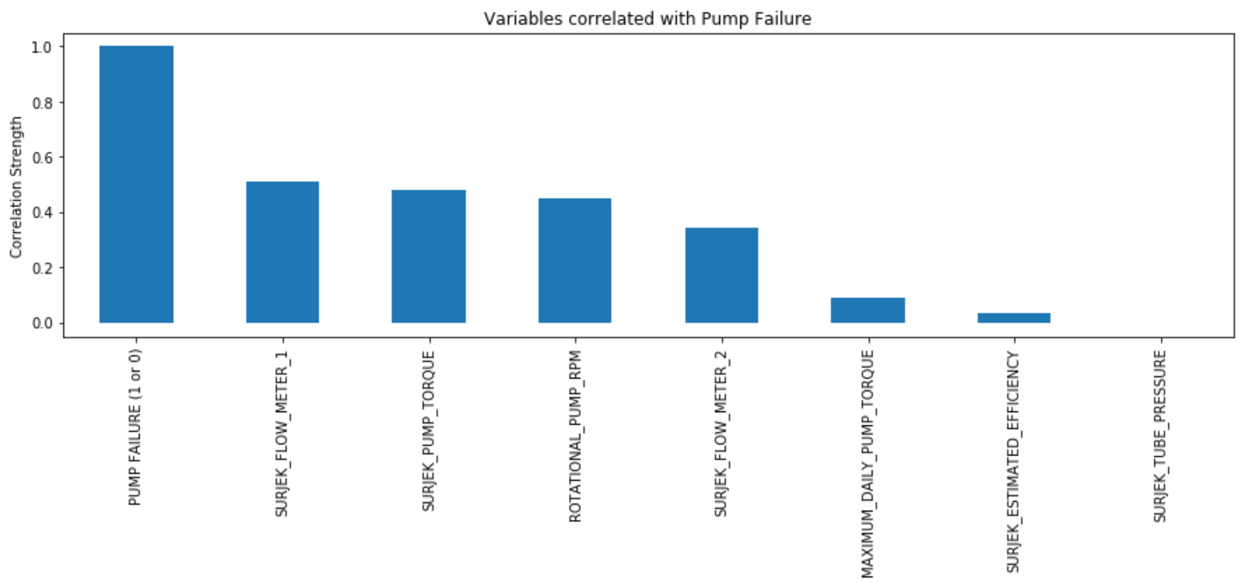
Please put your code here

In [22]:

```
filt2 = ['PUMP FAILURE (1 or 0)']
dataframe2 = dataframe[filt2]
corr2 = dataframe2.corr()
```

In [23]:

```
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.

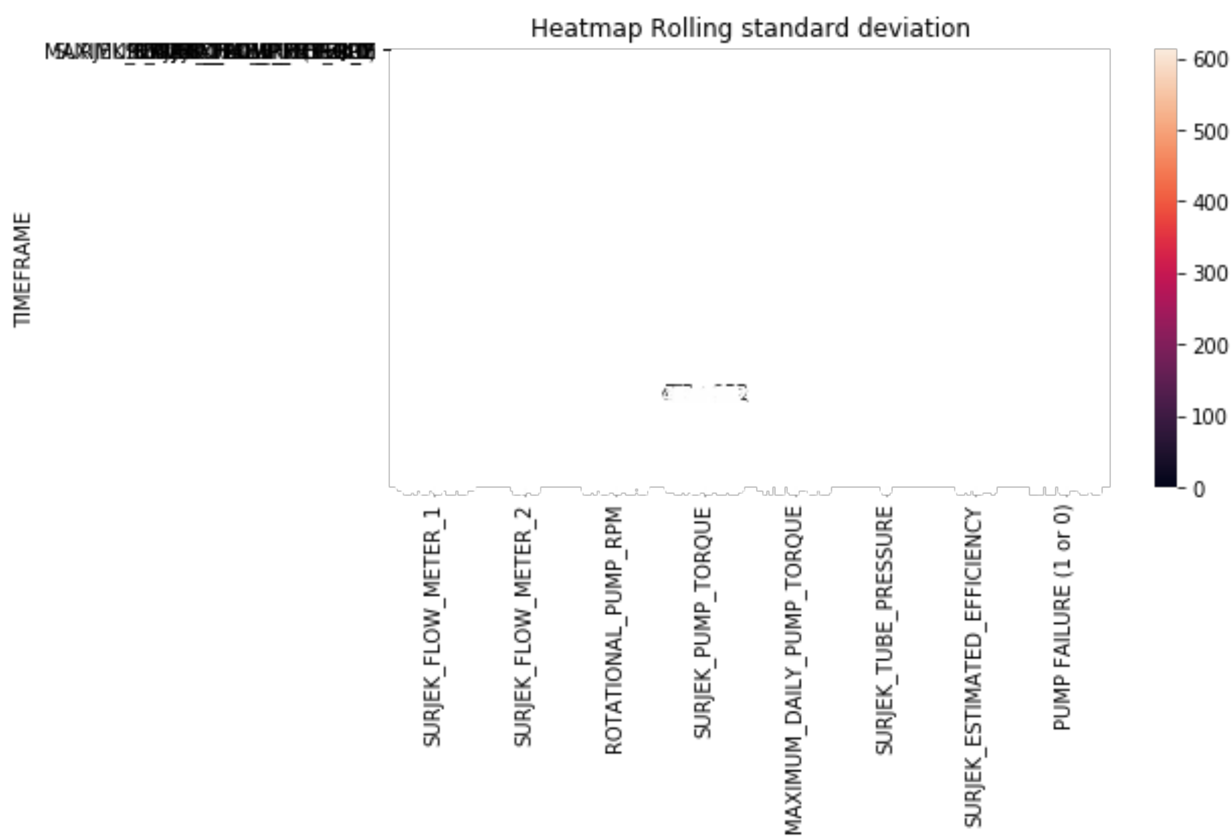
Do any variables stand out? If yes, list these out below your heatmap.

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(combine)
dataframe['TIMEFRAME'] = pd.to_datetime(dataframe['TIMEFRAME'], fo
dataframe.set_index('TIMEFRAME', inplace=True)
filt = ['SURJEK_FLOW_METER_1', 'SURJEK_FLOW_METER_2', 'ROTATIONAL_
       'SURJEK_PUMP_TORQUE', 'MAXIMUM_DAILY_PUMP_TORQUE',
       'SURJEK_TUBE_PRESSURE',
       'SURJEK_ESTIMATED_EFFICIENCY','PUMP FAILURE (1 or 0)']
#----write your code below------
dataframe = dataframe[filt]
dataframe2 = dataframe.rolling(30).std()
fig, ax = plt.subplots(figsize=(8,4))
sbn.heatmap(dataframe2, annot=True,linewidths=.1,
            xticklabels=corr.columns.values,
            yticklabels=corr.columns.values,
            ax=ax)
plt.title("Heatmap Rolling standard deviation")
plt.show()
```



In [31]:

```
print(dataframe)
```

```
                    SURJEK_FLOW_METER_1  SURJEK_FLOW_METER_2  \
TIMEFRAME
NaT                                 NaN                  NaN
NaT                                 NaN                  NaN
```

```
NaT                                      NaN                 NaN
NaT                                      NaN                 NaN
NaT                                      NaN                 NaN
...                                      ...                 ...
2014-12-10 16:52:00                 0.533913            2.900391
2014-12-10 16:52:00                 0.533862           17.468260
2014-12-10 16:52:00                 0.533811            3.603516
2014-12-10 16:52:00                 0.533657           -4.174805
2014-12-10 16:52:00                 0.533504           -1.867676

                     ROTATIONAL_PUMP_RPM  SURJEK_PUMP_TORQUE  \
TIMEFRAME
NaT                                  NaN                 NaN
NaT                                  NaN                 NaN
NaT                                  NaN                 NaN
NaT                                  NaN                 NaN
NaT                                  NaN                 NaN
...                                  ...                 ...
2014-12-10 16:52:00            -0.272157           -1.898485
2014-12-10 16:52:00            -0.272227           -1.898529
2014-12-10 16:52:00            -0.272296           -1.898573
2014-12-10 16:52:00            -0.272505           -1.898704
2014-12-10 16:52:00            -0.272714           -1.898834

                     MAXIMUM_DAILY_PUMP_TORQUE  SURJEK_TUBE_PRESSURE
\
TIMEFRAME
NaT                                        NaN                   NaN
NaT                                        NaN                   NaN
NaT                                        NaN                   NaN
NaT                                        NaN                   NaN
NaT                                        NaN                   NaN
...                                        ...                   ...
2014-12-10 16:52:00                   1284.838              379.9438
2014-12-10 16:52:00                   1284.838              379.9438
2014-12-10 16:52:00                   1284.838              379.9438
2014-12-10 16:52:00                   1284.838              379.9438
2014-12-10 16:52:00                   1284.838              379.9438

                     SURJEK_ESTIMATED_EFFICIENCY  PUMP FAILURE (1 or
0)
TIMEFRAME
NaT                                          NaN                    N
aN
NaT                                          NaN                    N
aN
NaT                                          NaN                    N
aN
NaT                                          NaN                    N
aN
NaT                                          NaN                    N
aN
...                                          ...                    .
```

```
..
2014-12-10 16:52:00                     1.992867                    0
.0
2014-12-10 16:52:00                     1.993286                    0
.0
2014-12-10 16:52:00                     1.993706                    0
.0
2014-12-10 16:52:00                     1.994967                    0
.0
2014-12-10 16:52:00                     1.996225                    0
.0

[6997 rows x 8 columns]
```

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?

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

Please put your code here

In [32]:

```
dataframe_two = pd.concat(combine)
dependentVar = dataframe_two['PUMP FAILURE (1 or 0)']
filt = ['SURJEK_FLOW_METER_1', 'SURJEK_FLOW_METER_2', 'ROTATIONAL_
        'SURJEK_PUMP_TORQUE', 'MAXIMUM_DAILY_PUMP_TORQUE',
        'SURJEK_AMMONIA_FLOW_RATE', 'SURJEK_TUBE_PRESSURE',
        'SURJEK_ESTIMATED_EFFICIENCY']
```

```
           'SURJEK_ESTIMATED_EFFICIENCY']
#----write your code below------
dataframe3 = dataframe_two[filt]
results = sm.OLS(dependentVar.dropna(),dataframe3.dropna()).fit()
print(results.summary())
```

```
                                  OLS Regression Results
======================================================================================
=====================
Dep. Variable:     PUMP FAILURE (1 or 0)   R-squared (uncentered):
0.296

Model:                               OLS   Adj. R-squared (uncentered
):                0.295
Method:                    Least Squares   F-statistic:
299.8
Date:                   Sat, 16 May 2020   Prob (F-statistic):
0.00
Time:                           20:16:25   Log-Likelihood:
4639.6
No. Observations:                   5000   AIC:
-9265.
Df Residuals:                       4993   BIC:
-9220.
Df Model:                              7
Covariance Type:               nonrobust
======================================================================================
==========================
                                  coef    std err          t      P>|
t|       [0.025      0.975]
--------------------------------------------------------------------------------------
--------------------------
SURJEK_FLOW_METER_1             0.0015      0.000     10.759      0.0
00        0.001       0.002
SURJEK_FLOW_METER_2         -1.604e-05   7.26e-05     -0.221      0.8
25       -0.000       0.000
ROTATIONAL_PUMP_RPM             0.0005      0.000      4.553      0.0
00        0.000       0.001
SURJEK_PUMP_TORQUE              0.0001   1.84e-05      7.064      0.0
00    9.39e-05       0.000
MAXIMUM_DAILY_PUMP_TORQUE    2.698e-05   2.96e-06      9.105      0.0
00    2.12e-05    3.28e-05
SURJEK_AMMONIA_FLOW_RATE     7.463e-18   2.32e-18      3.220      0.0
01    2.92e-18     1.2e-17
SURJEK_TUBE_PRESSURE        -4.753e-05   6.36e-06     -7.479      0.0
00       -6e-05   -3.51e-05
SURJEK_ESTIMATED_EFFICIENCY    -0.0065      0.002     -3.213      0.0
01       -0.010      -0.003
======================================================================================
=========
Omnibus:                    4812.180   Durbin-Watson:
0.046
Prob(Omnibus):                 0.000   Jarque-Bera (JB):                 3
05241.033
Skew:                          4.559   Prob(JB):
```

```
0.00
Kurtosis:                      40.176   Cond. No.
4.14e+20
======================================================================
=========

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors i
s correctly specified.
[2] The smallest eigenvalue is 1.93e-32. This might indicate that the

re are
strong multicollinearity problems or that the design matrix is singul
```

In [36]:

```python
dataframe_two = pd.concat(combine)
#dataframe_two.set_index("TIMEFRAME",inplace=True)
dependentVar = dataframe_two['PUMP FAILURE (1 or 0)']
filt = ['SURJEK_FLOW_METER_1', 'SURJEK_FLOW_METER_2', 'ROTATIONAL_
       '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.rename(columns = {'PUMP FAILURE (1 or 0)':'Pump_Fail
regression_eqn = ols(formula = "Pump_Failure ~ SURJEK_FLOW_METER_1
print(regression_eqn.summary())
```

```
                            OLS Regression Results
======================================================================
=========
Dep. Variable:           Pump_Failure   R-squared:
0.643
Model:                            OLS   Adj. R-squared:
0.643
Method:                 Least Squares   F-statistic:
1797.
Date:                Sat, 16 May 2020   Prob (F-statistic):
0.00
Time:                        20:38:53   Log-Likelihood:
16504.
No. Observations:                6997   AIC:                          -
3.299e+04
Df Residuals:                    6989   BIC:                          -
3.294e+04
Df Model:                           7
Covariance Type:            nonrobust
======================================================================
==========================
                                  coef    std err          t      P>|
```

```
t|      [0.025      0.975]
---------------------------------------------------------------------
--------------------------
Intercept                         -0.0028      0.000     -6.329      0.0
00      -0.004      -0.002
SURJEK_FLOW_METER_1                0.0002      0.000      1.228      0.2
19      -0.000       0.000
SURJEK_FLOW_METER_2            -4.905e-05   2.56e-05     -1.917      0.0
55   -9.92e-05    1.11e-06
ROTATIONAL_PUMP_RPM                0.0034   9.86e-05     34.005      0.0
00       0.003       0.004
SURJEK_PUMP_TORQUE             -1.091e-05   1.01e-05     -1.076      0.2
82   -3.08e-05    8.96e-06
MAXIMUM_DAILY_PUMP_TORQUE          0.0027   3.61e-05     73.794      0.0
00       0.003       0.003
SURJEK_AMMONIA_FLOW_RATE       -2.307e-19    1.9e-20    -12.119      0.0
00   -2.68e-19   -1.93e-19
SURJEK_TUBE_PRESSURE              -0.0085      0.001    -13.088      0.0
00      -0.010      -0.007
SURJEK_ESTIMATED_EFFICIENCY        0.0077      0.001      8.338      0.0
00       0.006       0.010
=====================================================================
=========
Omnibus:                     2164.542   Durbin-Watson:
0.043
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            4
16482.746
Skew:                           0.097   Prob(JB):
0.00
Kurtosis:                      40.796   Cond. No.
6.92e+18
=====================================================================
=========

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors i
s correctly specified.
[2] The smallest eigenvalue is 2.66e-31. This might indicate that the
re are
strong multicollinearity problems or that the design matrix is singul
ar.
```

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.

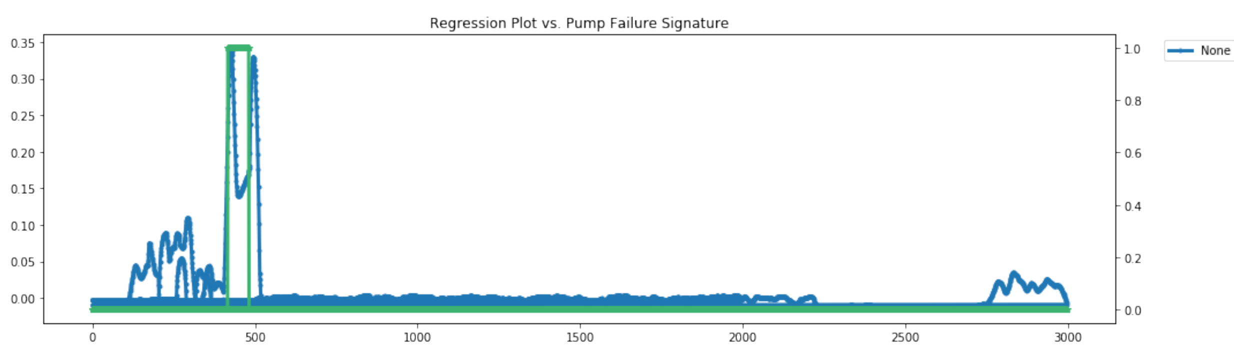
ii) Plot the rolling linear regression equation against the attribute 'PUMP FAILURE (1 or 0)'

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 [38]:
```
#Below is the first part of the code
mpl.rcParams['figure.figsize'] = (15,4)
#----write your code below------

X = dataframe_two.drop(['PumpFailure'],axis=1)
X = sm.add_constant(X)

ax = regression_eqn.predict(X).plot(linewidth=3, marker='.')
ax2 = ax.twinx()
ax2.plot(dataframe_two.PumpFailure, 'mediumseagreen', linewidth=3,
ax.legend(bbox_to_anchor=(1.04,1), loc="upper left")
plt.tight_layout()
plt.title("Regression Plot vs. Pump Failure Signature")
```

Out[38]:
```
Text(0.5, 1, 'Regression Plot vs. Pump Failure Signature')
```



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!