Welcome to the Southern Water Corp Python Case Study!

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

Don't forget that to modify the matplotlib plot sizes so they're at a comfortable reading size you should use the following:

import matplotlib as mpl

mpl.rcParams['figure.figsize'] = (20,5)

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.

statsmodels.api - This library will enable you to create statistical models. You will need this library when performing 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
%matplotlib inline
import seaborn as sns
import pandas as pd
import statsmodels.api as sm
%cd c:\SWC Case\SWC_Updated_Case_Study_Python
import numpy as np
```

```
c:\SWC Case\SWC_Updated_Case_Study_Python
```

Step 2: Descriptive Statistics

The data you've received from Southern Water Corp has been split into two files. The first file, titled DF_Raw_Data contains all the 'raw' Pump Data you will need for your analysis. The second file, titled DF_Rolling_Stdev contains the Rolling Standard Deviation Data you will need for questions 10 Onwards.

We have deliberately set up the data in this manner so please ensure that when you need to perform the rolling standard deviation calculations, you use the DF_Rolling_Stdev.csv file.

i. Import each of the two data sources and store them into their individual dataframes. Suggested names: dataframe_raw & dataframe_stdev respectively. Don't forget to use the header argument to ensure your columns have meaningful names!

ii. Print descriptive statistics for each of the dataframes using .describe() and .info()

In [2]:

```
dataframe_raw = pd.read_csv('DF_Raw_Data.csv', header=0)
dataframe_stdev = pd.read_csv('DF_Rolling_Stdev.csv', header=0)
print(dataframe_raw.describe())
print(dataframe_raw.info())
print(dataframe_stdev.describe())
print(dataframe_stdev.info())
```

```
       Volumetric Flow Meter 1  Volumetric Flow Meter 2  Pump Speed (RPM)  \
count              2453.000000              2453.000000       2453.000000
mean                 41.802629                41.796702         90.796576
std                   3.656576                 3.654873         10.217885
min                   0.000000                 0.000000          0.000000
25%                  41.050000                41.000000         85.000000
50%                  42.100000                42.140000         91.000000
75%                  43.140000                43.130000         97.000000
max                  45.900000                45.840000        124.000000

       Pump Torque   Ambient Temperature  Horse Power  Pump Efficiency  \
count  2453.000000          2453.000000  2453.000000      2453.000000
mean    202.851610            50.226661     3.540897        76.015149
std      22.683977             5.298203     0.579055         6.651633
min       0.000000             0.000000     0.000000         0.000000
25%     191.000000            48.000000     3.220000        74.560000
50%     203.000000            51.000000     3.480000        76.620000
75%     215.000000            53.000000     3.780000        78.470000
max     264.000000            65.000000     7.560000        83.450000

       PUMP FAILURE (1 or 0)
count            2453.000000
mean                0.021199
std                 0.144075
min                 0.000000
25%                 0.000000
50%                 0.000000
75%                 0.000000
max                 1.000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2453 entries, 0 to 2452
Data columns (total 10 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Data Source              2453 non-null   object
 1   TIMEFRAME (DD/MM/YYYY)   2453 non-null   object
 2   Volumetric Flow Meter 1  2453 non-null   float64
 3   Volumetric Flow Meter 2  2453 non-null   float64
 4   Pump Speed (RPM)         2453 non-null   int64
 5   Pump Torque              2453 non-null   int64
 6   Ambient Temperature      2453 non-null   int64
 7   Horse Power              2453 non-null   float64
```

```
 8   Pump Efficiency          2453 non-null   float64
 9   PUMP FAILURE (1 or 0)    2453 non-null   int64
dtypes: float64(4), int64(4), object(2)
memory usage: 191.8+ KB
None
       Volumetric Flow Meter 1  Volumetric Flow Meter 2  Pump Speed (RPM)  \
count              2452.000000              2452.000000       2452.000000
mean                  1.485126                 1.497361          6.648308
std                   2.294950                 2.282053          5.722897
min                   0.380000                 0.640000          0.580000
25%                   1.070000                 1.080000          5.520000
50%                   1.160000                 1.170000          5.990000
75%                   1.230000                 1.260000          6.460000
max                  21.390000                21.530000         59.310000

       Pump Torque   Ambient Temperature  Horse Power  Pump Efficiency  \
count  2452.000000           2452.000000   2452.00000      2452.000000
mean     13.945338              3.436370      0.37060         2.725232
std      12.394302              3.043042      0.29979         4.186723
min       5.000000              0.900000      0.11000         1.170000
25%      11.210000              2.920000      0.28000         1.960000
50%      12.180000              3.160000      0.32000         2.120000
75%      13.110000              3.370000      0.36000         2.270000
max     124.710000             30.650000      3.32000        39.150000

       PUMP FAILURE (1 or 0)
count            2452.000000
mean                0.021207
std                 0.144104
min                 0.000000
25%                 0.000000
50%                 0.000000
75%                 0.000000
max                 1.000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2452 entries, 0 to 2451
Data columns (total 10 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Data Source              2452 non-null   object
 1   TIMEFRAME (DD/MM/YYYY)   2452 non-null   object
 2   Volumetric Flow Meter 1  2452 non-null   float64
 3   Volumetric Flow Meter 2  2452 non-null   float64
 4   Pump Speed (RPM)         2452 non-null   float64
 5   Pump Torque              2452 non-null   float64
 6   Ambient Temperature      2452 non-null   float64
 7   Horse Power              2452 non-null   float64
 8   Pump Efficiency          2452 non-null   float64
 9   PUMP FAILURE (1 or 0)    2452 non-null   int64
dtypes: float64(7), int64(1), object(2)
memory usage: 191.7+ 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 from the .info command you used previously.

You should now also have a rough sense of the number of entires in each dataset (~2,452). 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) Using the dataframe_raw, create a boxplot visualising this information.

ii) Using the dataframe_raw, create a lineplot visualising this information.

Hint: You might want to reference the following .plot function (https://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandas.DataFrame.plot.html)

Please put your code here

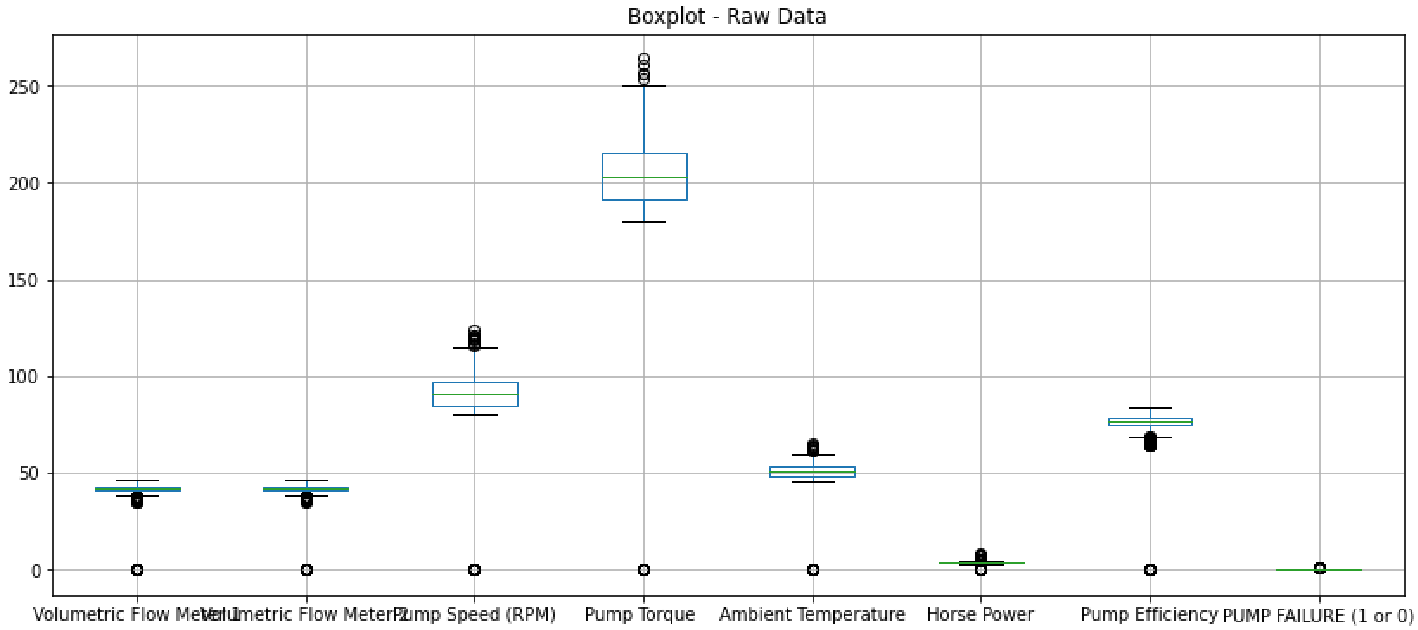
We've included an example of what your Box Plot *should* look like once you've plotted this using the dataframe_raw dataset

In [102]:

```
dataframe_raw.boxplot(figsize=(14, 6))
plt.title('Boxplot - Raw Data')
```

Out[102]:

```
Text(0.5, 1.0, 'Boxplot - Raw Data')
```



In [103]:

```
dataframe_raw.plot(figsize=(14, 6))
plt.title('Lineplot - Raw Data')
```

Out[103]:

```
Text(0.5, 1.0, 'Lineplot - Raw Data')
```

In [ ]:

```

```

What have you observed from the boxplot and line plots?

In [150]:

```
print('The boxplot shows that the data of all other variables except Pump Torque and Pump Speed are barely spread out. Only Pump Torque and Pump Speed data are relatively spread out.')
```

```
The boxplot shows that the data of all other variables except Pump Torque and Pump Speed
are barely spread out. Only Pump Torque and Pump Speed data are relatively spread out.
```

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.

More importantly, the dataset we do have contains Pump Failure Data where it has failed (i.e. Pump Failure = 0) as well as when it is operating normally. We should separate this data accordingly to more effectively visualise the information.

Step 4: Filtered Dataframes with Box Plots

i) Using the dataframe_raw dataset, create two boxplots specifically for when the Pump has failed (i.e. Pump Failure = 1) and 0 (Pump is in normal operations).

As part of best practice, don't forget to clearly title your box plots so we can identify which plot is for the failure and which plot is for the normal operations.

To do this, you'll have to recall how to apply boolean filtering to a dataframe.

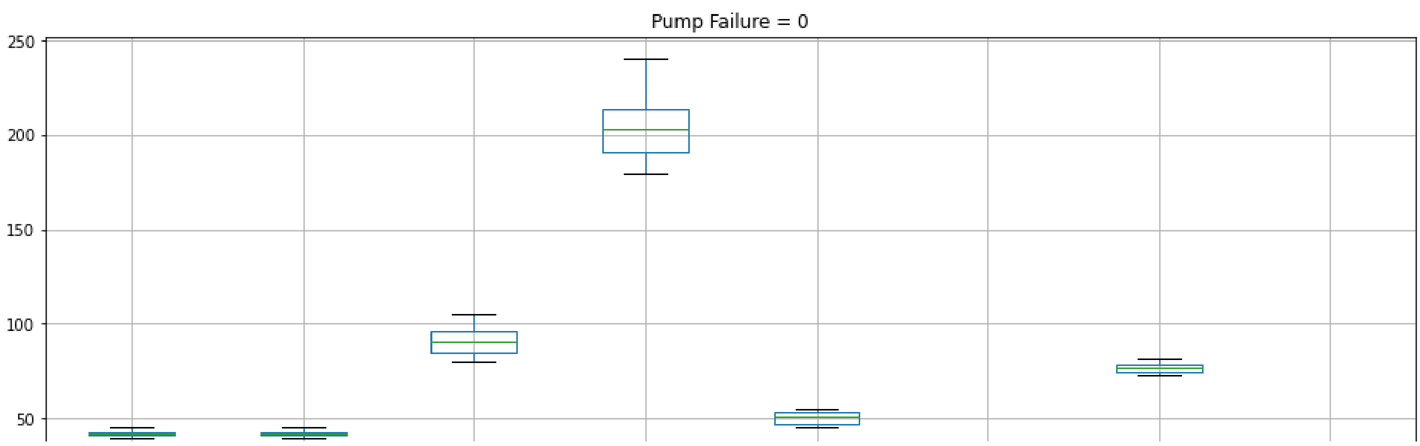
Please put your code here

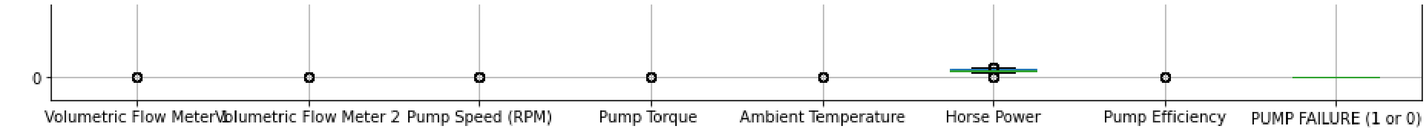
In [104]:

```
dataframe_raw[dataframe_raw['PUMP FAILURE (1 or 0)']==0].boxplot(figsize=(16, 6), )
plt.title('Pump Failure = 0')
```

Out[104]:

```
Text(0.5, 1.0, 'Pump Failure = 0')
```





Open-ended Question:

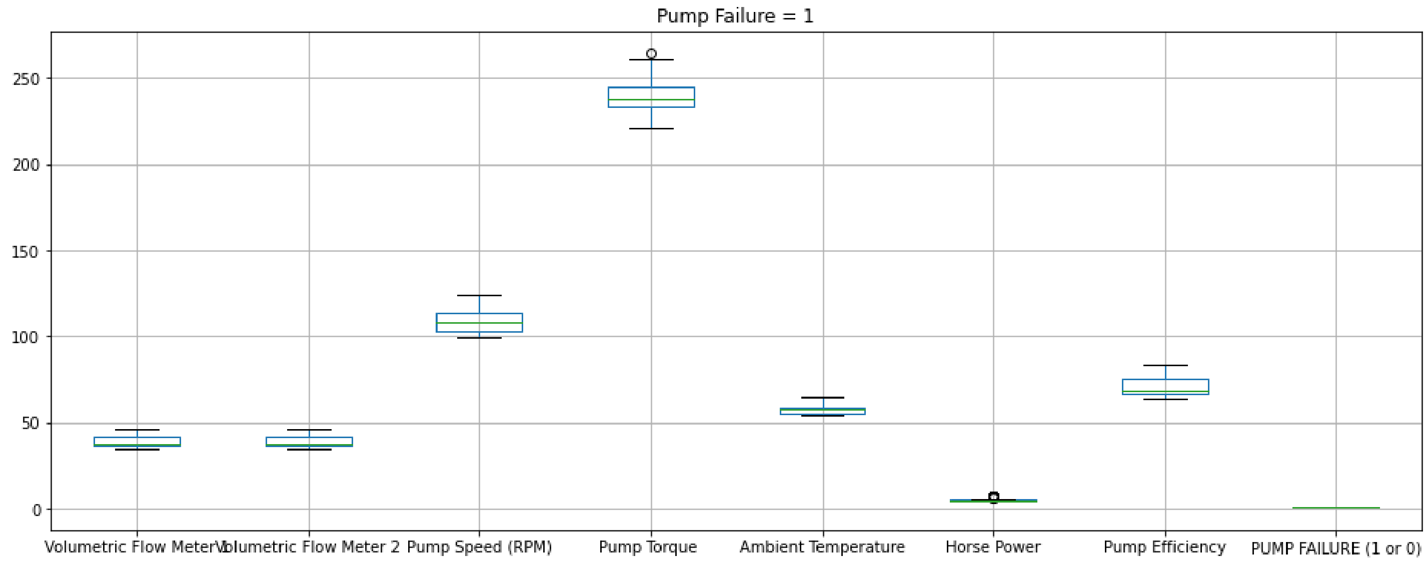
What have you noticed when you compared the dataset in this manner?

In [105]:

```
dataframe_raw[dataframe_raw['PUMP FAILURE (1 or 0)']==1].boxplot(figsize=(16, 6))
plt.title('Pump Failure = 1')
```

Out[105]:

```
Text(0.5, 1.0, 'Pump Failure = 1')
```



From analysing the boxplots, you might notice that there seem to be a number of outliers. We might want to see if we can actively remove this with Python.

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 5.

Step 5: 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 [115]:

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

```
print(Q1, Q3, IQR)
```

```
Volumetric Flow Meter 1     41.05
Volumetric Flow Meter 2     41.00
Pump Speed (RPM)            85.00
Pump Torque                191.00
Ambient Temperature         48.00
Horse Power                  3.22
Pump Efficiency             74.56
PUMP FAILURE (1 or 0)        0.00
Name: 0.25, dtype: float64 Volumetric Flow Meter 1     43.14
Volumetric Flow Meter 2     43.13
Pump Speed (RPM)            97.00
Pump Torque                215.00
Ambient Temperature         53.00
Horse Power                  3.78
Pump Efficiency             78.47
PUMP FAILURE (1 or 0)        0.00
Name: 0.75, dtype: float64 Volumetric Flow Meter 1     2.09
Volumetric Flow Meter 2     2.13
Pump Speed (RPM)           12.00
Pump Torque                24.00
Ambient Temperature         5.00
Horse Power                 0.56
Pump Efficiency             3.91
PUMP FAILURE (1 or 0)       0.00
dtype: float64
```

Step 6: 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.

Please put your code here

In [107]:

```
cnt = dataframe_raw.count()
u_ols = dataframe_raw[dataframe_raw > Q3+1.5*IQR].count()
l_ols = dataframe_raw[dataframe_raw < Q1-1.5*IQR].count()
cnt_wo_ols = cnt - u_ols - l_ols
prop_outliers = (u_ols+l_ols)*100/cnt
print(prop_outliers)
```

```
Data Source                0.000000
TIMEFRAME (DD/MM/YYYY)     0.000000
Volumetric Flow Meter 1    1.793722
Volumetric Flow Meter 2    1.834488
Pump Speed (RPM)           1.019160
Pump Torque                0.856095
Ambient Temperature        0.978394
Horse Power                3.383612
Pump Efficiency            1.752956
PUMP FAILURE (1 or 0)      2.119853
dtype: float64
```

Having removed the outliers from the dataset - do we think this is a good option? Why or why not?

In [125]:

```
print("Although the proportion of outliers found to be insignificant (highest found in Ho
rse Power at 3.38%), removing them might distort insights since the outliers may be relat
ed to pump failure.")
```

```
Although the proportion of outliers found to be insignificant (highest found in Horse Pow
er at 3.38%), removing them might distort insights since the outliers may be related to p
ump failure.
```

Step 7: Create a Boxplot without Outliers

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

i) A boxplot when PUMP FAILURE is 1 (Check what the length of the dataframe is before you try and plot this. You may be surprised!)

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. If you remove all the outliers - you might discover you have nothing to plot...!

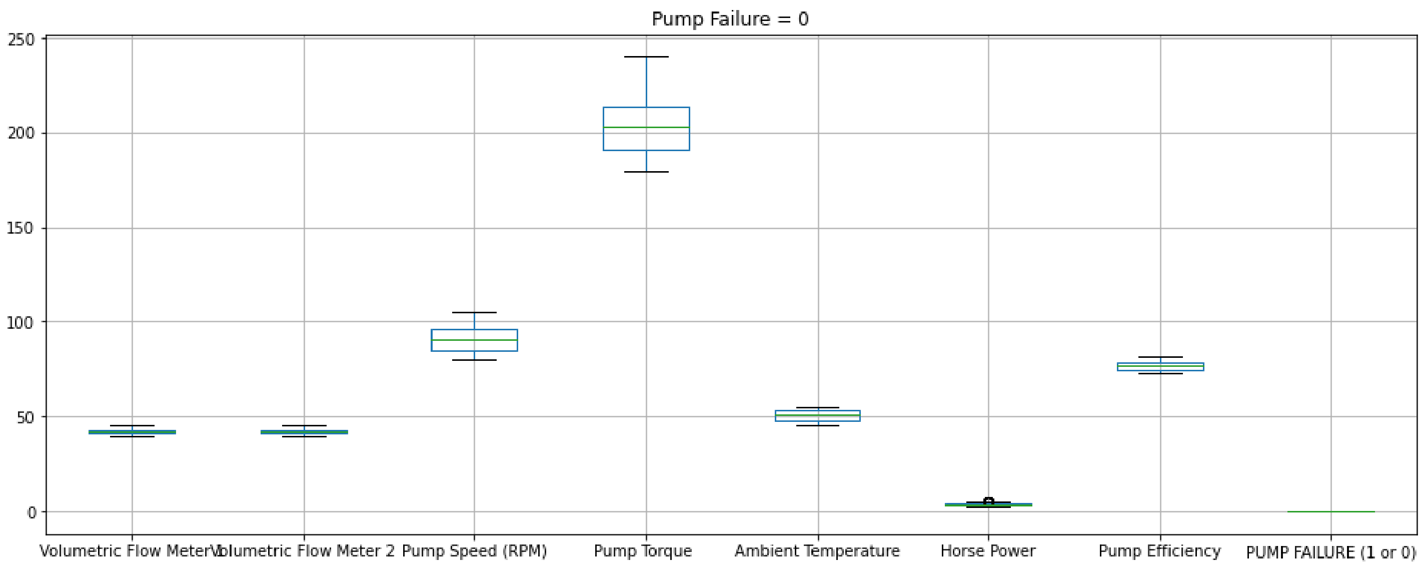
Please put your code here

In [123]:

```
lo_whis = Q1 - 1.5*IQR
up_whis = Q3 + 1.5*IQR
dataframe_wo_ols = dataframe_raw[(dataframe_raw >= lo_whis) & (dataframe_raw <= up_whis)
]
dataframe_wo_ols[dataframe_wo_ols['PUMP FAILURE (1 or 0)']==0].boxplot(figsize=(16, 6),
)
plt.title('Pump Failure = 0')
```

Out[123]:

```
Text(0.5, 1.0, 'Pump Failure = 0')
```

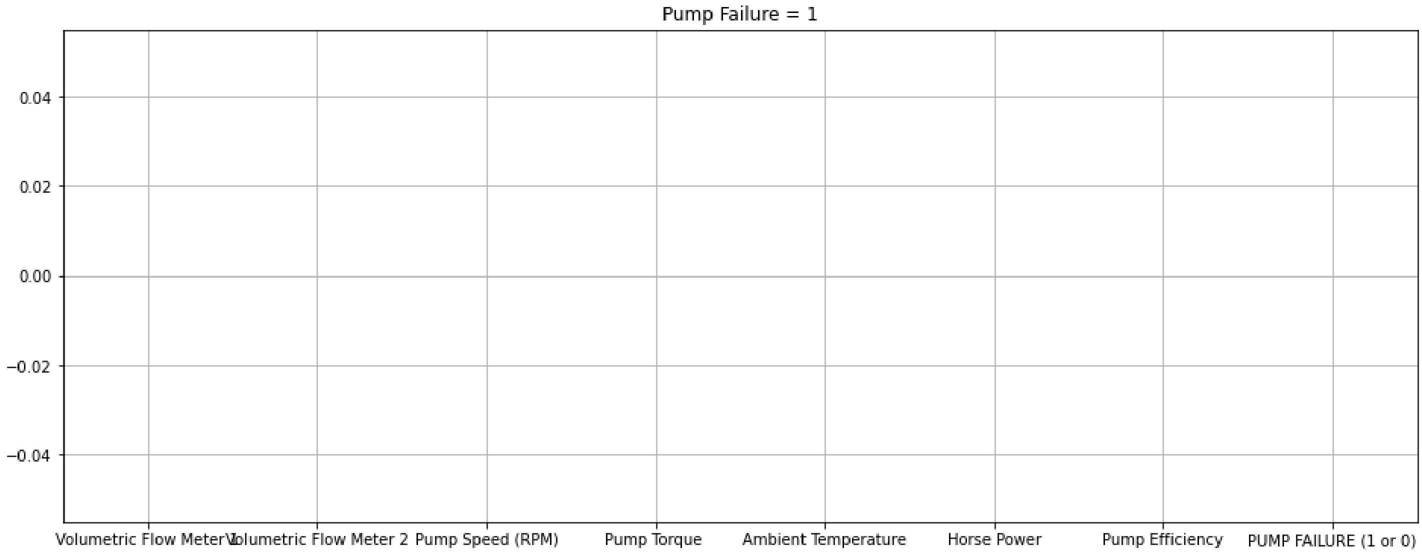


In [124]:

```
dataframe_wo_ols[dataframe_wo_ols['PUMP FAILURE (1 or 0)']==1].boxplot(figsize=(16, 6),
)
plt.title('Pump Failure = 1')
```

Out[124]:

```
Text(0.5, 1.0, 'Pump Failure = 1')
```



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 and the portion of data you need is actually stored WITHIN the Outliers.

This is exactly why you should never remove Outliers without Subject Matter Expertise input. Otherwise valuable information may be discarded.

Step 8: Plot and Examine Each Column

As you might recall from the earlier plot you had made with the line plot; it was hard to see which variables were the most significant with respect to pump failure when all the variables are plotted together. This is why we are going to ITERATE through the dataframe and plot each individual variable out and compare this with the Pump Failure.

This will require you to make use of the following syntax:

for variable in listOfVariables:

```
#Loop through each variable in the dataframe (i.e. dataframe[___].plot
#Specify the dual-axis (i.e. ax.twinx())
#Plot the Pump Failure (1 or 0) on the secondary axes
#Include Plot Titles for each plot (i.e. print ("This is for the attribute " + i))
```

Using the syntax provided, loop through the dataframe_raw dataset, plotting every variable individually, against the Pump Failure to better identify trends.

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 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.

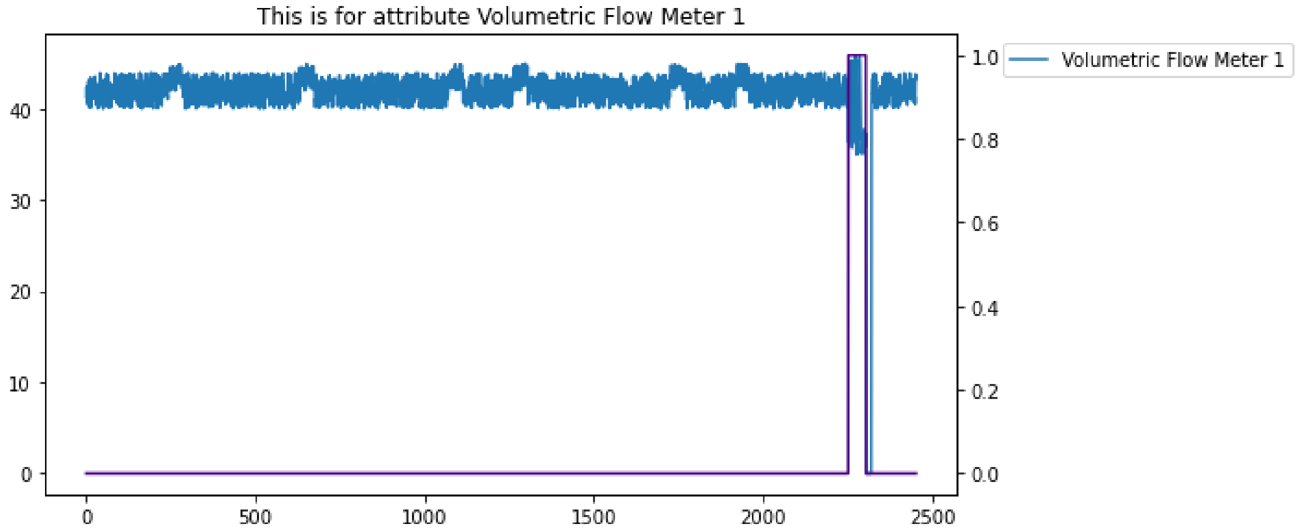
Check out this link to learn how to do this: https://matplotlib.org/gallery/api/two_scales.html

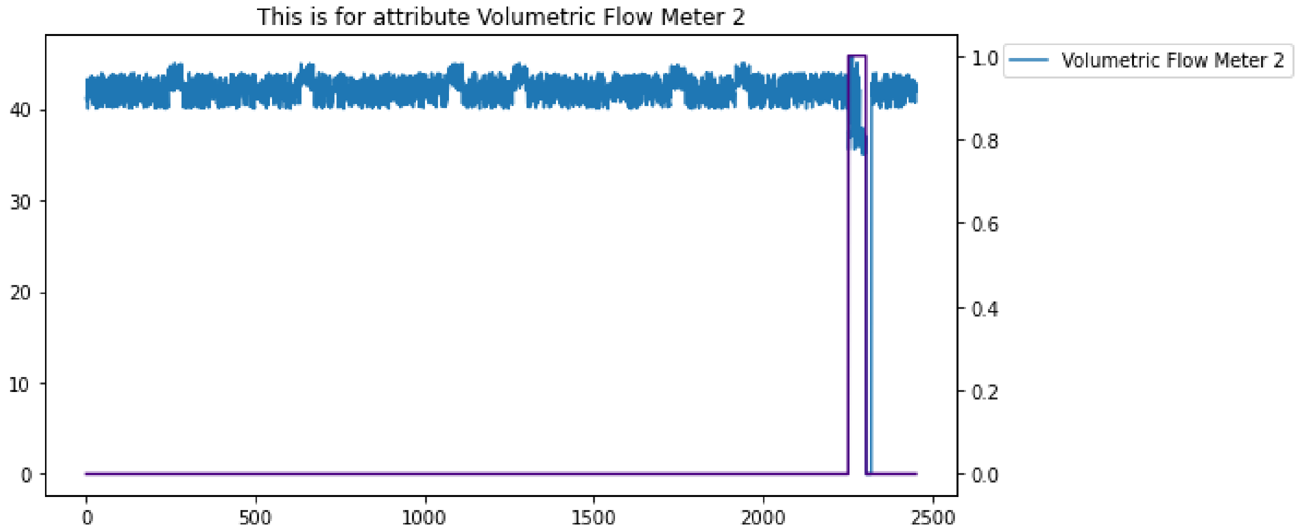
Note: Please ensure that the dataframe you are plotting contains all the outliers and that the Pump Failure Behaviour includes both the 0 and 1 State.

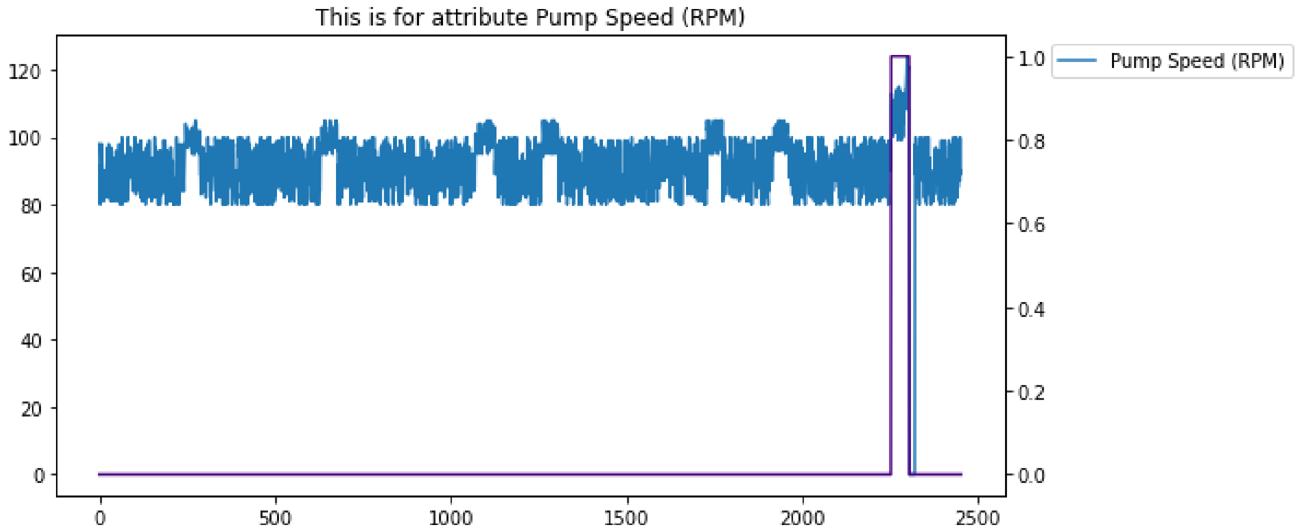
Please put your code here

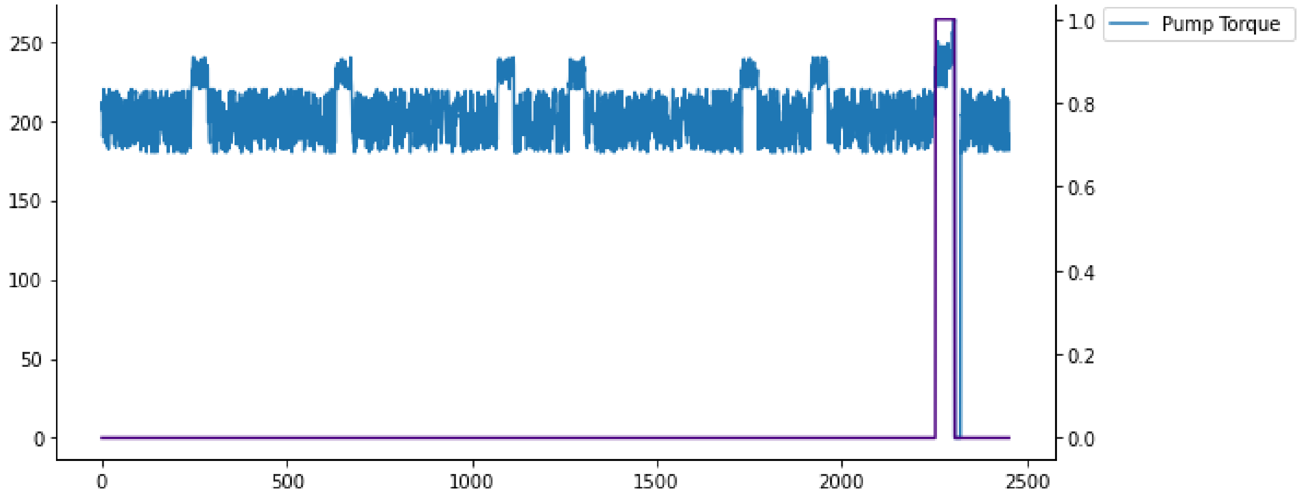
In [108]:

```
listOfVariables = dataframe_raw.drop(['PUMP FAILURE (1 or 0)','Data Source','TIMEFRAME (D
D/MM/YYYY)'], axis=1).columns

for i in listOfVariables:
    failureState = dataframe_raw['PUMP FAILURE (1 or 0)']
    ax = dataframe_raw[i].plot(figsize=(10, 4))
    ax2 = ax.twinx()
    ax2.plot(failureState, 'indigo')
    ax.legend(bbox_to_anchor=(1.04,1), loc="upper left")
    plt.tight_layout()
    plt.title("This is for attribute " + i)
    plt.show()
```

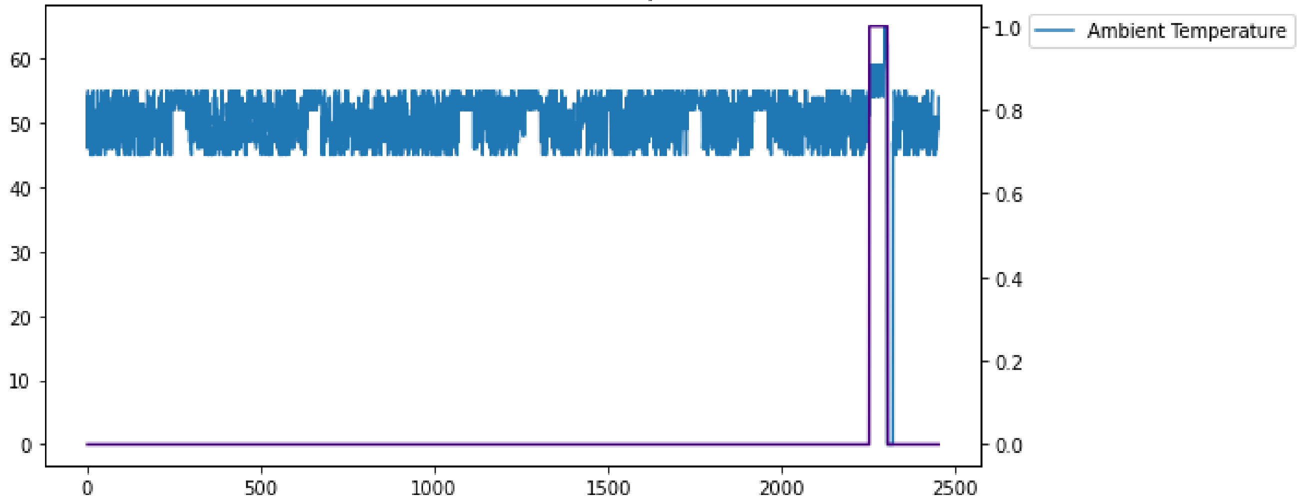




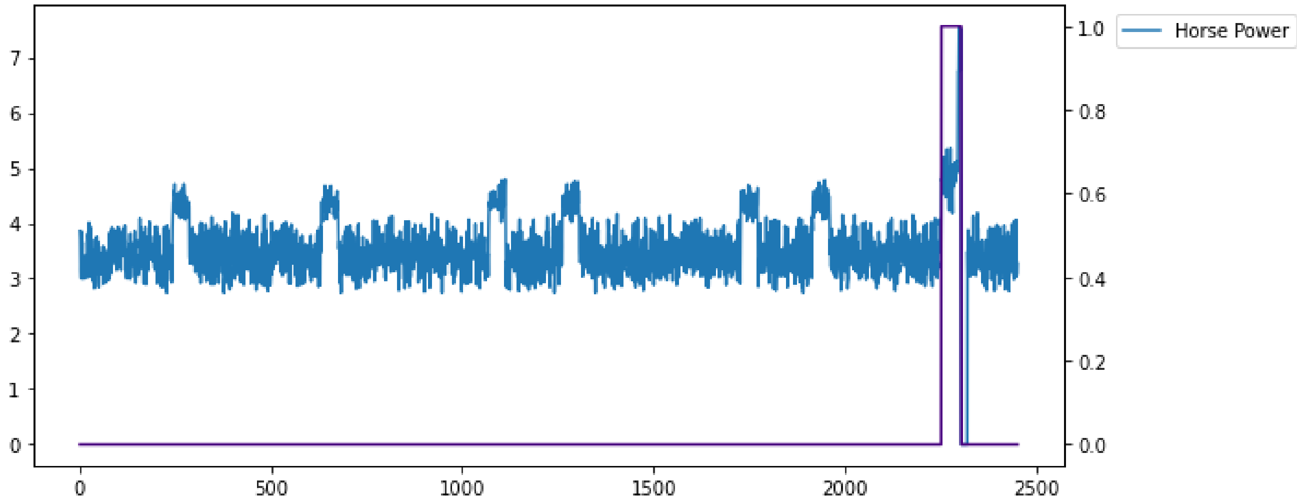




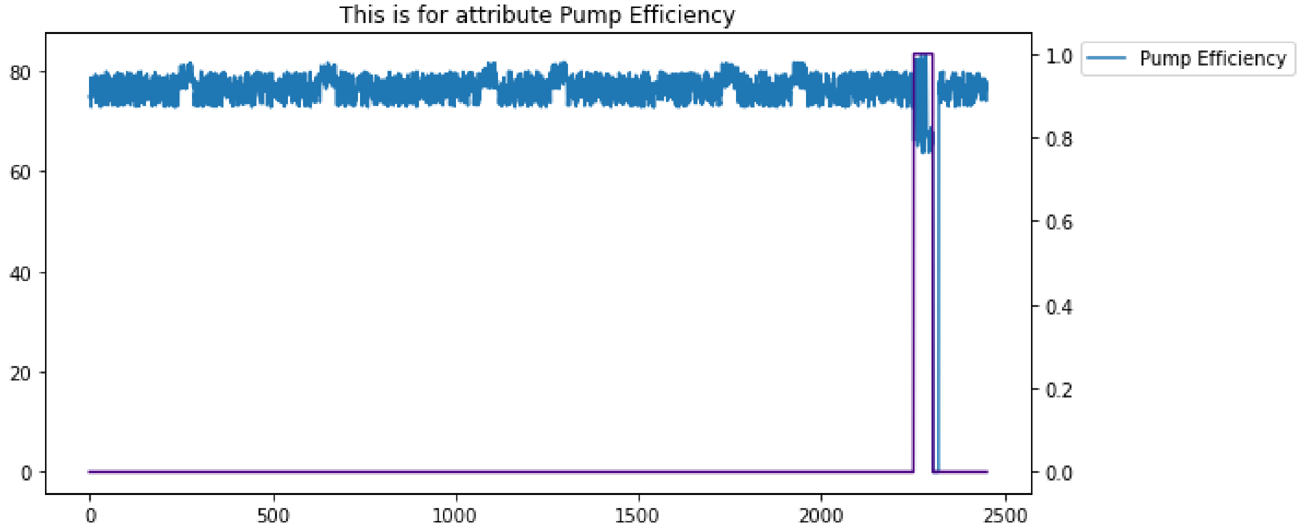
This is for attribute Ambient Temperature



This is for attribute Horse Power



This is for attribute Pump Efficiency



What do you notice when looking at the data in this way? Do any particular trends emerge?

In [110]:

```
print('While Pump Speed, Pump Torque and Horse Power are observed to be slightly spiked i
n the event of pump failure = 1, we have observed similar spikes on other intermittent oc
cassions when Pump Failure = 0 too. Only Ambient Temperature is observed to be clearly sp
iked only when Pump Failure = 1 signaling a clear correlation. Volumetric Flow Meter 1 an
d 2 and Pump Efficiency all showed some peaks and troughs at Pump Failure = 1.')
```

```
While Pump Speed, Pump Torque and Horse Power are observed to be slightly spiked in the e
vent of pump failure = 1, we have observed similar spikes on other intermittent occassion
s when Pump Failure = 0 too. Only Ambient Temperature is observed to be clearly spiked on
ly when Pump Failure = 1 signaling a clear correlation. Volumetric Flow Meter 1 and 2 and
Pump Efficiency all showed some peaks and troughs at Pump Failure = 1.
```

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.

This is where we will switch to using the dataframe_stdev that you had previously defined in Q1.

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 9: Create a Plot for Pump Failures Over a Rolling Time Period

i) Set the index of the dataframe to the TIMEFRAME (DD/MM/YYYY) attribute

ii) Exactly as you did in Q8, Re-plot all variables, now transformed via a rolling standard deviation in the dataframe_stdev for the time period 10/12/2014 13:30 to 10/12/2014 14:30 against Pump Failure.

Note: To effectively filter on the time period you will need to make use of the below syntax

```
# dataframe_time_filtered = dataframe[(dataframe.index >= "_____") & (dataframe.in
dex <= "_____")
```

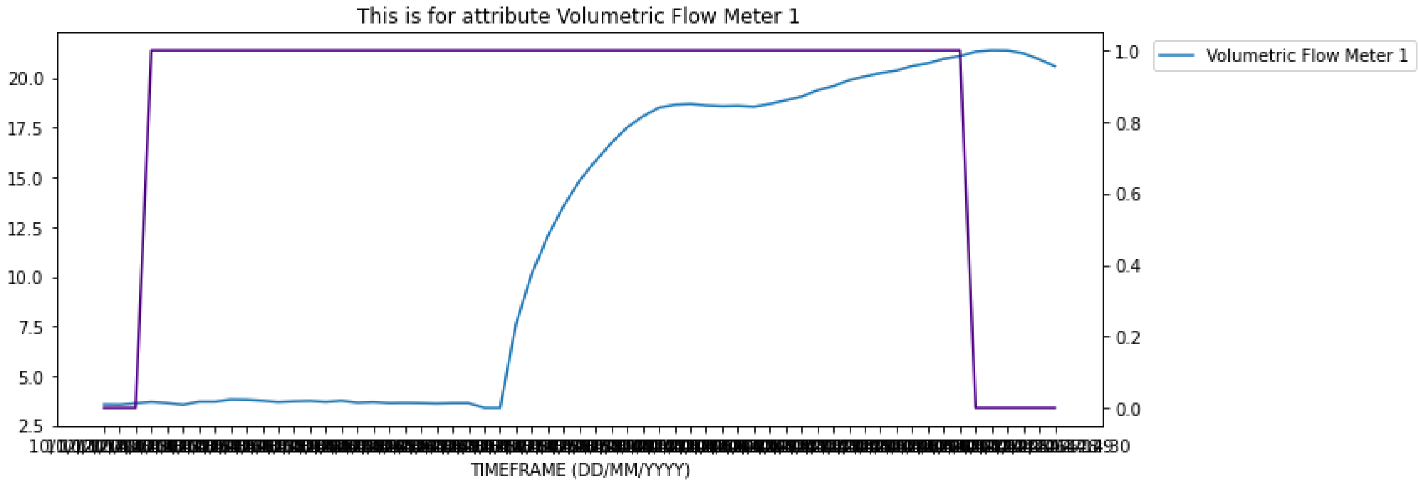
Please put your code here

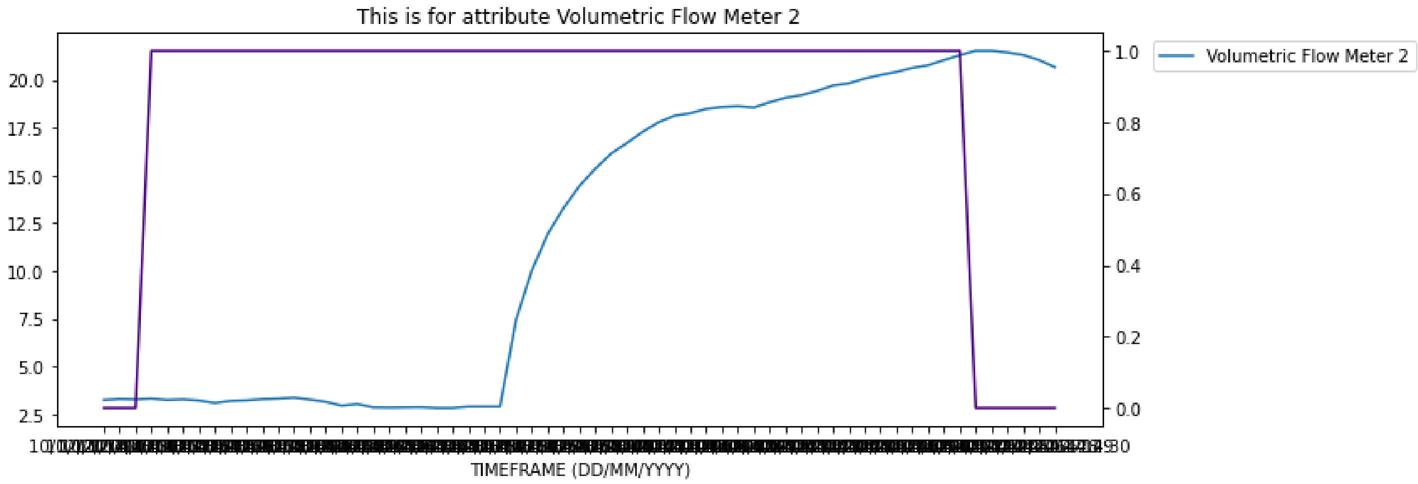
The output from your code should display image(s) like the one shown below

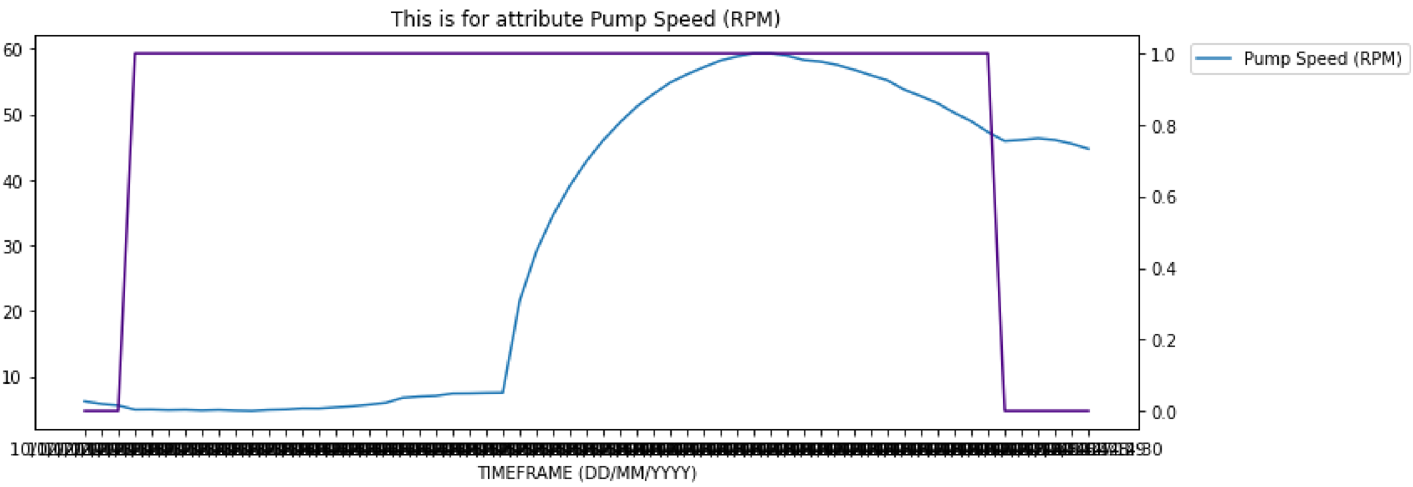
In [123]:

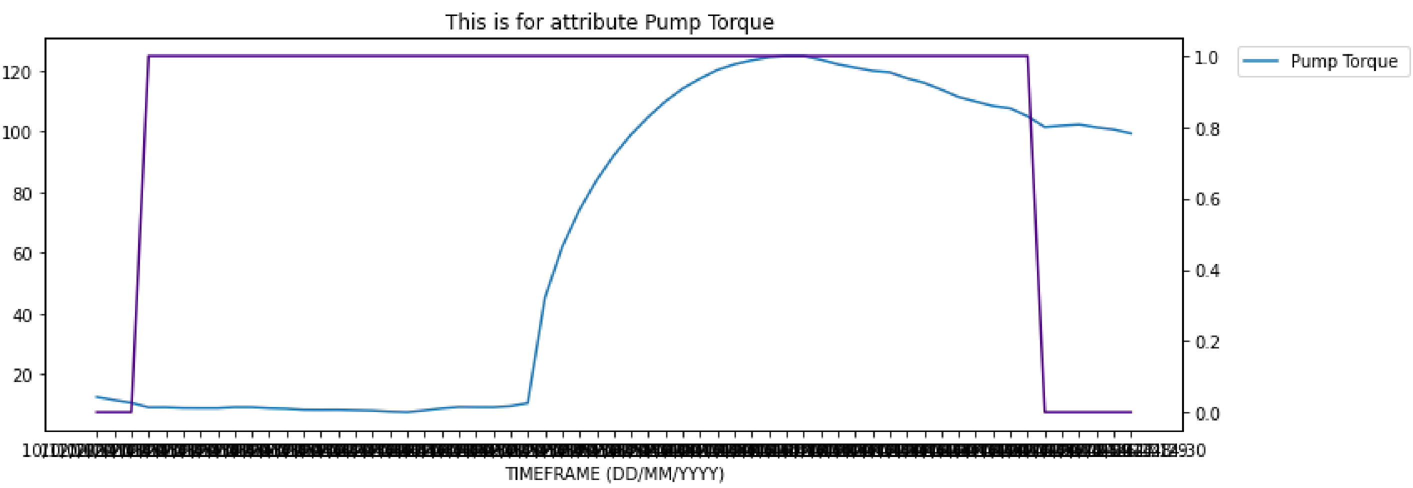
```
dataframe_stdev_indexed = dataframe_stdev.set_index(['TIMEFRAME (DD/MM/YYYY)'])
dataframe_time_filtered = dataframe_stdev_indexed[(dataframe_stdev_indexed.index >= "10/1
2/2014 13:30") & (dataframe_stdev_indexed.index <= "10/12/2014 14:30")]
listOfVariables = dataframe_time_filtered.drop(['PUMP FAILURE (1 or 0)','Data Source'], a
xis=1).columns

for i in listOfVariables:
    failureState = dataframe_time_filtered['PUMP FAILURE (1 or 0)']
    ax = dataframe_time_filtered[i].plot(figsize=(12, 4))
    ax2 = ax.twinx()
    ax2.plot(failureState, 'indigo')
    ax.legend(bbox_to_anchor=(1.04,1), loc="upper left")
    plt.tight_layout()
```

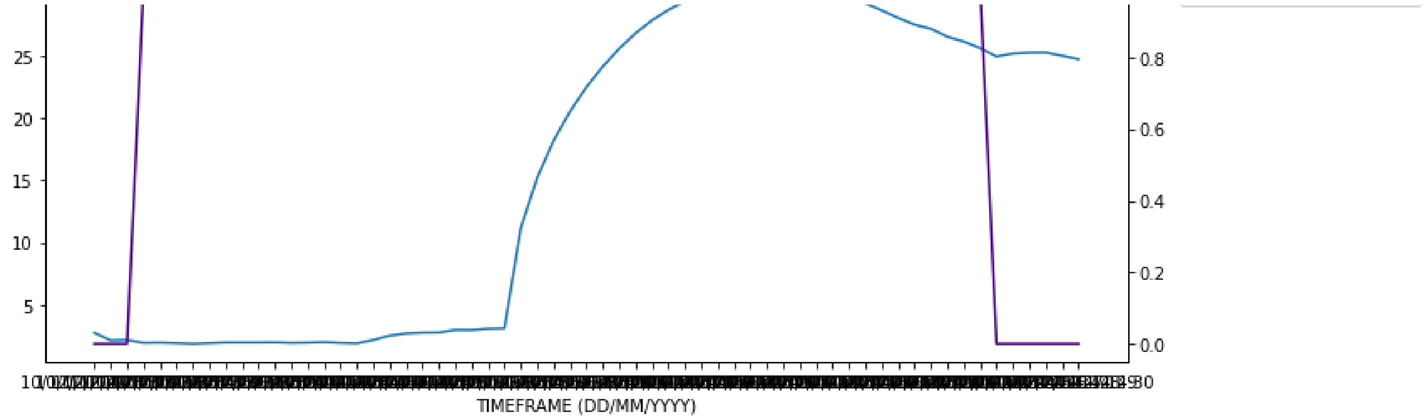
```
    plt.title("This is for attribute " + i)
    plt.show()
```

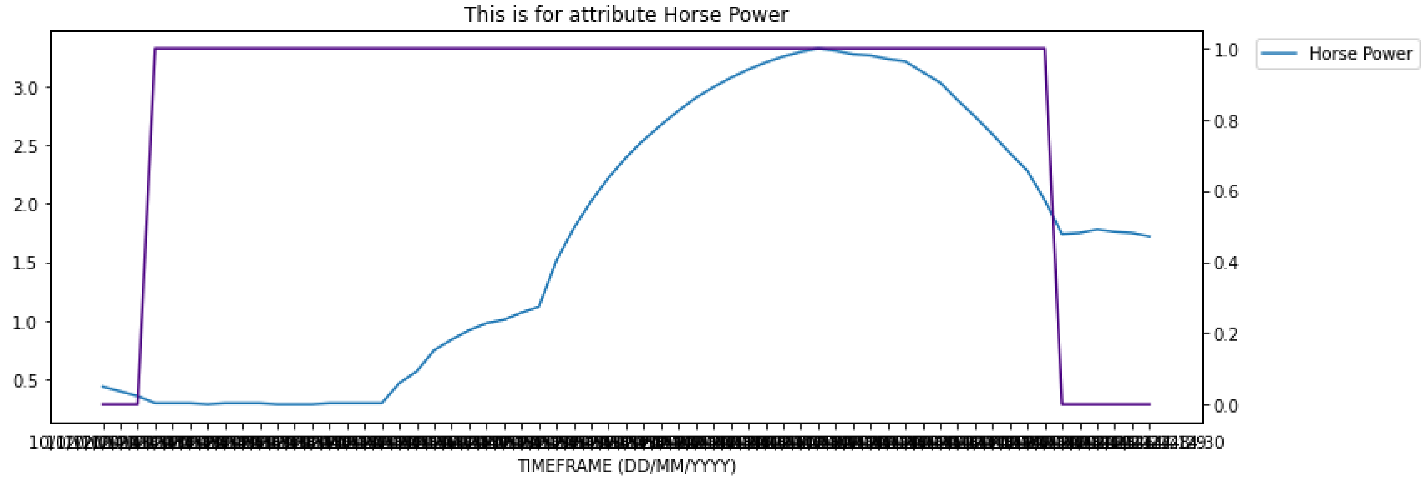


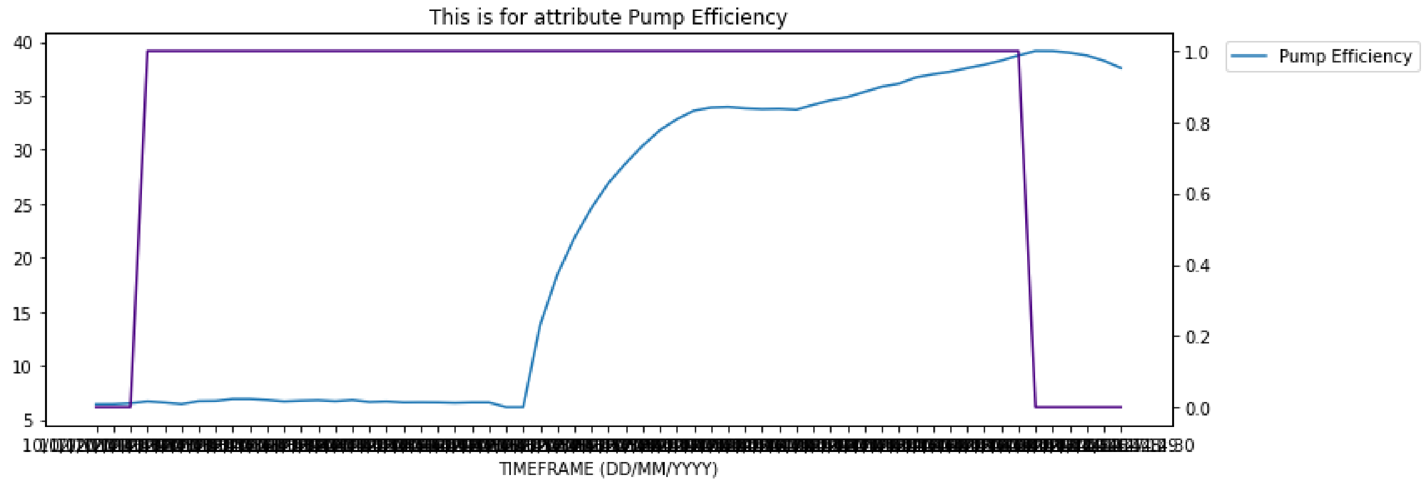












Part II: Inferential Statistical Analysis

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 10: Create a Heatmap

i) Using Seaborn's heatmap function, create a heatmap that clearly shows the correlations (including R Squared) for all variables using the dataframe_raw dataset.

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

Please put your code here

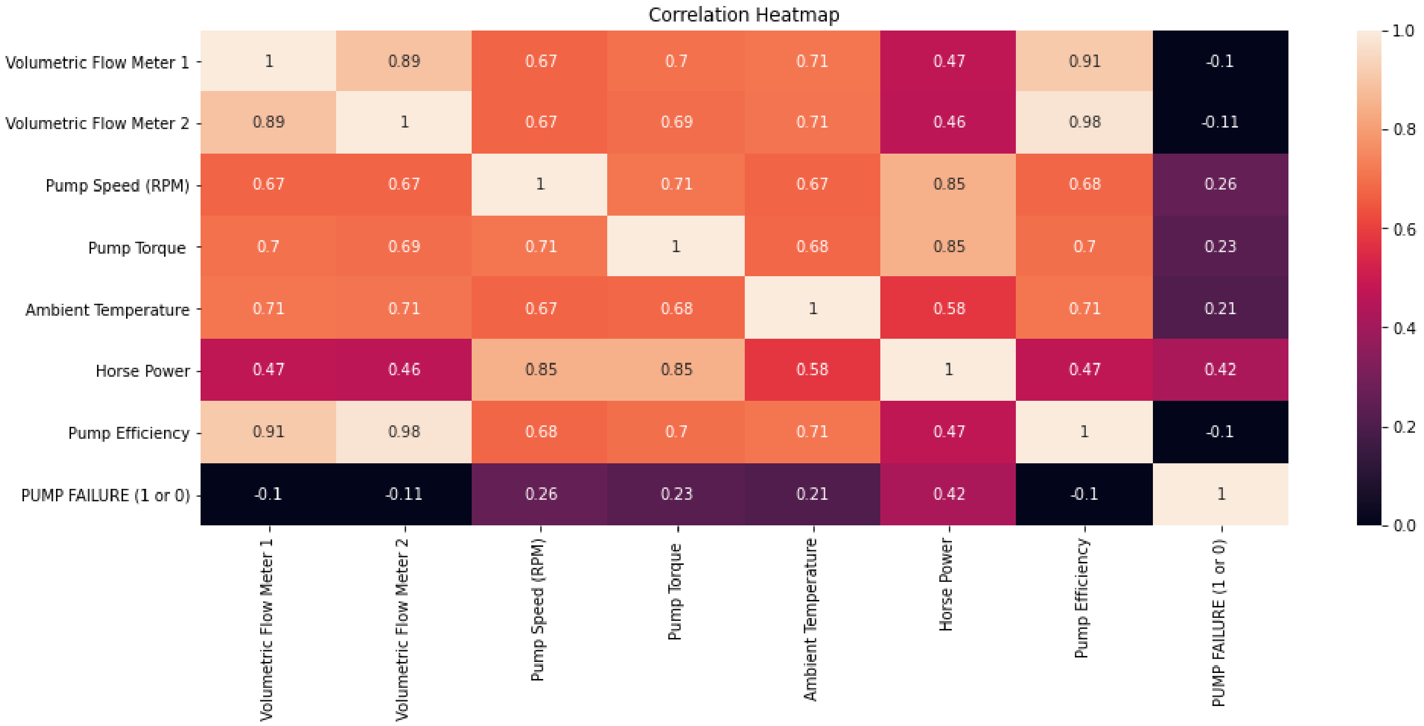
We've included an example of what the output *may* look like below

In [14]:

```
corr_mat = dataframe_raw.corr()
plt.figure(figsize=(16, 6))
heat_map = sns.heatmap(corr_mat, vmin=0, vmax=1, annot=True)
heat_map.set_title('Correlation Heatmap')
```

Out[14]:

```
Text(0.5, 1.0, 'Correlation Heatmap')
```



Open-ended Question:

Which variables seem to correlate with Pump Failure?

In [15]:

```
print('None of the variables seem to correlate with Pump Failure.')
```

```
None of the variables seem to correlate with Pump Failure.
```

Step 11: Create a Barplot of Correlated Features

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

You can do this with the matplotlib library when you specify matplotlib.pyplot(kind='bar')

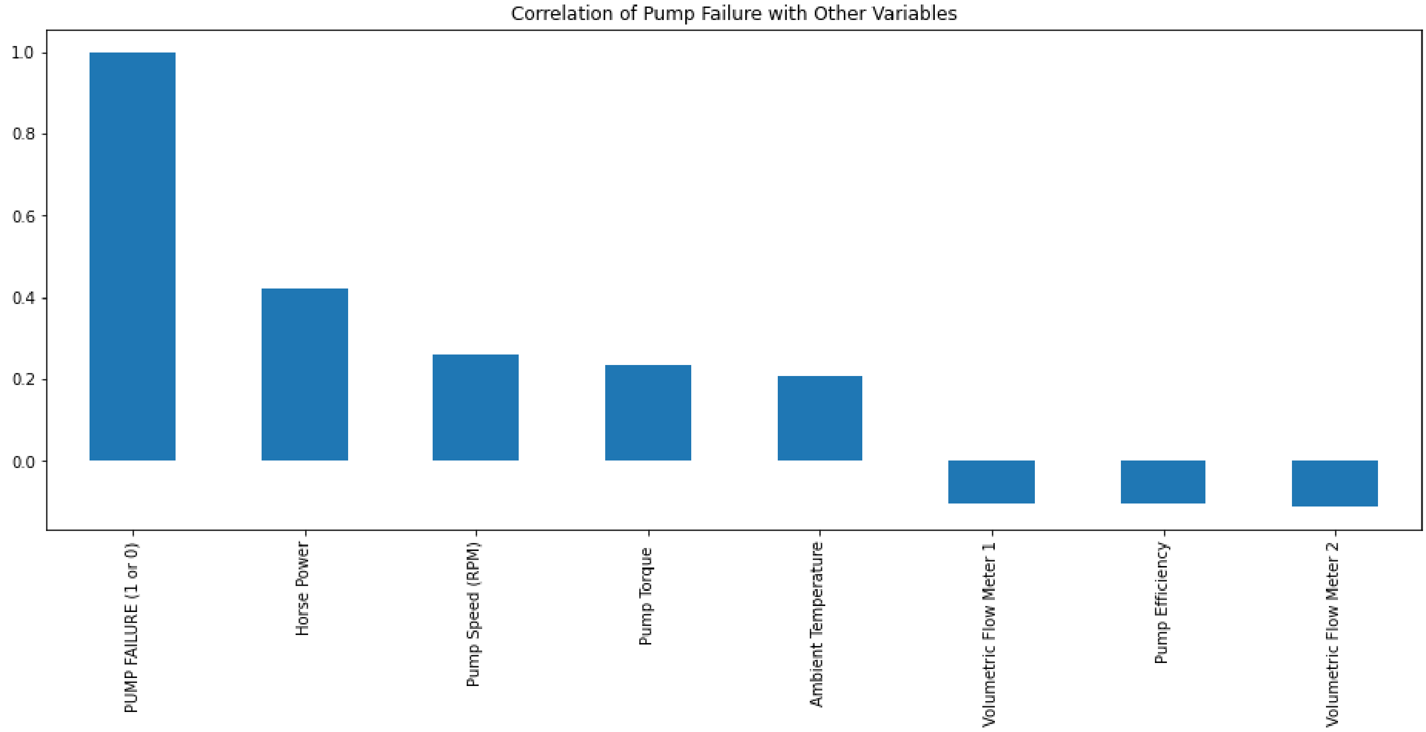
Please put your code here

In [60]:

```
pump_fail_corr = corr_mat['PUMP FAILURE (1 or 0)']
pump_fail_corr_asc = pump_fail_corr.sort_values(ascending=False)
pump_fail_corr_asc.plot(kind='bar', figsize=(16, 6))
plt.title('Correlation of Pump Failure with Other Variables')
```

Out[60]:

```
Text(0.5, 1.0, 'Correlation of Pump Failure with Other Variables')
```



Step 12: Create a Rolling Standard Deviation Heatmap

Previously, you created a correlation matrix using 'raw' variables. We saw *some* correlations with the raw data but they weren't necessarily as strong as we would have liked. This time, we'll recreate a Heatmap using the rolling standard deviation dataframe you had imported in Q1.

ii) Using Seaborn's heatmap function, create a heatmap that clearly shows the correlations (including R Squared) for all variables using the dataframe_stdev dataset.

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

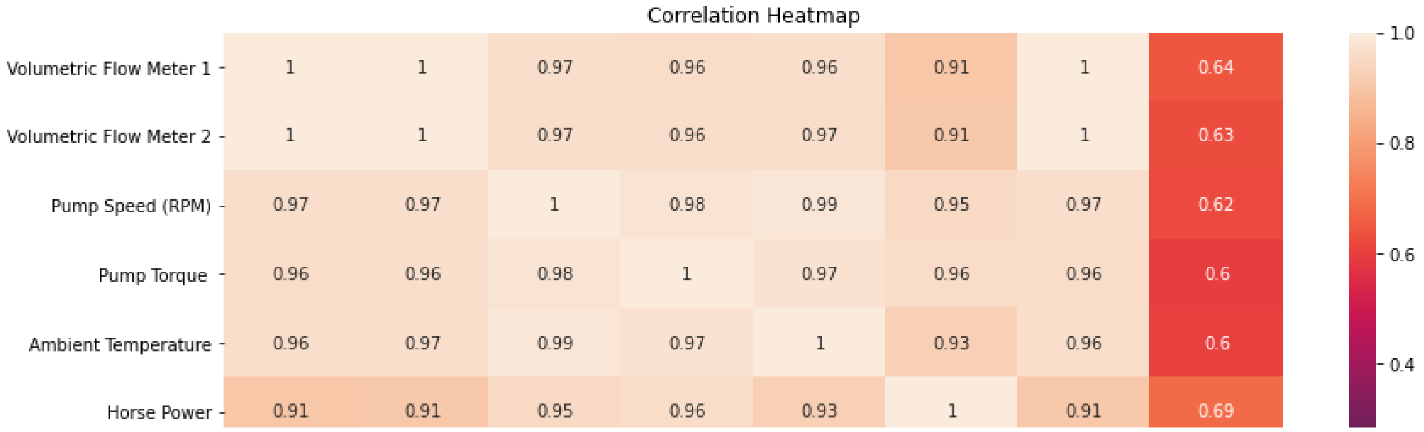
Please put your code here

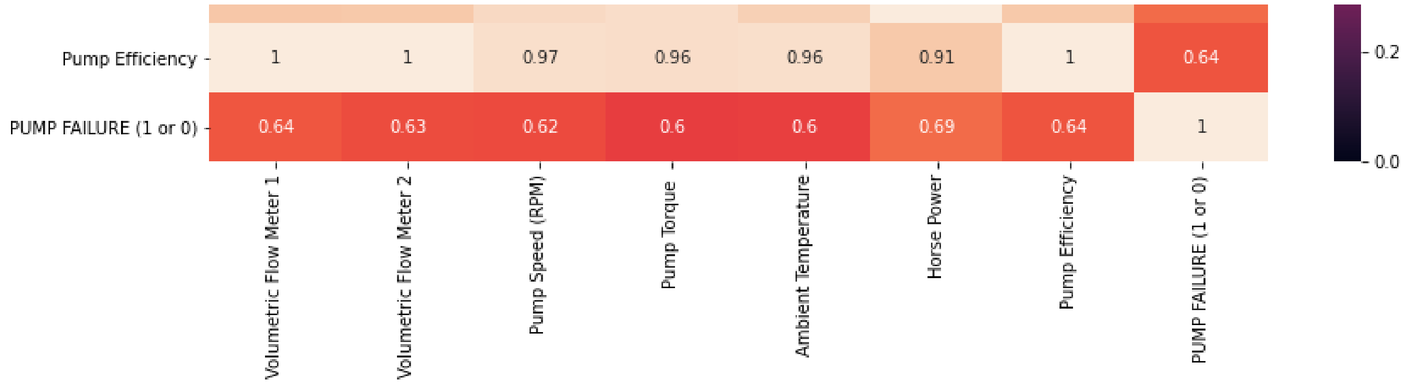
In [62]:

```
corr_mat = dataframe_stdev.corr()
plt.figure(figsize=(14, 6))
heat_map = sns.heatmap(corr_mat, vmin=0, vmax=1, annot=True)
heat_map.set_title('Correlation Heatmap')
```

Out[62]:

```
Text(0.5, 1.0, 'Correlation Heatmap')
```





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 13: 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).

In order to fit a linear regression model with statsmodels.api there are a few steps that need to be taken. We have demonstrated this below:

```
## Add a constant to follow the equation form: Ab + x (X = sm.add_constant(X))
## Instantiate the Ordinary Least Squares Model with: model = sm.OLS(Y,X) where Y
is the dependent variable and X is the independent variable (Make sure you don't in
clude the PUMP FAILURE (1 or 0) in your list of independent variables as this is wh
at you are trying to predict)
## Fit the Model (OLSmodelResult = OLSmodel.fit())
## Print the OLSModel Summary
```

Link: https://www.statsmodels.org/devel/generated/statsmodels.regression.linear_model.OLS.html

ii) Repeat i) but this time use the dataframe_stdev you imported previously. What is the R Squared for the model and what does this signify?

Y = dataframe_raw['PUMP FAILURE (1 or 0)'] X = dataframe_raw[['Volumetric Flow Meter 1', 'Volumetric Flow Meter 2', 'Pump Speed (RPM)', 'Ambient Temperature', 'Horse Power', 'Pump Torque ', 'Pump Efficiency']] X = sm.add_constant(X) model = sm.OLS(Y,X) results = model.fit() results.params print(results.summary())

In [100]:

```
Y = dataframe_raw['PUMP FAILURE (1 or 0)']
X = dataframe_raw[['Volumetric Flow Meter 1', 'Volumetric Flow Meter 2', 'Pump Speed (RPM
)', 'Ambient Temperature', 'Horse Power', 'Pump Torque ', 'Pump Efficiency']]
X = sm.add_constant(X)
model = sm.OLS(Y,X)
results = model.fit()
results.params
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     PUMP FAILURE (1 or 0)   R-squared:                       0.362
Model:                               OLS   Adj. R-squared:                  0.360
Method:                    Least Squares   F-statistic:                     197.9
```

```
Date:                  Mon, 21 Dec 2020   Prob (F-statistic):          5.58e-233
Time:                          18:07:35   Log-Likelihood:                 1823.0
No. Observations:                  2453   AIC:                            -3630.
Df Residuals:                      2445   BIC:                            -3583.
Df Model:                             7
Covariance Type:              nonrobust
==========================================================================================
==
                             coef    std err          t      P>|t|      [0.025      0.9
75]
------------------------------------------------------------------------------------------
----
const                      0.1138      0.028      4.058      0.000       0.059       0.
169
Volumetric Flow Meter 1   -0.0145      0.002     -7.974      0.000      -0.018      -0.
011
Volumetric Flow Meter 2   -0.0184      0.003     -5.497      0.000      -0.025      -0.
012
Pump Speed (RPM)           0.0021      0.001      2.589      0.010       0.001       0.
004
Ambient Temperature        0.0099      0.001     14.159      0.000       0.009       0.
011
Horse Power                0.0827      0.019      4.373      0.000       0.046       0.
120
Pump Torque                0.0007      0.000      1.801      0.072   -6.24e-05       0.
001
Pump Efficiency            0.0020      0.002      1.028      0.304      -0.002       0.
006
==============================================================================
Omnibus:                     2071.669   Durbin-Watson:                   0.458
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            58977.621
Skew:                           3.929   Prob(JB):                         0.00
Kurtosis:                      25.700   Cond. No.                     3.03e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifie
d.
[2] The condition number is large, 3.03e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Open-ended Question:

Which linear regression model seems to be a better fit? Why do you think this is the case?

In [126]:

```
Y = dataframe_stdev['PUMP FAILURE (1 or 0)']
X = dataframe_stdev[['Volumetric Flow Meter 1', 'Volumetric Flow Meter 2', 'Pump Speed (R
PM)', 'Ambient Temperature', 'Horse Power', 'Pump Torque ', 'Pump Efficiency']]
X = sm.add_constant(X)
model = sm.OLS(Y,X)
results = model.fit()
results.params
print(results.summary())
print('Which linear regression model seems to be a better fit? Why do you think this is t
he case?: Since the R-squared of rolling standard deviation data is much higher than for
raw data (0.778 vs 0.362) and the P-values of all variables of rolling standard deviation
are less than 0.05, the multivariate regression of rolling standard deviation represents
indicates a strong correlation.')
```

```
                              OLS Regression Results
==============================================================================
Dep. Variable:     PUMP FAILURE (1 or 0)   R-squared:                       0.778
Model:                               OLS   Adj. R-squared:                  0.778
Method:                    Least Squares   F-statistic:                     1225.
Date:                   Mon, 21 Dec 2020   Prob (F-statistic):               0.00
Time:                           19:30:50   Log-Likelihood:                 3117.4
No. Observations:                   2452   AIC:                            -6219.
Df Residuals:                       2444   BIC:                            -6172.
```

```
Df Residuals:                    2444   BIC:                            -6172.
Df Model:                           7
Covariance Type:            nonrobust
==========================================================================================
==
                              coef    std err          t      P>|t|      [0.025      0.9
75]
------------------------------------------------------------------------------------------
----
const                      -0.0354      0.004    -10.045      0.000      -0.042      -0.
029
Volumetric Flow Meter 1     0.0513      0.010      5.109      0.000       0.032       0.
071
Volumetric Flow Meter 2    -0.6085      0.020    -30.450      0.000      -0.648      -0.
569
Pump Speed (RPM)           -0.0178      0.002     -9.057      0.000      -0.022      -0.
014
Ambient Temperature         0.0178      0.003      5.773      0.000       0.012       0.
024
Horse Power                 0.7636      0.022     35.045      0.000       0.721       0.
806
Pump Torque                -0.0189      0.001    -22.511      0.000      -0.020      -0.
017
Pump Efficiency             0.3407      0.012     28.118      0.000       0.317       0.
364
==============================================================================
Omnibus:                     1432.369   Durbin-Watson:                   0.242
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            72092.923
Skew:                           2.052   Prob(JB):                         0.00
Kurtosis:                      29.245   Cond. No.                         426.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifie
d.
Which linear regression model seems to be a better fit? Why do you think this is the case
?: Since the R-squared of rolling standard deviation data is much higher than for raw dat
a (0.778 vs 0.362) and the P-values of all variables of rolling standard deviation are le
ss than 0.05, the multivariate regression of rolling standard deviation represents indica
tes a strong correlation.
```

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

Step 14: 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.

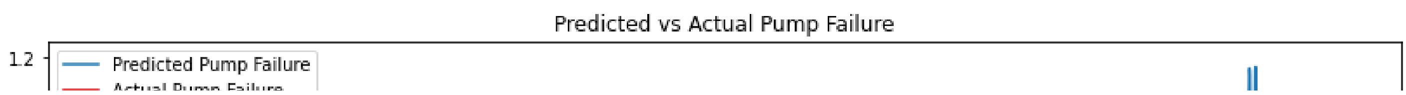
Please put your code here

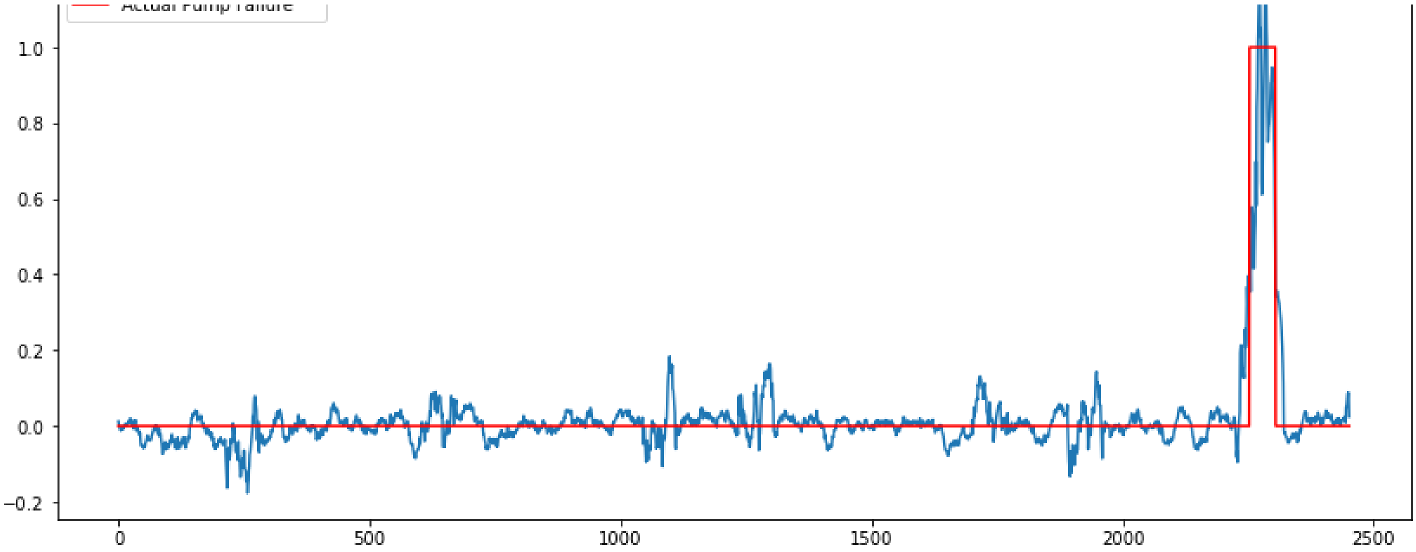
In [149]:

```
yPrediction = results.predict(X)
plt.figure(figsize=(14, 6))
plt.plot(yPrediction, label='Predicted Pump Failure')
plt.plot(Y, label='Actual Pump Failure', color ='red')
plt.title('Predicted vs Actual Pump Failure')
plt.legend(loc=2)
```

Out[149]:

```
<matplotlib.legend.Legend at 0x279d549c4f0>
```





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 proficient with the fundamentals of data analysis in Python!

In [ ]:

```

```