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.pyplot as plt
import matplotlib as mpl
import seaborn as sns
import pandas as pd
import numpy as np
import statsmodels.api as sm
from datetime import datetime
```

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 [3]:
```
dataframe1 = pd.read_csv('Desalination_Unit_File_001.csv', header = 1)
dataframe2 = pd.read_excel('Desalination_Unit_File_002.xlsx', header = 1)
dataframe3 = pd.read_excel('Desalination_Unit_File_003.xlsx', header = 1)
combine = pd.concat([dataframe1, dataframe2, dataframe3])
combine['TIMEFRAME'] = pd.to_datetime(combine['TIMEFRAME']).apply(lambda x: x.
strftime('%d/%m/%Y %H:%M:%S')if not pd.isnull(x) else '')
combine['PUMP FAILURE (1 or 0)'].fillna(0,inplace=True)
combine.dropna(inplace=True)
combine.reset_index(drop=True, inplace=True)
print(combine.describe())
print(combine.info())
```

```
       SURJEK_FLOW_METER_1  SURJEK_FLOW_METER_2  ROTATIONAL_PUMP_RPM  \
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

       SURJEK_PUMP_TORQUE  MAXIMUM_DAILY_PUMP_TORQUE  \
count         6998.000000                6998.000000
mean            39.091614                 427.295713
std            124.174236                 473.250507
min             -2.436085                  -2.278918
25%             -2.030993                   9.177878
50%             -1.896835                 285.493400
75%             -1.680961                 285.493400
max           1284.681000                1284.838000

       SURJEK_AMMONIA_FLOW_RATE  SURJEK_TUBE_PRESSURE  \
count                    6998.0           6998.000000
mean                        0.0            380.696815
std                         0.0              6.817019
min                         0.0              0.000000
25%                         0.0            379.028300
50%                         0.0            381.317366
75%                         0.0            382.690400
max                         0.0            386.352500

       SURJEK_ESTIMATED_EFFICIENCY  PUMP FAILURE (1 or 0)
count                  6998.000000            6998.000000
mean                      0.646718               0.009288
std                       0.755587               0.095934
min                       0.000000               0.000000
25%                       0.000000               0.000000
50%                       0.204052               0.000000
75%                       1.240724               0.000000
max                       2.000000               1.000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6998 entries, 0 to 6997
Data columns (total 10 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   float64
 6   SURJEK_TUBE_PRESSURE         6998 non-null   float64
 7   SURJEK_ESTIMATED_EFFICIENCY  6998 non-null   float64
 8   PUMP FAILURE (1 or 0)        6998 non-null   float64
 9   TIMEFRAME                    6998 non-null   object
dtypes: float64(9), object(1)
memory usage: 546.8+ KB
None
```

Step 3: Create a Boxplot

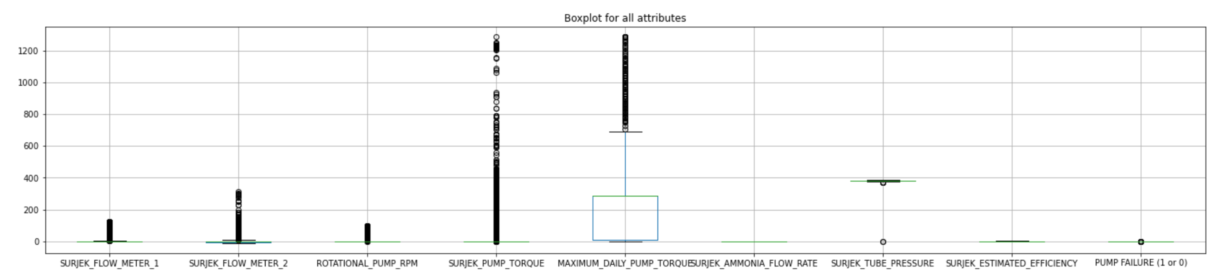
When you look at your dataframe, you should now be able to see the upper and lower quartiles for each row of data. You should now also have a rough sense of the number of entires in each dataset. However, just as you learned when using Excel, creating a visualization of the data using Python is often 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 [5]:
```
combine.boxplot(figsize=(25,5))
plt.title("Boxplot for all attributes")
```

Out[5]: Text(0.5, 1.0, 'Boxplot for all attributes')



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.

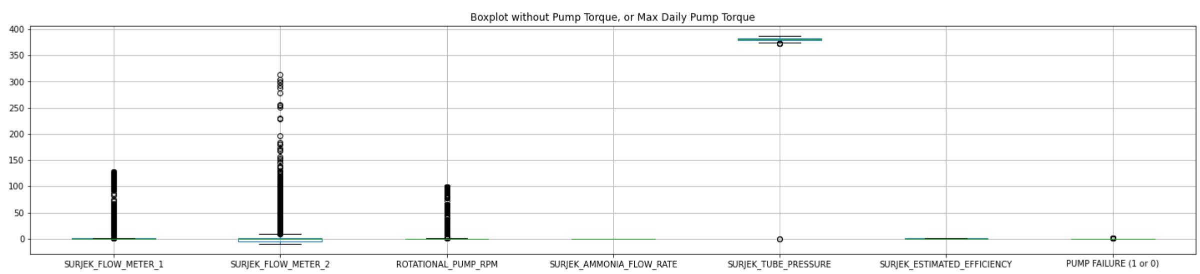
Open-ended question:

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

Please put your code here

In [6]:
```
#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)']
mpl.rcParams['figure.figsize'] = (25,5)
#--write your code below------
combine.boxplot(column=filt)
plt.title('Boxplot without Pump Torque, or Max Daily Pump Torque')
```

Out[6]:
```
Text(0.5, 1.0, 'Boxplot without Pump Torque, or Max Daily Pump Torque')
```



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.

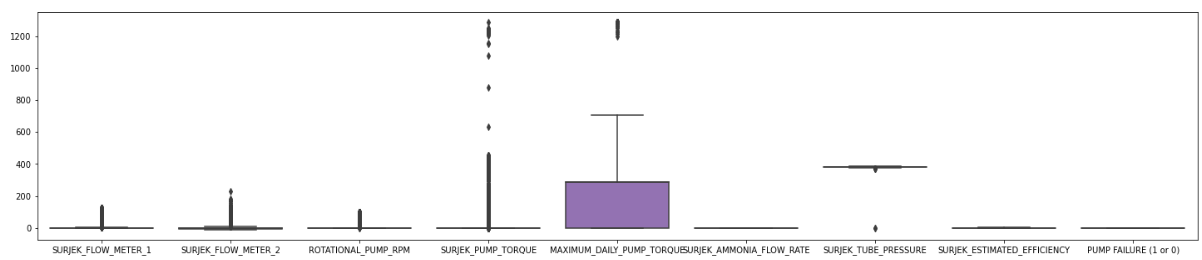
Open-ended Question:

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

Please put your code here

In [7]:
```
combine_1 = combine[combine['PUMP FAILURE (1 or 0)']==0]
sns.boxplot(data=combine_1)
```

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x1d79be24790>



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 [9]:
```
Q1=combine.quantile(0.25)
Q3=combine.quantile(0.75)
IQR=Q3-Q1
lower_range=Q1-(1.5*IQR)
upper_range=Q3+(1.5*IQR)
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 [14]:
```
#Below is the first part of the code
dataframe = combine
dataframe = dataframe.dropna()
#---write your code below----------------
normal = len(dataframe)
normal_wo = len(dataframe[~((dataframe < (Q1 - 1.5 * IQR)) |(dataframe > (Q3 +
1.5 * IQR))).any(axis=1)])
normal_prop = normal_wo / normal

#We have provided the print line, you need to provide the calculation after th
e quoted text:
print ("When we have not removed any outliers from the dataset, we have " + st
r(normal) + " entries")
print ("The proportion of outliers which exist when compared to the dataframe
 are: " + str(normal_prop))
```

```
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

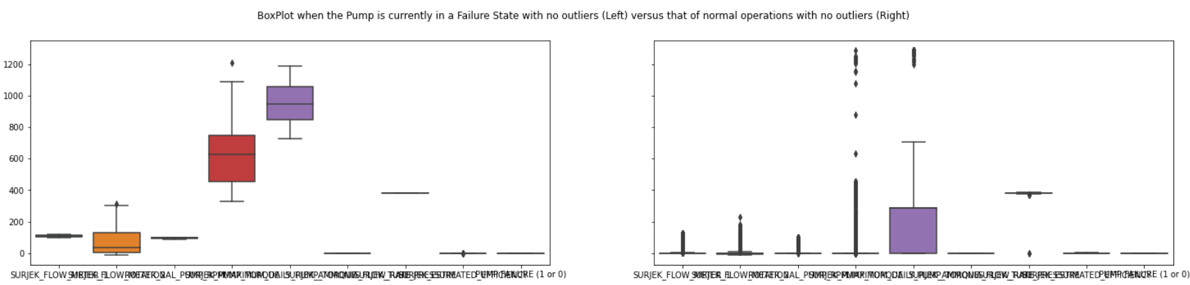
Note 1: Removing outliers is very situational and specific. Outliers can skew the dataset unfavourably; however, if you are doing a failure analysis, it is likely those outliers actually contain valuable insights you will want to keep as they represent a deviation from the norm that you'll need to understand.

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 [15]:
```python
#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 no outliers (Left) versus that of normal operations with no outliers (Right)")
mpl.rcParams['figure.figsize'] = (15,5)
#---write your code below-------------
combine_1 = dataframe[dataframe['PUMP FAILURE (1 or 0)']==1]
sns.boxplot(data=combine_1, ax = axes[0])
combine_2 = dataframe[dataframe['PUMP FAILURE (1 or 0)']==0]
sns.boxplot(data=combine_2, ax = axes[1])
```

Out[15]: `<matplotlib.axes._subplots.AxesSubplot at 0x1d79cfb1100>`



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.

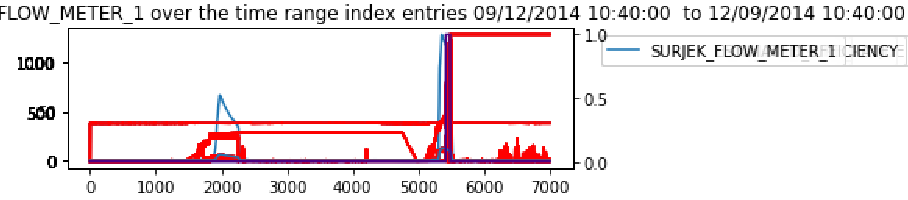
Check out this link to learn how to do this: [https://matplotlib.org/gallery/api/two_scales.html (https://matplotlib.org/gallery/api/two_scales.html)](https://matplotlib.org/gallery/api/two_scales.html)

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.

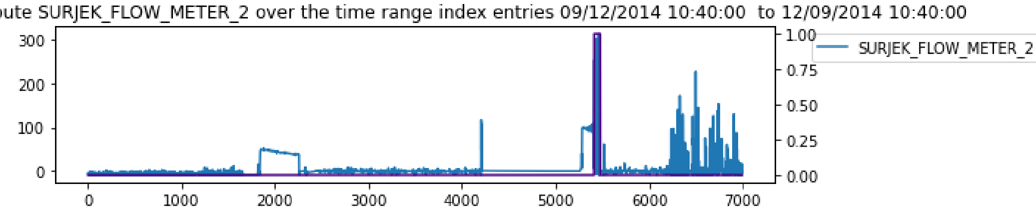
Please put your code here

In [16]:
```
#Below is the first part of the code
filt = ['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']
filt2 = ['PUMP FAILURE (1 or 0)']
colList = dataframe[filt].columns
mpl.rcParams['figure.figsize'] = (10,2)
#---write your code below-------
fig, ax1 = plt.subplots()

for i in filt:
    columns = combine[filt]
    ax = combine[i].plot()
    ax1 = ax.twinx()
    ax1.plot(columns, 'r')
    ax.legend(bbox_to_anchor=(1.04,1), loc="upper left")
    plt.tight_layout()

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

    minTime = combine['TIMEFRAME'].min()
    maxTime= combine['TIMEFRAME'].max()
    plt.tight_layout()
    plt.title("This is for attribute " + i + " over the time range index entri
es " + str(minTime) + " " +  " to " + str(maxTime))
#---To Here------
    fig.tight_layout()
    plt.show()

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

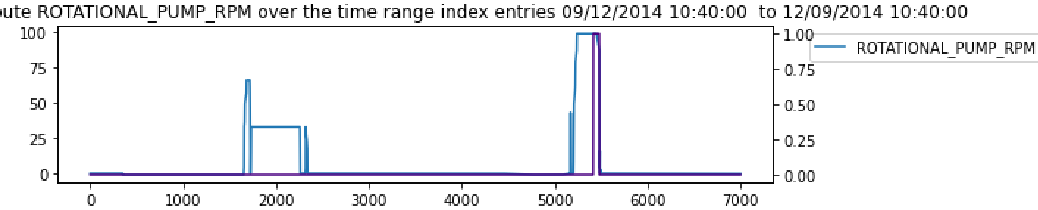
This is for attribute SURJEK_FLOW_METER_1 over the time range index entries 09/12/2014 10:40:00 to 12/09/2014 10:40:00



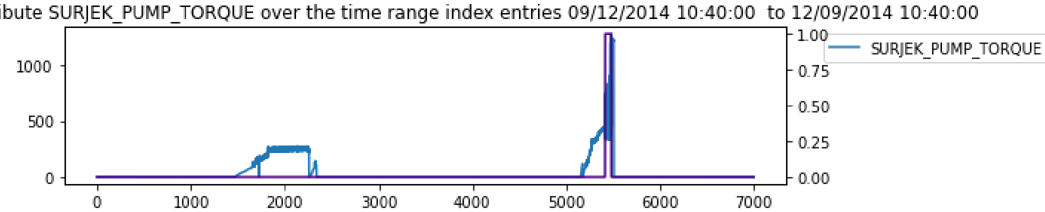
This is for attribute SURJEK_FLOW_METER_2 over the time range index entries 09/12/2014 10:40:00 to 12/09/2014 10:40:00



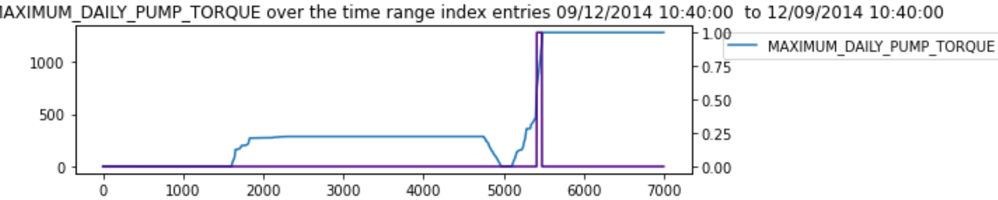
This is for attribute ROTATIONAL_PUMP_RPM over the time range index entries 09/12/2014 10:40:00 to 12/09/2014 10:40:00



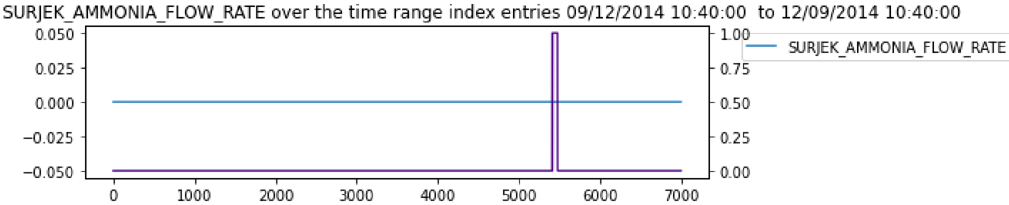
This is for attribute SURJEK_PUMP_TORQUE over the time range index entries 09/12/2014 10:40:00 to 12/09/2014 10:40:00



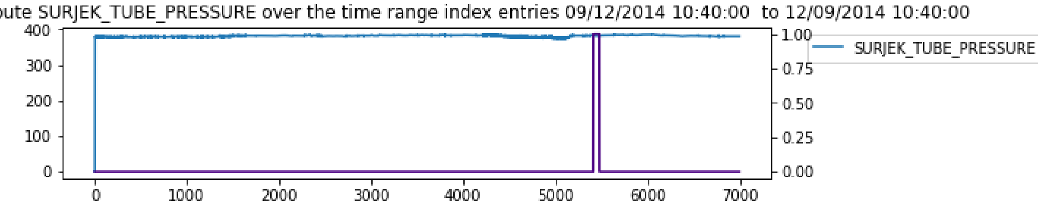
This is for attribute MAXIMUM_DAILY_PUMP_TORQUE over the time range index entries 09/12/2014 10:40:00 to 12/09/2014 10:40:00



This is for attribute SURJEK_AMMONIA_FLOW_RATE over the time range index entries 09/12/2014 10:40:00 to 12/09/2014 10:40:00

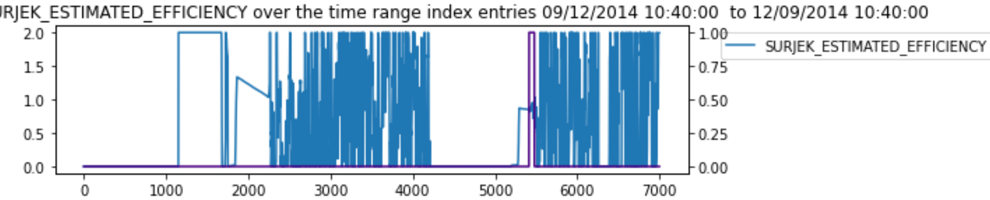


This is for attribute SURJEK_TUBE_PRESSURE over the time range index entries 09/12/2014 10:40:00 to 12/09/2014 10:40:00




```
<ipython-input-16-a44bc59c8004>:32: UserWarning: Tight layout not applied. Th
e left and right margins cannot be made large enough to accommodate all axes 
decorations.
  fig.tight_layout()
```

This is for attribute SURJEK_ESTIMATED_EFFICIENCY over the time range index entries 09/12/2014 10:40:00 to 12/09/2014 10:40:00



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'.

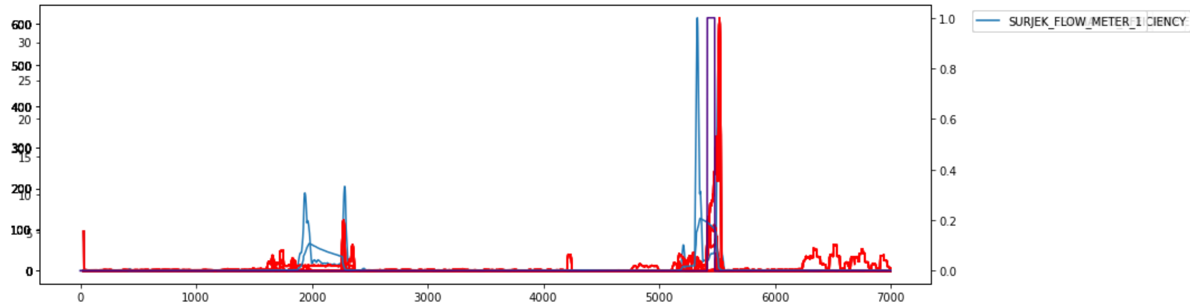
ii) Re-plot all variables for the time period 10/12/2014 14:40 to 10/12/2014 14:45, focusing specifically on the first Pump “Failure”.

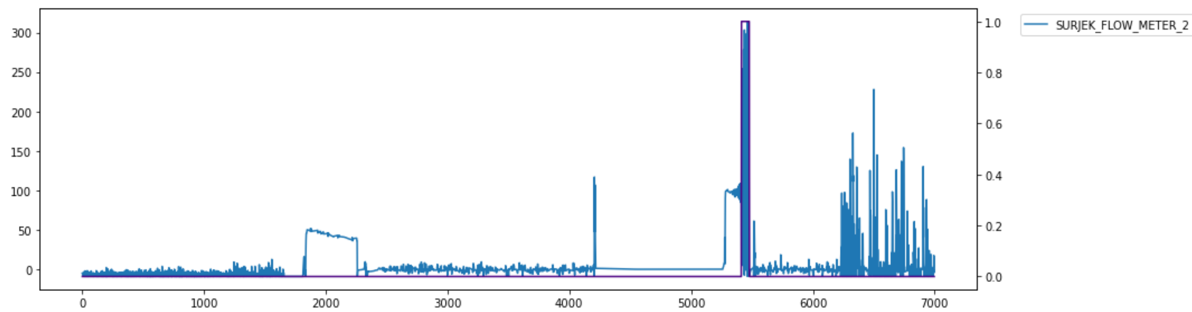
Open-ended Question: Do any particular variables seem to move in relation to the failure event?

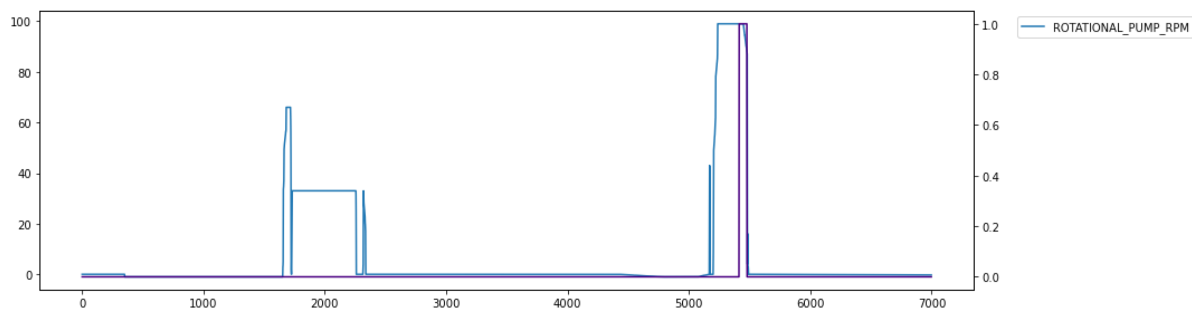
Please put your code here

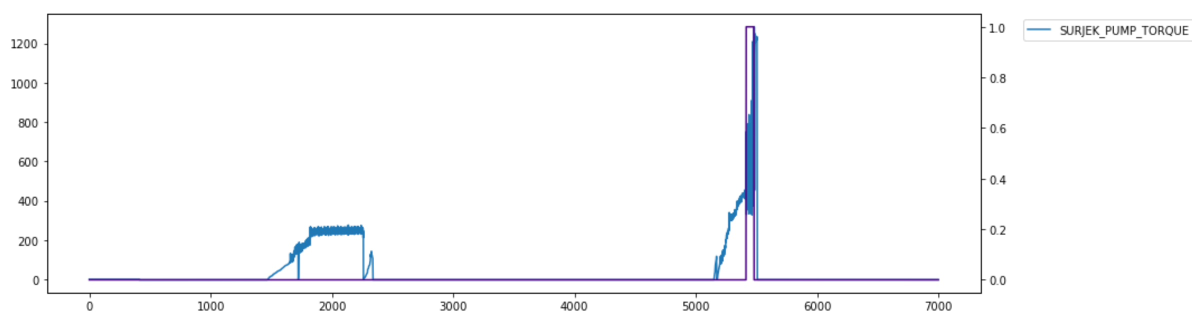
In [25]:

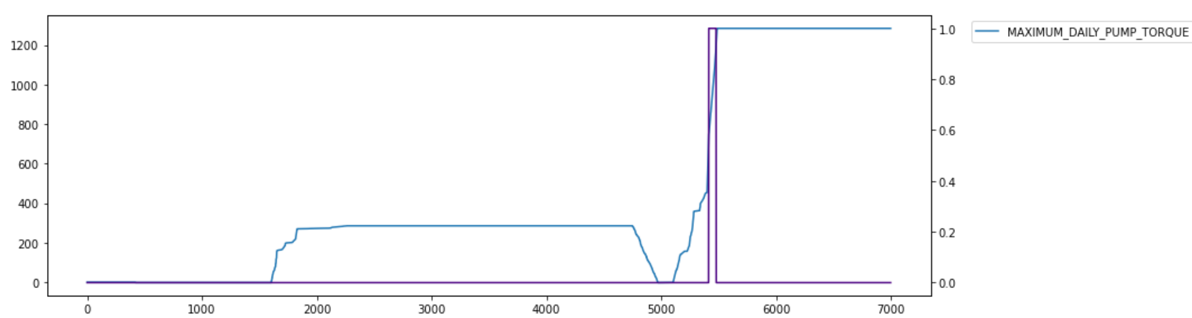
```
#Below is the first part of the code
#dataframe = pd.concat([dataframe1, dataframe2, dataframe3])
#dataframe['TIMEFRAME'] = pd.to_datetime(dataframe['TIMEFRAME']).apply(lambda
 x: x.strftime('%d/%m/%Y %H:%M:%S')if not pd.isnull(x) else '')
filt = ['SURJEK_FLOW_METER_1', 'SURJEK_FLOW_METER_2', 'ROTATIONAL_PUMP_RPM',
       'SURJEK_PUMP_TORQUE', 'MAXIMUM_DAILY_PUMP_TORQUE',
       'SURJEK_AMMONIA_FLOW_RATE', 'SURJEK_TUBE_PRESSURE',
       'SURJEK_ESTIMATED_EFFICIENCY','PUMP FAILURE (1 or 0)', 'TIMEFRAME']
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']
combine_std = combine[filt3].rolling(30).std()
combine_std = combine_std.dropna()
combine_std = combine_std.join(combine[['PUMP FAILURE (1 or 0)', 'TIMEFRAME'
]], how="inner")
dataframe = combine_std[(combine_std['TIMEFRAME'] >= "12/10/2014 14:30:00")&(c
ombine_std['TIMEFRAME'] <="12/10/2014 14:45:00")]
colList = combine[filt3].columns
mpl.rcParams['figure.figsize'] = (15,4)

#Loop through the Plot

fig, ax1 = plt.subplots()

for i in filt3:
    columns = combine_std[filt3]
    ax = combine_std[i].plot()
    ax1 = ax.twinx()
    ax1.plot(columns, 'r')
    ax.legend(bbox_to_anchor=(1.04,1), loc="upper left")
    plt.tight_layout()

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

    minTime = dataframe.min()
    maxTime = dataframe.max()
    plt.tight_layout()
#---To Here------
    fig.tight_layout()
    plt.show()
```

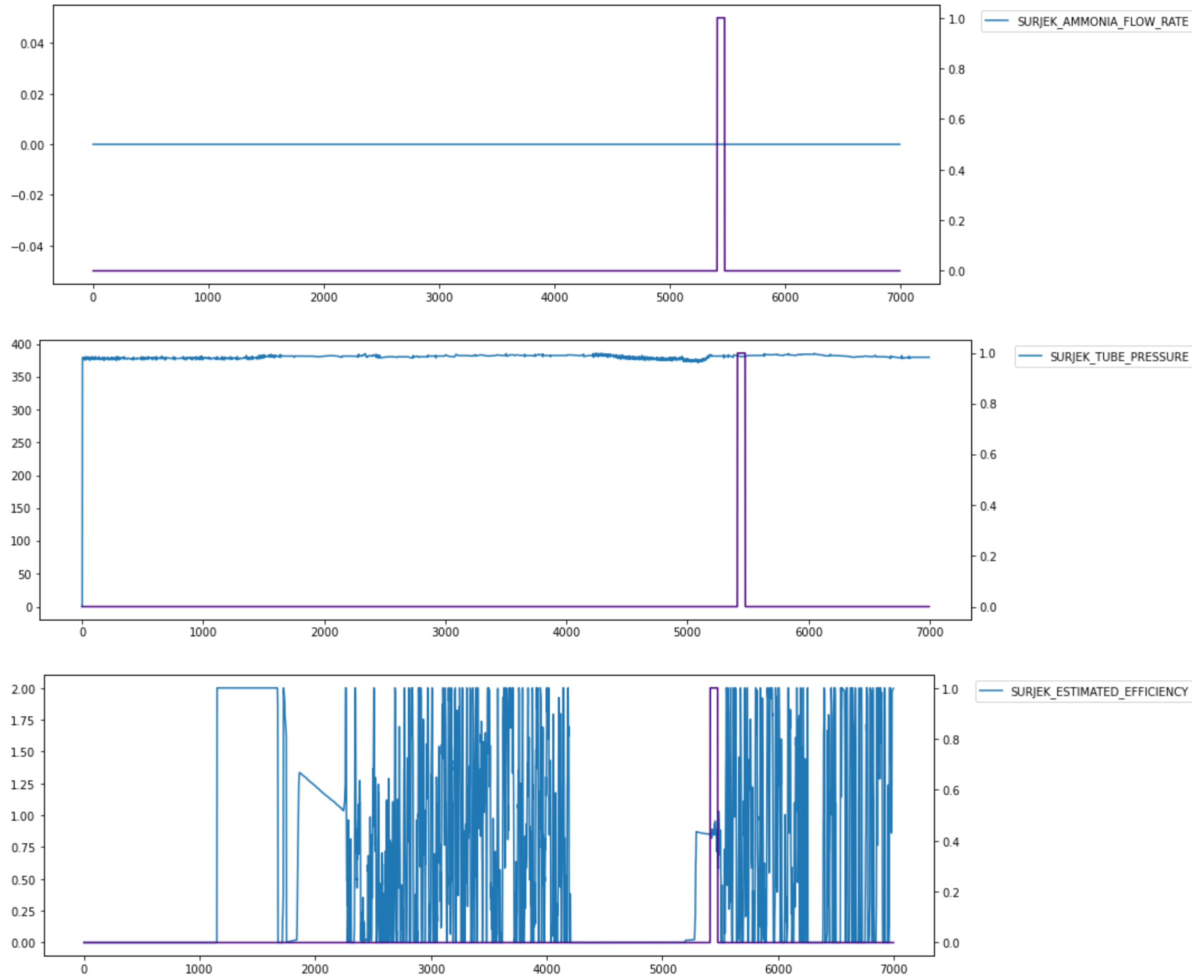

SURJEK_FLOW_METER_1


SURJEK_FLOW_METER_2


ROTATIONAL_PUMP_RPM


SURJEK_PUMP_TORQUE


MAXIMUM_DAILY_PUMP_TORQUE



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 [47]:
```
#Below is the first part of the code
from datetime import datetime
dataframe = combine
#dataframe['TIMEFRAME'] = pd.to_datetime(dataframe['TIMEFRAME'], format="%d/%m/%Y %H:%M:%S", infer_datetime_format=True )
filt = ['SURJEK_FLOW_METER_1', 'SURJEK_FLOW_METER_2', 'ROTATIONAL_PUMP_RPM',
       'SURJEK_PUMP_TORQUE', 'MAXIMUM_DAILY_PUMP_TORQUE',
       'SURJEK_TUBE_PRESSURE',
       'SURJEK_ESTIMATED_EFFICIENCY','PUMP FAILURE (1 or 0)']
dataframe = dataframe[filt]
corrMatrix = dataframe[filt].corr()
#----write your code below--------

ax = sns.heatmap(corrMatrix, annot=True)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

Out[47]: (8.5, -0.5)

Open-ended Question:

Which variables seem to correlate with Pump Failure?

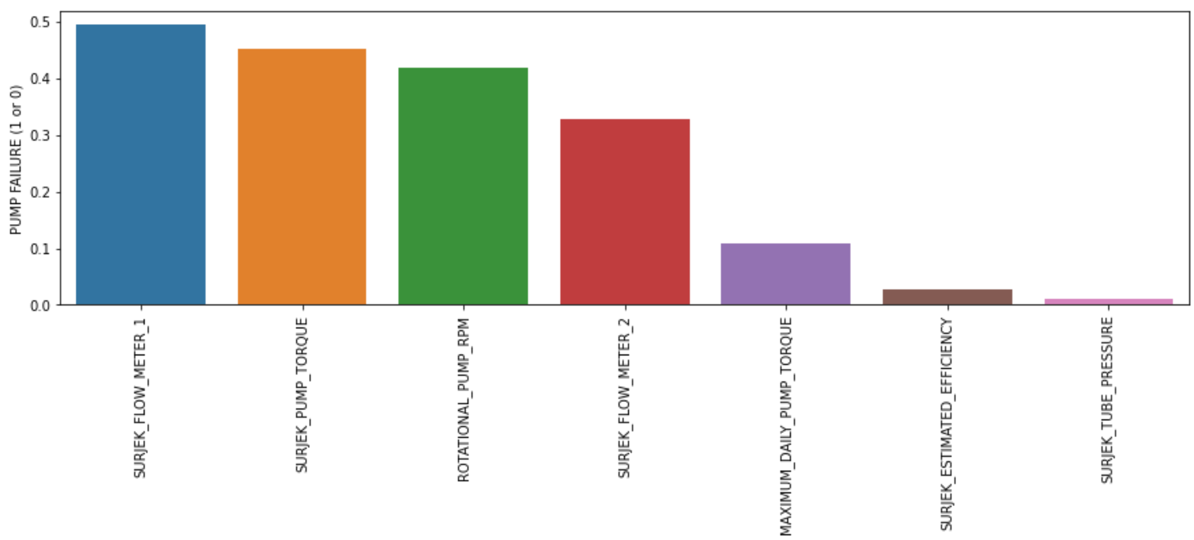
Step 12: Create a Barplot of Correlated Features

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

Please put your code here

In [37]:
```
filt = ['SURJEK_FLOW_METER_1', 'SURJEK_FLOW_METER_2', 'ROTATIONAL_PUMP_RPM',
       'SURJEK_PUMP_TORQUE', 'MAXIMUM_DAILY_PUMP_TORQUE',
       'SURJEK_TUBE_PRESSURE',
       'SURJEK_ESTIMATED_EFFICIENCY','PUMP FAILURE (1 or 0)']
corrMatrix = combine[filt].corr()
corr_data = corrMatrix.filter(items=['PUMP FAILURE (1 or 0)'])
corr_data = corr_data.loc[~corr_data.index.isin(['PUMP FAILURE (1 or 0)'])]
corr_data.sort_values(by='PUMP FAILURE (1 or 0)', ascending=False, inplace=Tru
e)
sns.barplot(x=corr_data.index,y='PUMP FAILURE (1 or 0)',data=corr_data)
plt.xticks(rotation=90)
print(corr_data)
```

```
                             PUMP FAILURE (1 or 0)
SURJEK_FLOW_METER_1                       0.494104
SURJEK_PUMP_TORQUE                        0.452761
ROTATIONAL_PUMP_RPM                       0.417384
SURJEK_FLOW_METER_2                       0.326740
MAXIMUM_DAILY_PUMP_TORQUE                 0.107467
SURJEK_ESTIMATED_EFFICIENCY               0.027143
SURJEK_TUBE_PRESSURE                      0.009966
```



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 [40]:
```
#Below is the first part of the code
filt = ['SURJEK_FLOW_METER_1', 'SURJEK_FLOW_METER_2', 'ROTATIONAL_PUMP_RPM',
       'SURJEK_PUMP_TORQUE', 'MAXIMUM_DAILY_PUMP_TORQUE',
       'SURJEK_TUBE_PRESSURE',
       'SURJEK_ESTIMATED_EFFICIENCY', 'PUMP FAILURE (1 or 0)']
#----write your code below------
rolling_corr = combine_std[filt].corr()
ax = sns.heatmap(rolling_corr, annot=True)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

Out[40]: (8.5, -0.5)

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 [41]:
```
#Answer for step i):
#Below is the first part of the code
dependentVar = combine['PUMP FAILURE (1 or 0)']
filt = ['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', 'PUMP FAILURE (1 or 0)']
filt2 = ['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']
#----write your code below------
mod = sm.OLS(dependentVar, combine[filt2])
res = mod.fit()

print(res.summary())
print('Parameters: ', res.params)
print('R2: ', res.rsquared)
```

```
                               OLS Regression Results                              
===============================================================================
=============
Dep. Variable:     PUMP FAILURE (1 or 0)   R-squared (uncentered):            
0.271
Model:                               OLS   Adj. R-squared (uncentered):       
0.270
Method:                    Least Squares   F-statistic:                       
370.6
Date:                   Fri, 18 Sep 2020   Prob (F-statistic):                
0.00
Time:                           20:04:21   Log-Likelihood:                    
7546.2
No. Observations:                   6998   AIC:                               
-1.508e+04
Df Residuals:                       6991   BIC:                               
-1.503e+04
Df Model:                              7                                      
Covariance Type:               nonrobust                                      
===============================================================================
==================
                                  coef    std err          t      P>|t|      
[0.025      0.975]
-------------------------------------------------------------------------------
------------------
SURJEK_FLOW_METER_1             0.0017   9.98e-05     16.845      0.000      
0.001       0.002
SURJEK_FLOW_METER_2            -0.0001   5.77e-05     -2.091      0.037      
-0.000   -7.53e-06
ROTATIONAL_PUMP_RPM             0.0003   8.16e-05      4.014      0.000      
0.000       0.000
SURJEK_PUMP_TORQUE              0.0001   1.43e-05      6.994      0.000    7.
21e-05       0.000
MAXIMUM_DAILY_PUMP_TORQUE     1.97e-05   2.17e-06      9.095      0.000    1.
55e-05    2.39e-05
SURJEK_AMMONIA_FLOW_RATE     1.309e-17   3.21e-18      4.079      0.000      
6.8e-18    1.94e-17
SURJEK_TUBE_PRESSURE        -2.925e-05   4.11e-06     -7.112      0.000   -3.
73e-05   -2.12e-05
SURJEK_ESTIMATED_EFFICIENCY    -0.0054      0.001     -4.058      0.000      
-0.008      -0.003
===============================================================================
=
Omnibus:                     7989.491   Durbin-Watson:                    0.04
4
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            975104.63
2
Skew:                           5.878   Prob(JB):                          0.0
0
Kurtosis:                      59.621   Cond. No.                      2.59e+1
9
===============================================================================
=

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
```

```
[2] The smallest eigenvalue is 5.08e-30. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
Parameters:  SURJEK_FLOW_METER_1            1.681222e-03
SURJEK_FLOW_METER_2           -1.206016e-04
ROTATIONAL_PUMP_RPM            3.275825e-04
SURJEK_PUMP_TORQUE             1.001112e-04
MAXIMUM_DAILY_PUMP_TORQUE      1.970263e-05
SURJEK_AMMONIA_FLOW_RATE       1.309230e-17
SURJEK_TUBE_PRESSURE          -2.924937e-05
SURJEK_ESTIMATED_EFFICIENCY   -5.393725e-03
dtype: float64
R2:  0.2706273642509147
```

In [44]:
```
#Answer for step ii):
#Below is the first part of the code
y = combine_std['PUMP FAILURE (1 or 0)']
filt = ['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', 'PUMP FAILURE (1 or 0)']
filt2 = ['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']
#----write your code below------
X = combine_std[filt2]
mod = sm.OLS(y, X).fit()

print(mod.summary())
print('Parameters: ', mod.params)
print('R2: ', mod.rsquared)
```

```
                                 OLS Regression Results                                
=============================================================================================
Dep. Variable:     PUMP FAILURE (1 or 0)   R-squared (uncentered):                   0.623
Model:                               OLS   Adj. R-squared (uncentered):              0.622
Method:                    Least Squares   F-statistic:                              1640.
Date:                   Fri, 18 Sep 2020   Prob (F-statistic):                        0.00
Time:                           20:05:03   Log-Likelihood:                          9795.5
No. Observations:                   6969   AIC:                                -1.958e+04
Df Residuals:                       6962   BIC:                                -1.953e+04
Df Model:                              7                                                  
Covariance Type:               nonrobust                                                  
==================================================================================================
                                 coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------------
SURJEK_FLOW_METER_1           -0.0031      0.000     -9.286      0.000      -0.004      -0.002
SURJEK_FLOW_METER_2            0.0016   6.43e-05     24.210      0.000       0.001       0.002
ROTATIONAL_PUMP_RPM           -0.0066      0.000    -30.866      0.000      -0.007      -0.006
SURJEK_PUMP_TORQUE             0.0003   2.56e-05     11.072      0.000       0.000       0.000
MAXIMUM_DAILY_PUMP_TORQUE      0.0057   8.77e-05     65.365      0.000       0.006       0.006
SURJEK_AMMONIA_FLOW_RATE     2.555e-18   3.57e-19      7.160      0.000    1.86e-18    3.25e-18
SURJEK_TUBE_PRESSURE          -0.0017      0.000     -3.766      0.000      -0.003      -0.001
SURJEK_ESTIMATED_EFFICIENCY   -0.0144      0.002     -7.931      0.000      -0.018      -0.011
==============================================================================
Omnibus:                     3048.583   Durbin-Watson:                   0.083
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           402317.328
Skew:                           1.071   Prob(JB):                         0.00
Kurtosis:                      40.161   Cond. No.                     2.77e+19
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
[2] The smallest eigenvalue is 1.68e-32. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
Parameters:  SURJEK_FLOW_METER_1           -3.097922e-03
SURJEK_FLOW_METER_2            1.555912e-03
ROTATIONAL_PUMP_RPM           -6.613811e-03
SURJEK_PUMP_TORQUE             2.832798e-04
MAXIMUM_DAILY_PUMP_TORQUE      5.731974e-03
SURJEK_AMMONIA_FLOW_RATE       2.554609e-18
SURJEK_TUBE_PRESSURE          -1.723347e-03
SURJEK_ESTIMATED_EFFICIENCY   -1.440638e-02
dtype: float64
R2:  0.622501809640275
```

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)'

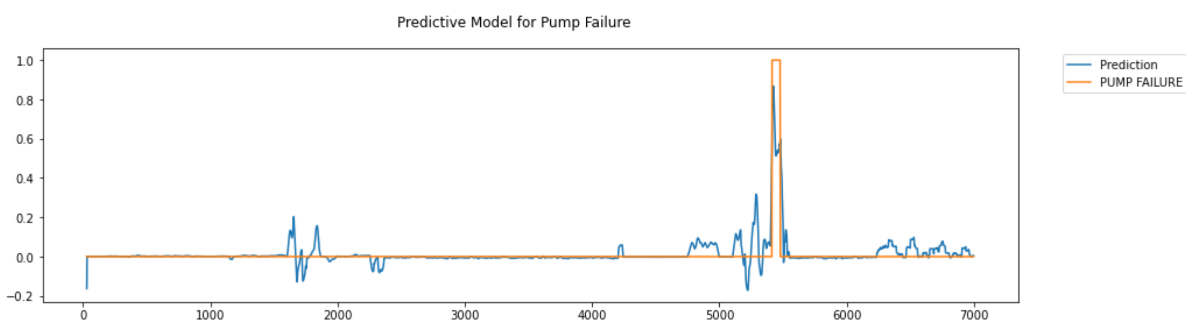
Note: Please ensure all axes are clearly labelled and ensure that you use Dual Axes to plot this. Make the line widths wider than 1 so the plots are easier to see. We have provided the initial figure size.

Please put your code here

In [45]:

```
#Below is the first part of the code
#----write your code below------
fig,ax = plt.subplots(figsize=(15,4))
fig.suptitle('Predictive Model for Pump Failure')
predicted = mod.predict(X)
ax.plot(predicted, label='Prediction')
ax.plot(y, label='PUMP FAILURE')
ax.legend(bbox_to_anchor=(1.04,1), loc="upper left")
```

Out[45]: <matplotlib.legend.Legend at 0x1d79c6060d0>



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!