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.

You might have realised that while Excel is powerful, it was a bit tricky to use for visualizations like Box Plots, multiple line plots and in general, there are challenges that arise when doing quick analysis.

In this Case Study, you're going to *revisit* the Pump Data, with a few different questions that help you see how you can use Python to speed up your analysis. You'll also witness how powerful Python's plotting and modelling libraries can be!

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.

Time to get started!

Part I: Descriptive Statistics

Step 1: Import Libraries

Import 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. 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.

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.

Please note we've included the mpl.rcParams['figure'figsize'] code for you.

This code controls how big your charts will be with the syntax of (X,Y) where X represents the X Axis and Y, the Y Axis.

Place your code here

In [2]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import pandas as pd
import statsmodels.api as sm
sns.set_style("darkgrid")
mpl.rcParams['figure.figsize'] = (20,5)
```

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 Question 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.

Please complete the following below:

i. Import each of the two data sources and store them into their individual DataFrames.

We suggest you use the names : **dataframe_raw & dataframe_stdev respectively.**.

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

In [3]:
```
dataframe_raw = pd.read_csv('DF_Raw_Data.csv')
dataframe_stdev = pd.read_csv('DF_Rolling_Stdev.csv')
dataframe_raw.describe()
```

Out[3]:

	Volumetric Flow Meter 1	Volumetric Flow Meter 2	Pump Speed (RPM)	Pump Torque	Ambient Temperature	Horse Power	Pump Efficiency	PUMP FAILURE (1 or 0)
count	2453.000000	2453.000000	2453.000000	2453.000000	2453.000000	2453.000000	2453.000000	2453.000000
mean	41.802629	41.796702	90.796576	202.851610	50.226661	3.540897	76.015149	0.021199
std	3.656576	3.654873	10.217885	22.683977	5.298203	0.579055	6.651633	0.144075
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	41.050000	41.000000	85.000000	191.000000	48.000000	3.220000	74.560000	0.000000
50%	42.100000	42.140000	91.000000	203.000000	51.000000	3.480000	76.620000	0.000000
75%	43.140000	43.130000	97.000000	215.000000	53.000000	3.780000	78.470000	0.000000
max	45.900000	45.840000	124.000000	264.000000	65.000000	7.560000	83.450000	1.000000

In [4]:

```
dataframe_raw.info()
```

```
<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
```

When looking at the Descriptive Statistics for both datasets; pay attention specifically to the standard deviation and mean. What do you observe when you compare the dataframe_raw standard deviation and mean, versus the dataframe_stdev standard deviation and mean?

**Place your answer below. **

In [5]:
```
dataframe_raw.head()
```

Out[5]:

	Data Source	TIMEFRAME (DD/MM/YYYY)	Volumetric Flow Meter 1	Volumetric Flow Meter 2	Pump Speed (RPM)	Pump Torque	Ambient Temperature	Horse Power	Pump Efficiency	PUMP FAILURE (1 or 0)
0	Raw	9/12/2014 0:00	41.30	41.16	98	207	54	3.86	74.84	0
1	Raw	9/12/2014 0:01	42.40	41.39	92	212	46	3.71	75.25	0
2	Raw	9/12/2014 0:02	41.43	41.15	80	207	55	3.15	74.82	0
3	Raw	9/12/2014 0:03	42.21	40.93	83	190	49	3.00	74.42	0
4	Raw	9/12/2014 0:04	40.51	43.32	90	195	50	3.34	78.76	0

In [11]:
```
print("The maximum numbers seem to be ~16x higher for volumetric flow meter 1 and for Horse Power")
dataframe_stdev.describe()
```

```
The maximum numbers seem to be ~16x higher for volumetric flow meter 1 and for Horse Power
```

Out[11]:

	Volumetric Flow Meter 1	Volumetric Flow Meter 2	Pump Speed (RPM)	Pump Torque	Ambient Temperature	Horse Power	Pump Efficiency	PUMP FAILURE (1 or 0)
count	2452.000000	2452.000000	2452.000000	2452.000000	2452.000000	2452.00000	2452.000000	2452.000000
mean	1.485126	1.497361	6.648308	13.945338	3.436370	0.37060	2.725232	0.021207
std	2.294950	2.282053	5.722897	12.394302	3.043042	0.29979	4.186723	0.144104
min	0.380000	0.640000	0.580000	5.000000	0.900000	0.11000	1.170000	0.000000
25%	1.070000	1.080000	5.520000	11.210000	2.920000	0.28000	1.960000	0.000000
50%	1.160000	1.170000	5.990000	12.180000	3.160000	0.32000	2.120000	0.000000
75%	1.230000	1.260000	6.460000	13.110000	3.370000	0.36000	2.270000	0.000000
max	21.390000	21.530000	59.310000	124.710000	30.650000	3.32000	39.150000	1.000000

In [12]:
```
print("Analysisng what type of values is the data comprised of and what the first 5 rows look like")
dataframe_stdev.info()
dataframe_stdev.head()
```

```
Analysisng what type of values is the data comprised of and what the first 5 rows look like
<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
```

Out[12]:

	Data Source	TIMEFRAME (DD/MM/YYYY)	Volumetric Flow Meter 1	Volumetric Flow Meter 2	Pump Speed (RPM)	Pump Torque	Ambient Temperature	Horse Power	Pump Efficiency	PUMP FAILURE (1 or 0)
0	Rolling Stdev (30 Minute)	9/12/2014 0:00	1.04	0.96	5.54	11.70	3.40	0.32	1.74	0
1	Rolling Stdev (30 Minute)	9/12/2014 0:01	1.06	1.01	5.49	11.73	3.36	0.31	1.83	0
2	Rolling Stdev (30 Minute)	9/12/2014 0:02	1.06	1.03	5.62	11.94	3.40	0.31	1.87	0
3	Rolling Stdev (30 Minute)	9/12/2014 0:03	1.06	1.05	5.61	12.10	3.30	0.31	1.90	0
4	Rolling Stdev (30 Minute)	9/12/2014 0:04	1.07	1.03	5.61	12.31	3.36	0.30	1.88	0

Step 3: Create a Boxplot

When you look at your dataframe, you should be able to see the upper and lower quartiles for each row of data from when you used .describe.

This gives you an understanding 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. With that in mind, convert the DataFrames into a boxplot by following these instructions:

Please repeat these steps for both the dataframe_raw dataset and the dataframe_stdev dataset

i) Using the DataFrame, create a boxplot visualising this data.

ii) Using the DataFrame, create a lineplot visualising this data.

An example of the syntax to consider would be:

```
dataframe.plot(kind='box')
plt.show()
```

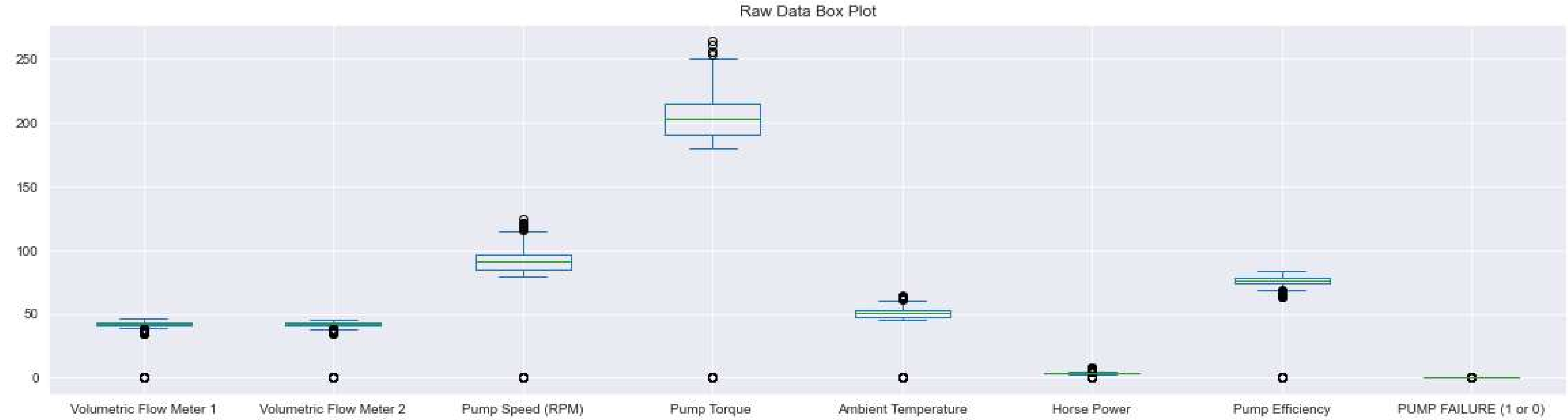
Hint: You might want to reference the following .plot function here

Please put your code here

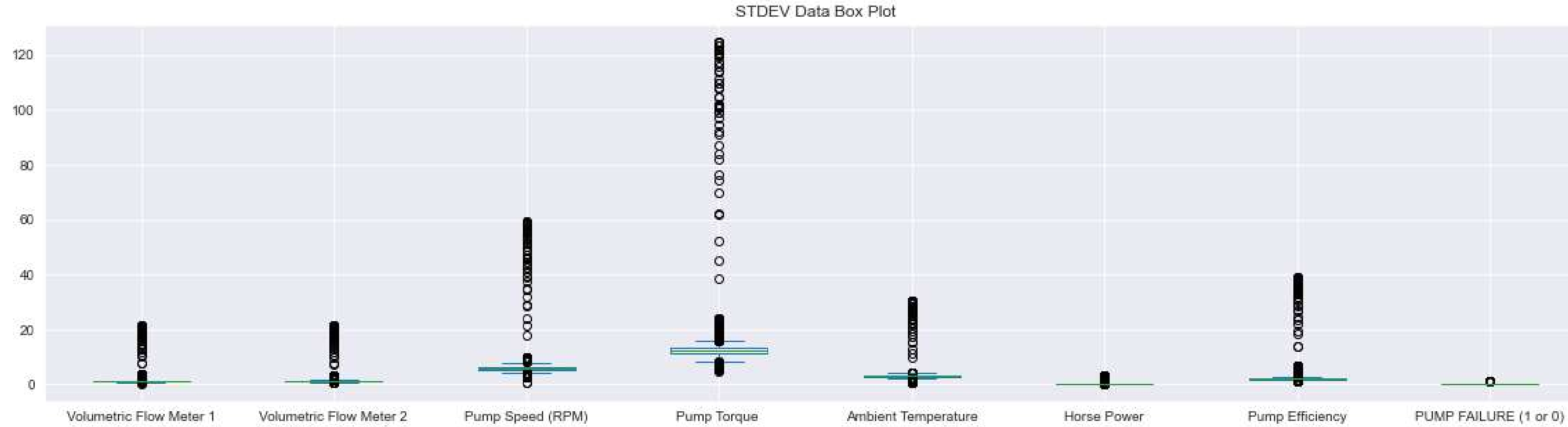
We've included an example of what your Box Plot and Line Plot *should* look like once you've plotted this using the dataframe_raw and dataframe_stdev datasets.

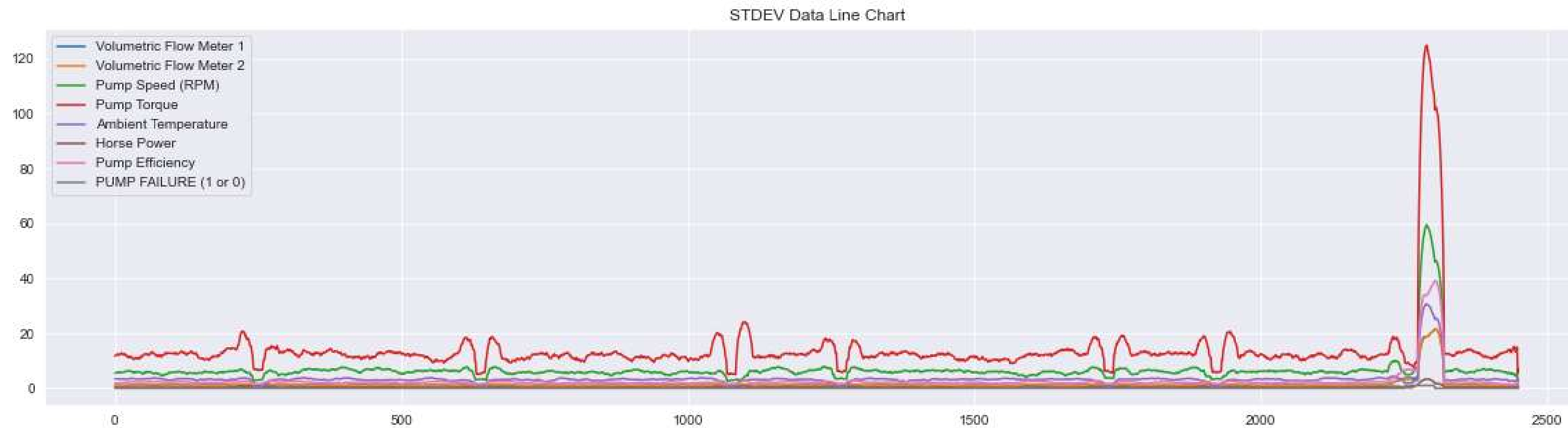
In [8]:

```
dataframe_raw.plot(kind='box', title='Raw Data Box Plot')
dataframe_raw.plot(kind='line', title='Raw Data Line Chart')
plt.show()
```



In [9]:
```
dataframe_stdev.plot(kind='box', title='STDEV Data Box Plot')
dataframe_stdev.plot(kind='line', title='STDEV Data Line Chart')
plt.show()
```





What have you observed from the boxplot and line plots for both the dataframe_raw and dataframe_stdev datasets?

In [13]:
```
print("I noticed that the standard deviation has a thin range in the interquartile range and many outliers, whereas the
```

```
I noticed that the standard deviation has a thin range in the interquartile range and many outliers, whereas the raw da
ta has a wider interquartile range. I also noticed a pattern of gentle rises in the data during functional periods, and
a large spike in both the raw data and the standard deviation when the pump is broken.
```

Step 4: Filtered DataFrames with Box Plots

You would have noted the datasets we have contain a Pump Failure (1 or 0) Variable that indicate whether the pump is failing (1) or whether the pump is behaving normally (0).

It is very likely you will see differences in the overall populations of Pre-Failure vs. Post Failure. To visualise this difference, you should separate your datasets, filtering for when the Pump Failure = 0 or when the Pump Failure = 1, and see what trends emerge.

This will require you to SUBSET your dataframe using *boolean filters*.

We've included an example below to show-case how this syntax works:

```
condition_1 = dataframe['SomeColumn']== 1
dataframe_0 = dataframe[condition_1]
dataframe.plot(kind='box')
plt.title("Example Plot")
plt.show()
```

Please repeat these steps for both the dataframe_raw dataset and the dataframe_stdev dataset

i) Using the dataframe_raw dataset, create two boxplots specifically for when the pump has failed and when the pump is working normally.

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.

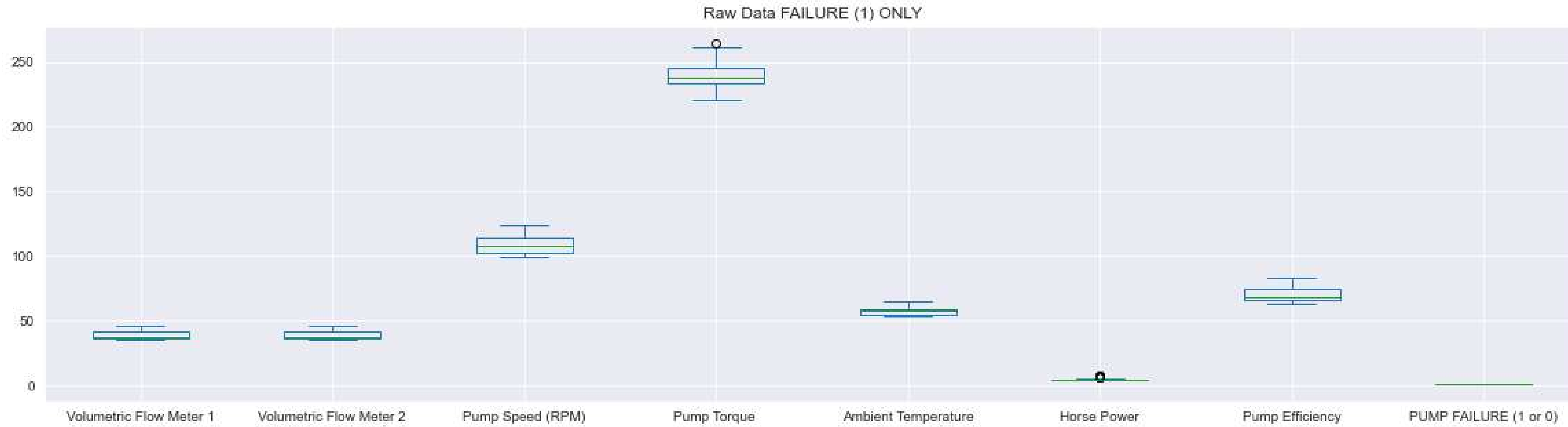
To do this, you'll have to recall how to apply boolean filtering to a dataframe. If you're not sure how to do this, re-read the hints we've given above and pay careful attention to the syntax we've shown.

Please put your code here

In [14]:

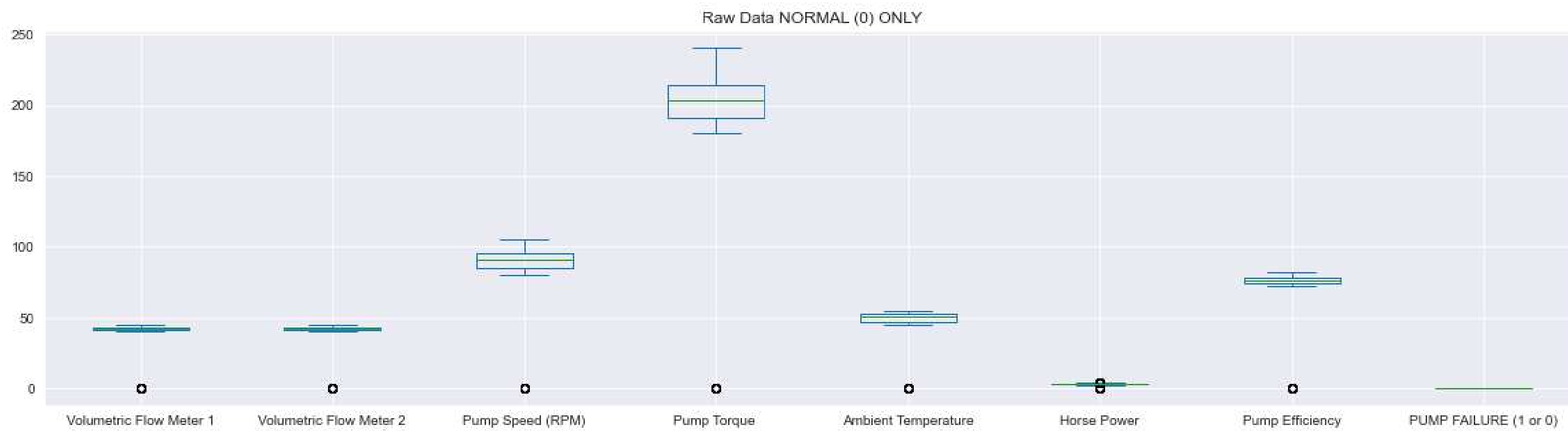
```
#Failure Raw Box Plot
Cond1 = dataframe_raw['PUMP FAILURE (1 or 0)'] == 1
dataframe_raw_FAIL = dataframe_raw[Cond1]
dataframe_raw_FAIL.plot(kind='box')
```

```
plt.title("Raw Data FAILURE (1) ONLY")
plt.show()
```



In [16]:

```
#Normal Raw Box Plot
Cond0 = dataframe_raw['PUMP FAILURE (1 or 0)'] == 0
dataframe_raw_NORMAL = dataframe_raw[Cond0]
dataframe_raw_NORMAL.plot(kind='box')
plt.title("Raw Data NORMAL (0) ONLY")
plt.show()
```

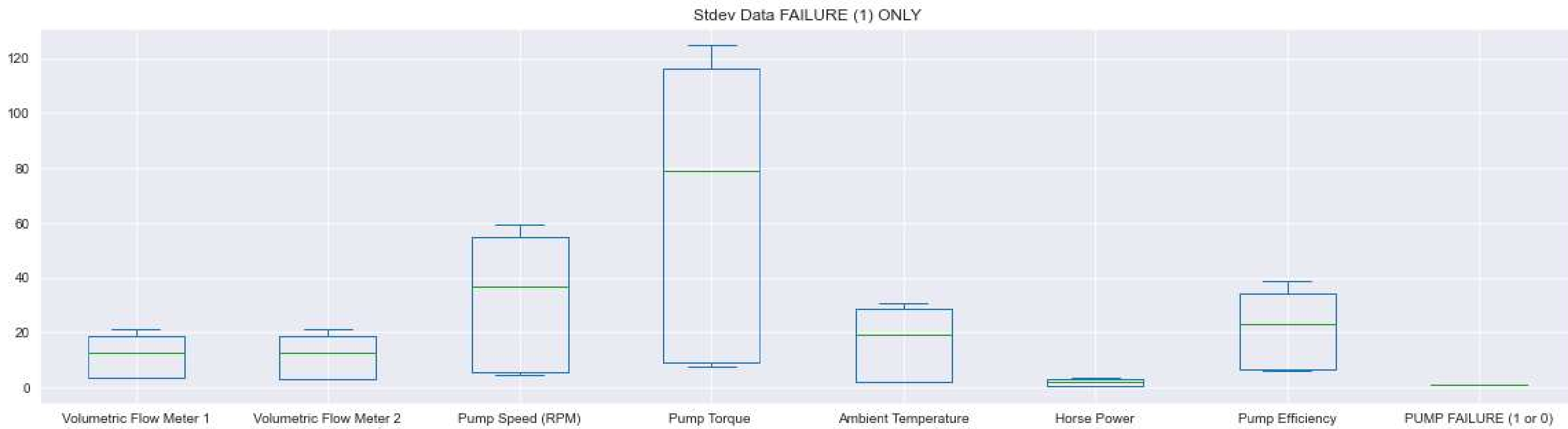


In [18]:

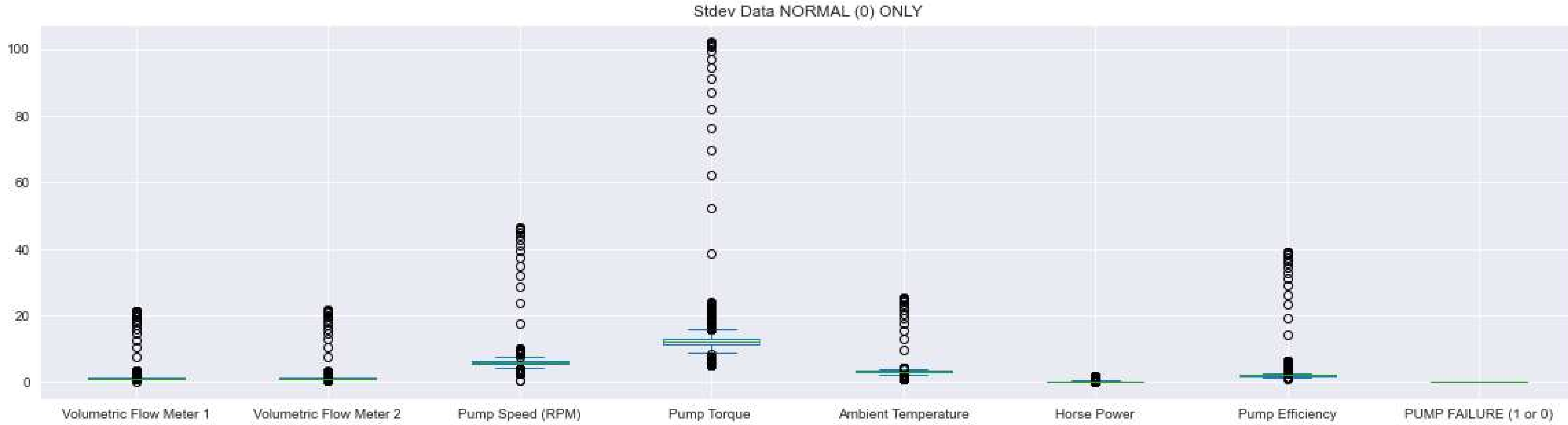
```
print("It's hard to say that there is a noticeably huge difference, however some variables do increase. When viewing Pu
```

It's hard to say that there is a noticeably huge difference, however some variables do increase. When viewing Pump Torque closely, it shows that the top value for failure (1) has a maximum value that is higher than when compared to the normal (0) value. When viewing Pump Speed (RPM), Failure ranges appear to be higher than outside the 75th percentile of the normal box plot when comparing the two figures closely.

In [19]:
```
#Failure STDEV Box Plot
Cond1 = dataframe_stdev['PUMP FAILURE (1 or 0)'] == 1
dataframe_stdev_FAIL = dataframe_stdev[Cond1]
dataframe_stdev_FAIL.plot(kind = 'box')
plt.title("Stdev Data FAILURE (1) ONLY")
plt.show()
```



In [20]:
```
#Normal STDEV Box Plot
Cond0 = dataframe_stdev['PUMP FAILURE (1 or 0)'] == 0
dataframe_stdev_NORMAL = dataframe_stdev[Cond0]
dataframe_stdev_NORMAL.plot(kind='box')
plt.title("Stdev Data NORMAL (0) ONLY")
plt.show()
```



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

In [22]:
```
print("The Stdev gives a much greater contrast, when compared to the Raw Data Box Plots.For all the variables within th
```

```
The Stdev gives a much greater contrast, when compared to the Raw Data Box Plots.For all the variables within the box p
lots, the outliers of Normal (0) only apepar to be the typical range of Failure. This could indicate that a combination
of the values - Pump Speed (RPM), Pump Torque, Volumetric Flow Meter 1 & 2. If the Stdev's hit a certain threshold, the
se values can be used as triggers for warning notifications to shut doen the pumps before they lose control.
```

A quick analysis might show-case there are many *outliers* that exist in your dataset.

As you all know, outliers can easily *skew* your analysis. You might want to remove them.

The general logic for removing an outliers is:

Any Value that is either Q1 - 1.5x IQR or greater than Q3 + 1.5x IQR is considered an outlier.

Where Q1 = Quartile 1 and Q3 = Quartile 3

Let's break these down to a few simple steps and tackle how to calculate this.

1. **Define the Quartiles Q1 and Q3.**
2. **Calculate the Interquartile Range (Q3 - Q1)**
3. **Create two new variables; Lower_Range (Q1 - 1.5x IQR) and Upper_Range (Q3 + 1.5x IQR).**

4. **Using Boolean Subsetting, filter the DataFrame for outliers and remove them.**
5. **Calculate what the proportion of outliers exist (i.e. Number of entries left after outlier removal / Number of total entries in dataset).**

Step 5: Create Quartiles

Create two new variables called Q1 and Q3 using the dataframe_raw dataset.

i) Q1 should contain the 25th percentile for all columns in the DataFrame. Q3 should contain the 75th percentile for all the columns in the DataFrame.

You may want to use the .quantile() function explained here.

ii) After defining Q1 and Q3, calculate the interquartile range (IQR = Q3 - Q1) for all columns in the DataFrame and print it to the screen.

We've included a sample of the IQR output below for your reference!

Please put your code here

In [31]:
```
Q1 = dataframe_raw.quantile(0.25)
Q3 = dataframe_raw.quantile(0.75)
Raw_IQR = Q3 - Q1
print(Raw_IQR)
```

```
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

Continuing in from Step 5 - we still need to calculate how we can remove our outliers.

We're aware of the overall formula as shown per below.

Outlier = Q1 - 1.5x IQR OR Q3 + 1.5x IQR

Now work out how to identify these outliers step by step.

i) Define two new variables, Lower_Limit and Upper_Limit can be calculated as:

- **Lower_Limit = Q1 - 1.5 * IQR**
- **Upper_Limit = Q3 + 1.5 * IQR**

ii) Using Lower_Limit and Upper_Limit with an OR (|) condition, filter the DataFrame to include *only* the outliers. How many outliers do we have?

The syntax should look like this:

```
Outliers = some_dataframe [ ((some_dataframe < Lower_Limit) | ((dataframe_raw >
Upper_Limit))).any(axis=1) ]
```

The .any(axis=1) indicates that for *any* entry in the columns that matches the criteria of being *less* than the lower limit or *greater* than the upper limit, for that column (axis=1), that row will be removed.

If you're stuck, here's a link that will show you how you can remove outliers in Python!

iii) What is the percentage of data that remains after we've removed the outliers from the dataframe_raw dataset?

Please put your code here

In [78]:
```
# Lower & Upper Limit
lower_limit= Q1 - 1.5*IQR
upper_limit= Q3 + 1.5*IQR
lower_limit
```

Out[78]:

	Ambient Temperature	Horse Power	PUMP FAILURE (1 or 0)	Pump Efficiency	Pump Speed (RPM)	Pump Torque	Volumetric Flow Meter 1	Volumetric Flow Meter 2	index
0	40.5	2.38	0.0	68.695	67.0	155.0	37.915	37.805	NaN

In [79]:

```
upper_limit
```

Out[79]:

	Ambient Temperature	Horse Power	PUMP FAILURE (1 or 0)	Pump Efficiency	Pump Speed (RPM)	Pump Torque	Volumetric Flow Meter 1	Volumetric Flow Meter 2	index
0	60.5	4.62	0.0	84.335	115.0	251.0	46.275	46.325	NaN

In [ ]:

```
Outliers=dataframe_raw[((dataframe_raw<lower_limit) | (dataframe_raw>upper_limit)).any(axis=1)]
```

In [81]:

```
data_left_after_outliers= len(dataframe_raw)-len(outliers)
print("Total number of rows in the dataset:" ,len(dataframe_raw))
print("Total Outliers:" ,len(outliers))
print("% of data left after removing outliers:" , data_left_after_outliers/len(dataframe_raw)*100)
print("Total no of rows after removing outliers:", data_left_after_outliers)
```

```
Total number of rows in the dataset: 2453
Total Outliers: 95
% of data left after removing outliers: 96.12719119445576
Total no of rows after removing outliers: 2358
```

You've removed the outliers and still have a significant amount of data left. Do you think removing outliers is problematic or not? Dtate your answer below and explain your thoughts.

In [109…

```
print("There are 95 outliers, that means 96% of data left after removing outliers. It is a terrible idea to remove outl
```

```
There are 95 outliers, that means 96% of data left after removing outliers. It is a terrible idea to remove outliers in
this case. If there are a lot of outliers, then it could mean your data is too messy to draw conclusions from. On the o
ther hand, sometimes it's necessary to remove outliers because it skews the data in a way that is unrepresentative of t
he big picture. It's important to investigate the potential cause of the outlier and make your best judgement call whet
her to include that outlier data or remove it. We are looking for anomolous readings, so removing outliers will make it
much harder to find when things are going wrong. If we includede outliers for the failure case, it would never include
any of the failures. It would only show the results when failure = 0
```

In the previous exercise, we've showed you how to calculate outliers. Now you want to remove *all* the outliers in your dataset and create box plots to see how the data looks without outliers.

We've defined our outliers DataFrame as per below:

```
Outliers = some_dataframe [ ((some_dataframe < Lower_Limit) | ((dataframe_raw >
Upper_Limit))).any(axis=1) ]
```

We're now going to add one symbol that will help you return a dataframe with 0 Outliers.

**This is the ~ Symbol. **

Essentially this Symbol tells Python to 'invert' the current boolean Value from True, to False.

An example use of this syntax would be:

```
some_dataframe = some_dataframe[ ~ ( (some_condition) ) ]
```

Step 7: Create a box plot without outliers

i) Create a new DataFrame called no_outliers and using the ~ operator, remove all the outliers from the DataFrame

ii) Using the no_outliers dataframe, create two box plots as per below:

- **A boxplot when PUMP FAILURE is 1 (Failure)**
- **A boxplot when PUMP FAILURE is 0 (Normal Behaviour)**

Did any of your plots reveal nothing to plot? If yes, why might that be...? Think very carefully regarding what you have performed.

Please put your code here

In [ ]:

```
PumpFailureFilter=no_outliers['PUMP FAILURE (1 or 0)']==1
PumpFaildf=no_outliers[PumpFailureFilter]
PumpFaildf.plot(kind='box',title='Box Plot during Pump Failure')
plt.show()
```

```
PumpNormalFilter=no_outliers['PUMP FAILURE (1 or 0)']==0
PumpNormaldf=no_outliers[PumpNormalFilter]
PumpNormaldf.plot(kind='box',title='Box Plot during Pump Operations')
plt.show()
```

Please note that Step 8 and Step 9 are Challenge Questions and will intentionally be more difficult

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 common when variables are at different scales, the trends can be more challenging to interpret. This is why we are going to ITERATE through the DataFrame, plot each individual variable out, and compare this with the Pump Failure for a more streamlined analysis.

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

1. **Define a list variable called ListOfVariables; this is to contain the column names of all the numerical variables you wish to iterate through in the dataframe_raw dataset**

2. **Instantiate your for loop with the following syntax:**

```
for item in ListOfVariables:
    first_axis = dataframe[___].plot #Looping through every item in the dataframe.
    second_axis = first_axis.twinx() #The Twinx function is used to ensure we share the X-Axis for
both plots
    second_axis.plot(dataframe['ColumnOfInterest'], color='teal')
    plt.title(item)
    plt.show()
```

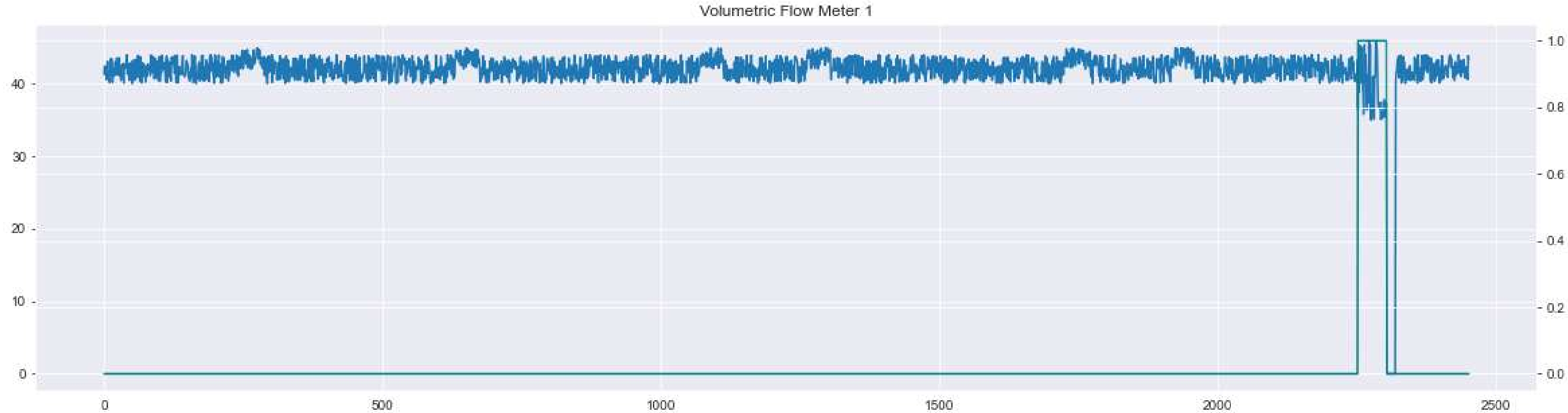
i) Using the syntax provided, loop through the dataframe_raw dataset, plotting every variable individually, against the Pump Failure to better identify trends.

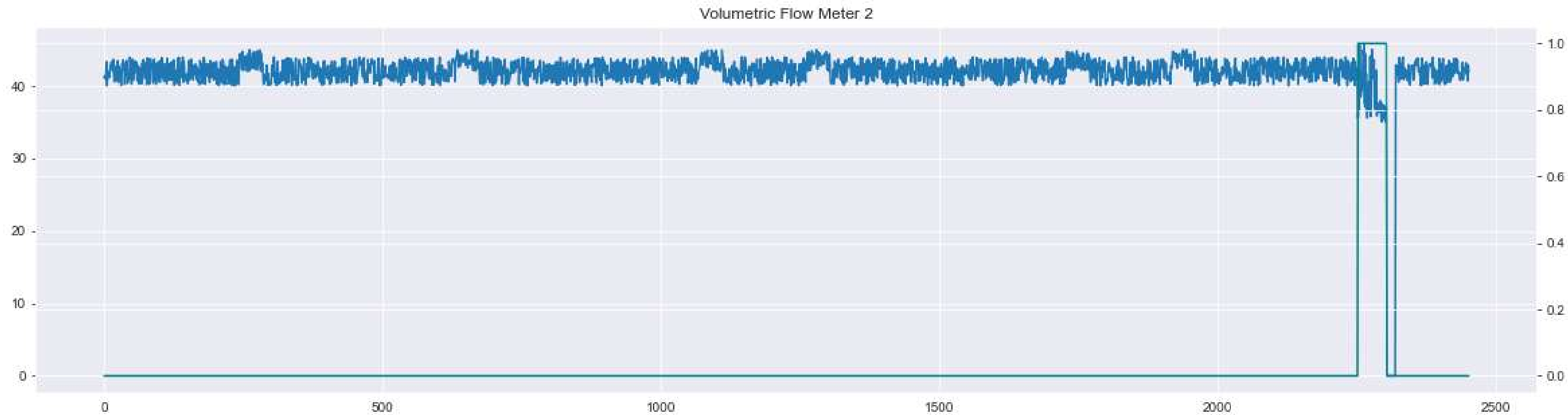
Note: For each plot, ensure that you have a dual axis set up so you can see the Pump Failure (0 or 1) on the second Y-axis, and the attribute on the first Y-Axis.

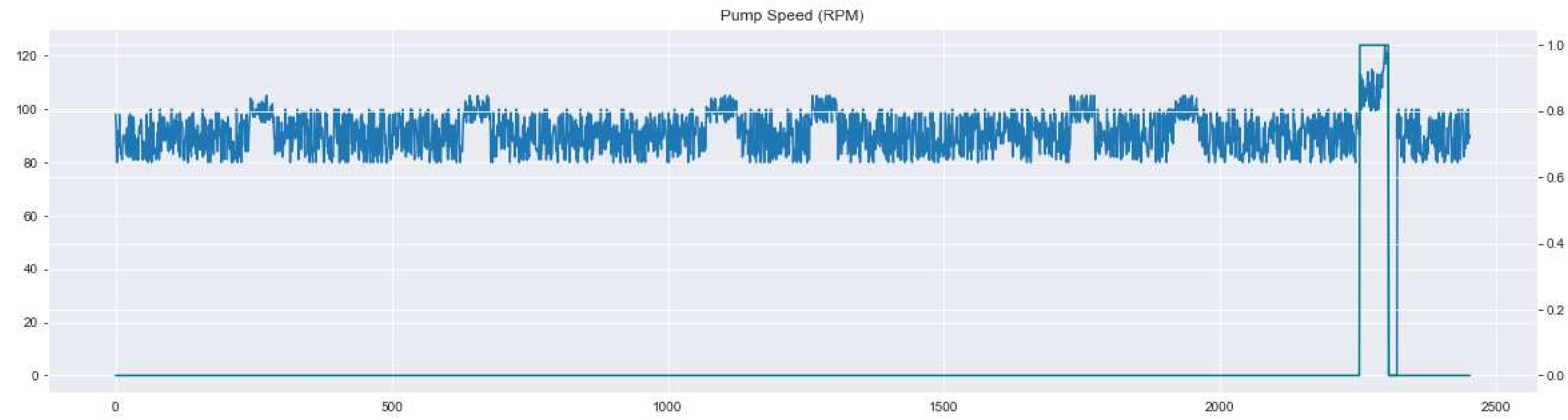
Check out this link to learn how to do this: https://matplotlib.org/gallery/api/two_scales.html

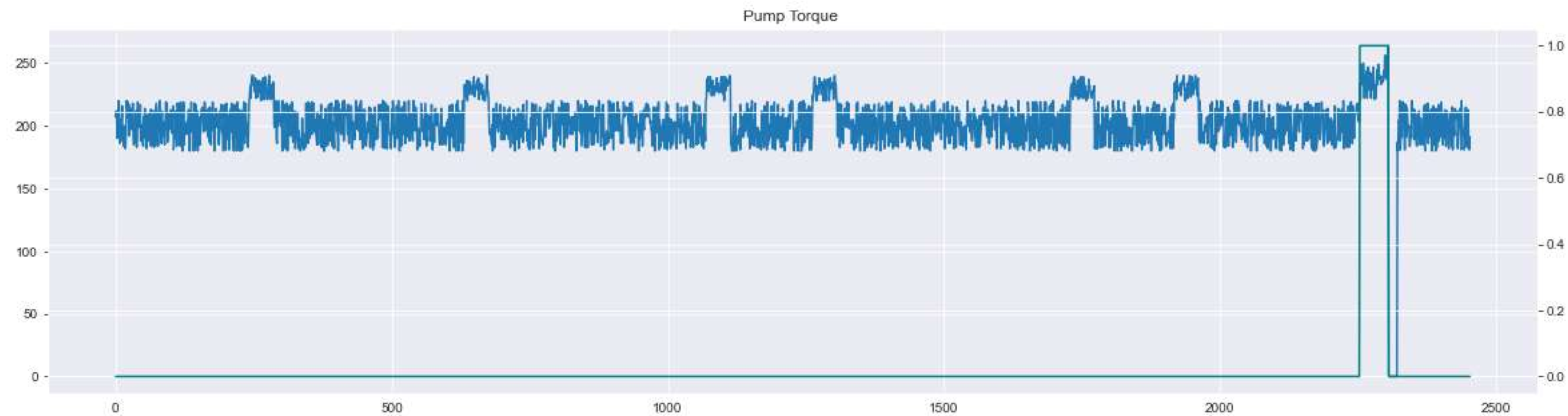
Please put your code here

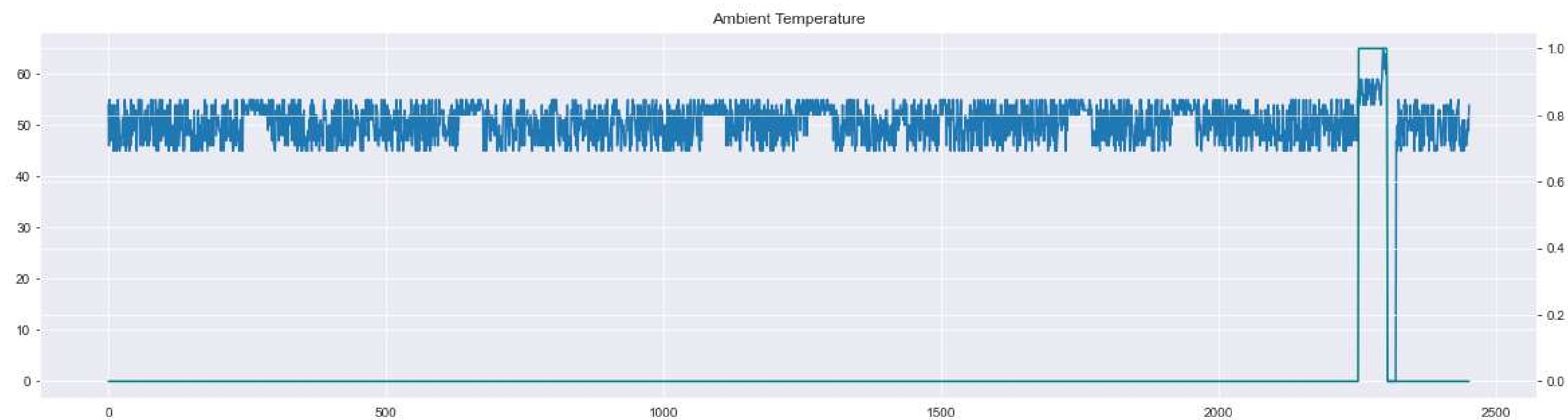
In [67]:
```
ListOfVariables=['Volumetric Flow Meter 1',
       'Volumetric Flow Meter 2', 'Pump Speed (RPM)', 'Pump Torque ',
       'Ambient Temperature', 'Horse Power', 'Pump Efficiency']
for item in ListOfVariables:
        first_axis = dataframe_raw[item].plot()
        second_axis = first_axis.twinx()
        second_axis.plot(dataframe_raw['PUMP FAILURE (1 or 0)'], color='teal')
        plt.title(item)
        plt.show()
```

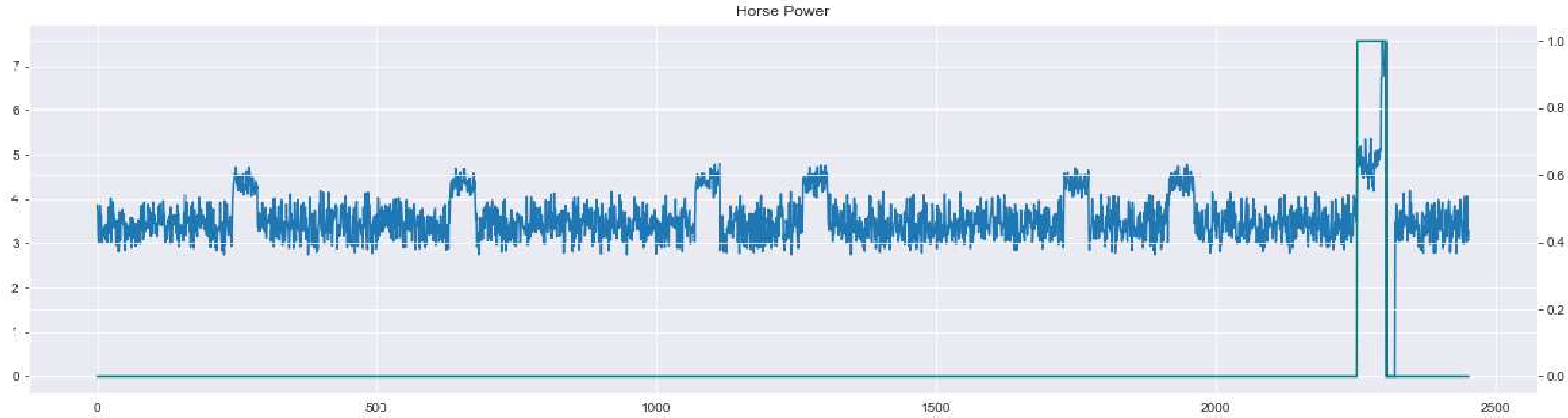


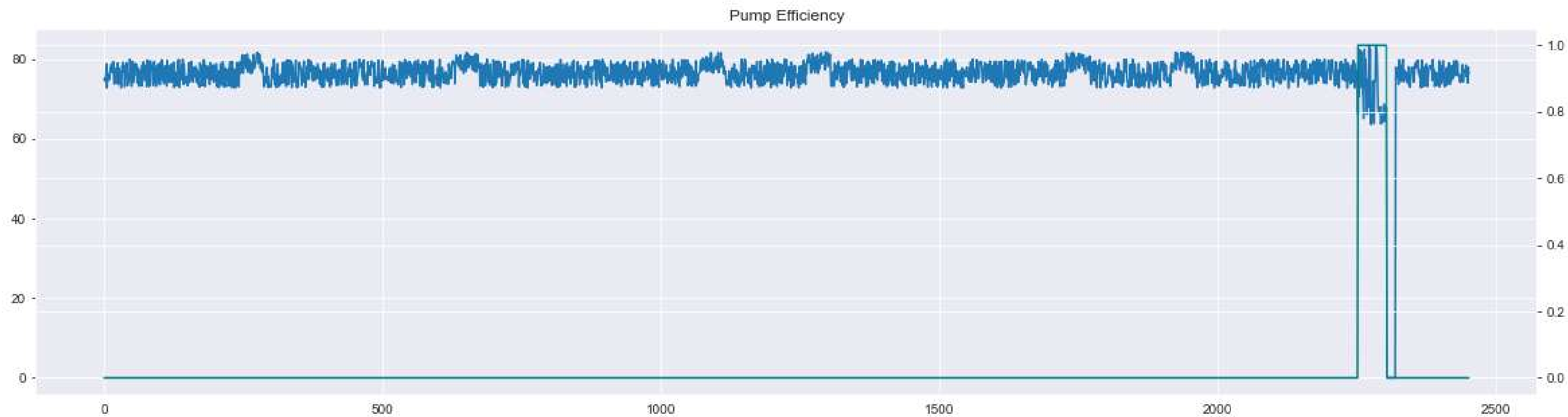



Pump Speed (RPM)
120
100
80
60
40
20
0
0
500
1000
1500
2000
2500
1.0
0.8
0.6
0.4
0.2
0.0


Pump Torque
250
200
150
100
50
0
0
500
1000
1500
2000
2500
1.0
0.8
0.6
0.4
0.2
0.0


Ambient Temperature
60
50
40
30
20
10
0
0
500
1000
1500
2000
2500
1.0
0.8
0.6
0.4
0.2
0.0





Have you noticed any particular trends when looking at the data in this way? Has it made it easier to see which variables *might* be reacting more strongly to the Pump Failure than others?

In [68]:
```
#We can see that there are a few variables to *increase* their value during the pump failure
#This view makes it easier to see which variables might react more to the pump failure than the others.
#The variables to react strongly would be Horse Power, Pump Speed (RPM), Pump Torque, and Ambient Temperature
```

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

---

Now that you've iterated through the dataframe_raw, we're going to do something similar with the dataframe_stdev dataset.

In these next few exercises you'll be seeking to better understand how to make use of Pythons Powerful Inferential Statistics and Plotting libraries to Zoom In on periods of interest that you'd like to examinate further.

First, prepare the DataFrame with the TIMEFRAME (DD/MM/YYYY) column set as the index.

You might ask: What is an Index?

An index is used as a unique identifier for each record in the dataset. This is used primarily for JOINING operations (think SQL joins), or in our case, for filtering on a specific time period.

You'll be making use of the following function:

```
dataframe.set_index(['someColumn', inplace=True)
```

For an example of how this works, click here :

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

i) Set the index of the dataframe_stdev dataset to the TIMEFRAME (DD/MM/YYYY) attribute.

ii) Using the List_Of_Variables you created in Step 8, Re-plot all the numerical variables in the dataframe_stdev for the following time periods: 10/12/2014 12:00 to 10/12/2014 14:30.

Please ensure you set the secondary axes as the Pump Failure Variable so you can observe how the variables move with respect to Pump Failure.

```
for item in ListOfVariables:
    first_axis = dataframe[___].plot #We are looping through every item in the dataframe.
    first_axis.xaxis.set_major_locator(plt.MaxNLocator(10)) #This will ensure we only plot a maximum
of 10 Time Stamps
    second_axis = first_axis.twinx() #The Twinx function is used to ensure we share the X-Axis for
both plots
    second_axis.plot(dataframe['ColumnOfInterest'], color='orange')
    second_axis.xaxis.set_major_locator(plt.MaxNLocator(10))
    plt.title(item)
```

```
plt.show()
```

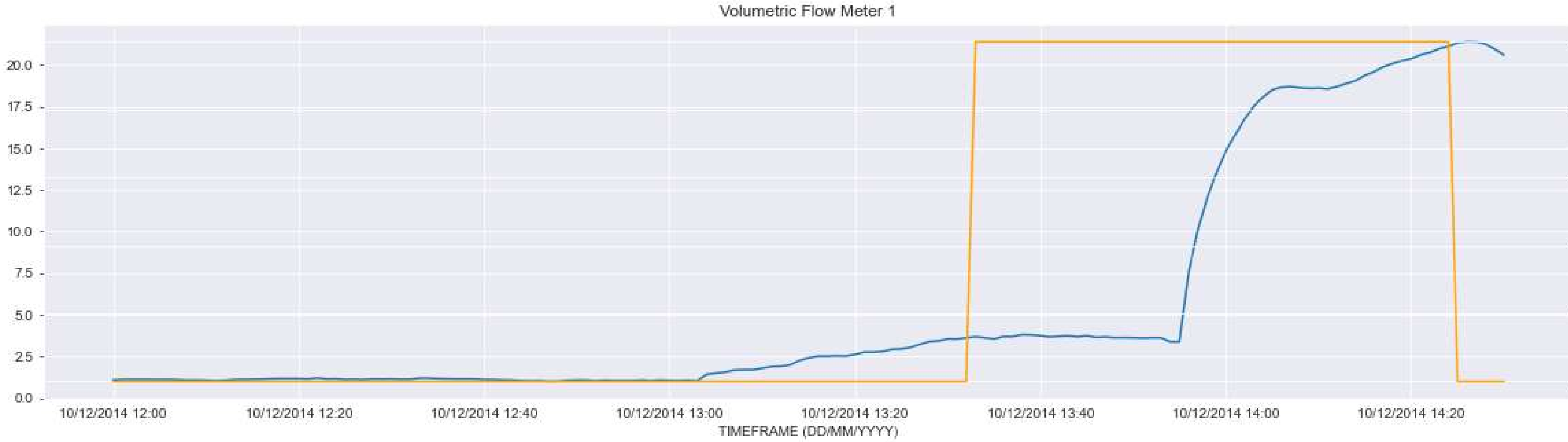
Note: To filter on the time period you will need to make use of the below syntax which *should* be familiar to you as you are subsetting your DataFrame, but instead of on a column, you're subsetting a time period.

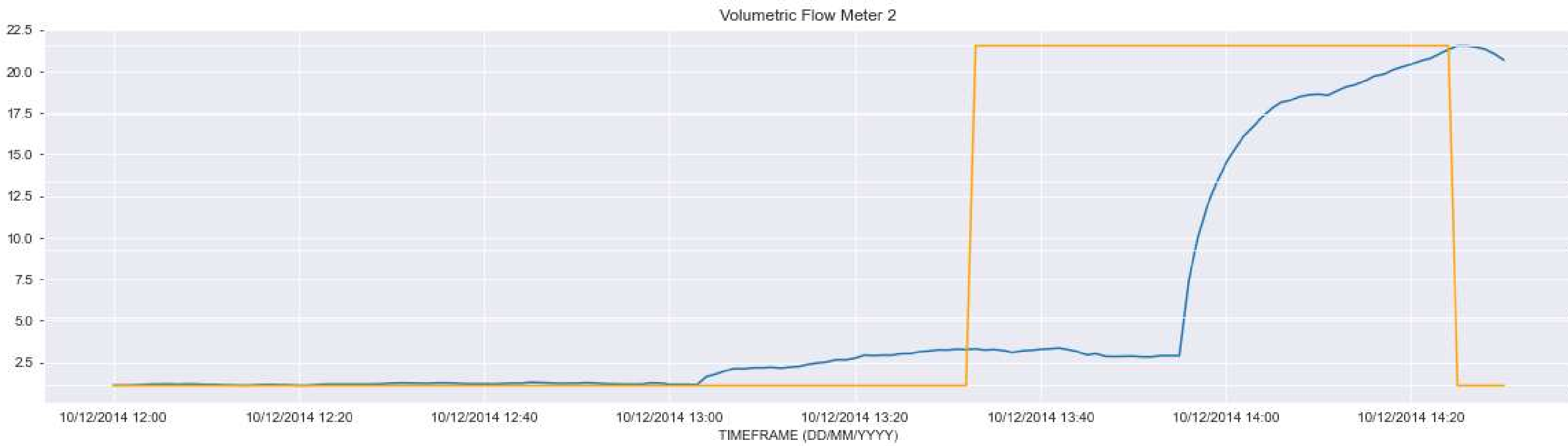
```
dataframe_time_filtered = dataframe[(dataframe.index >= "_____") & (dataframe.index <= "_____")]
```

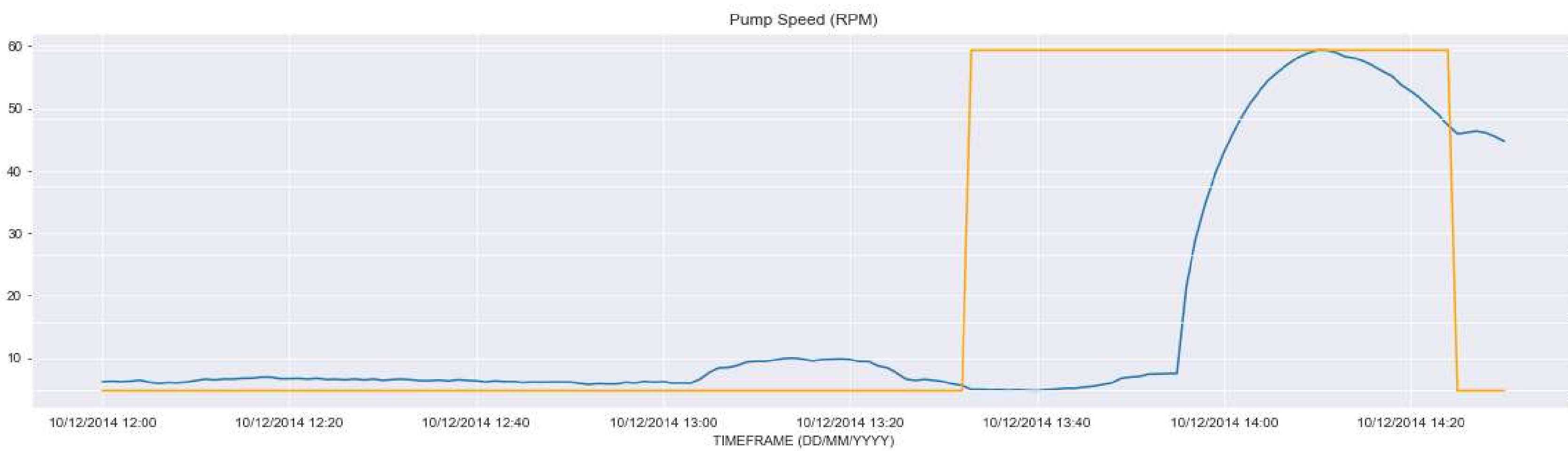
Please put your code here

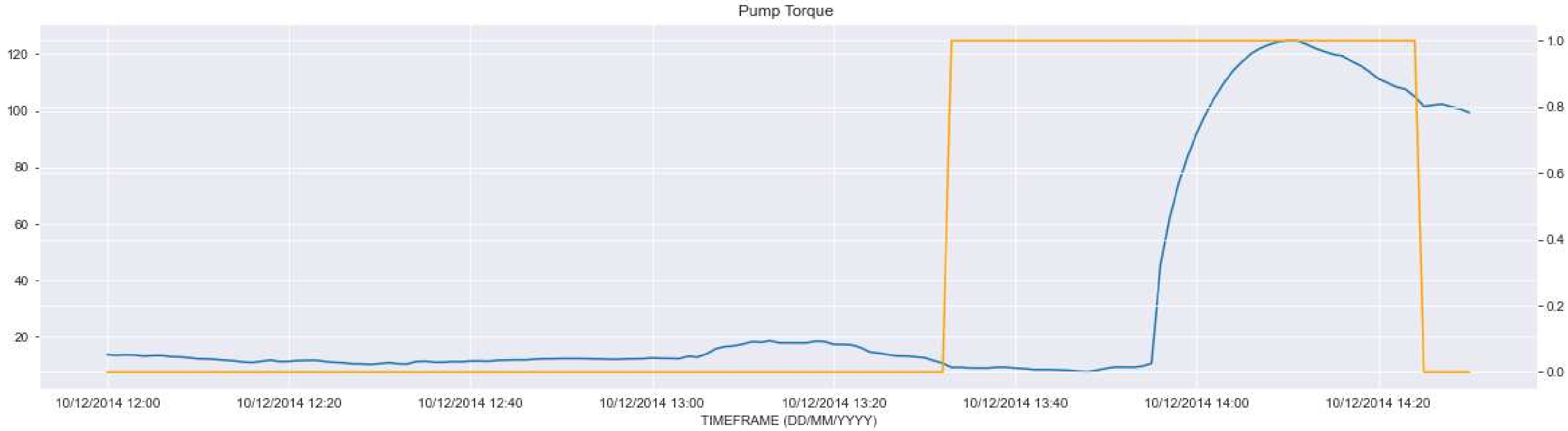
The output from your code should display image(s) like the one shown below

In [76]:
```
dataframe_stdev = pd.read_csv('DF_Rolling_Stdev.csv')
dataframe_stdev = dataframe_stdev.set_index(['TIMEFRAME (DD/MM/YYYY)'])

dataframe_time_filtered = dataframe_stdev[(dataframe_stdev.index >= '10/12/2014 12:00') & (dataframe_stdev.index <= '10
ListOfVariables = ['Volumetric Flow Meter 1', 'Volumetric Flow Meter 2', 'Pump Speed (RPM)', 'Pump Torque ', 'Ambient T

for item in ListOfVariables:
    first_axis = dataframe_time_filtered[item].plot()
    first_axis.xaxis.set_major_locator(plt.MaxNLocator(10))
    second_axis = first_axis.twinx()
    second_axis.plot(dataframe_time_filtered['PUMP FAILURE (1 or 0)'], color='orange')
    second_axis.xaxis.set_major_locator(plt.MaxNLocator(10))
    plt.title(item)
    plt.show()
```

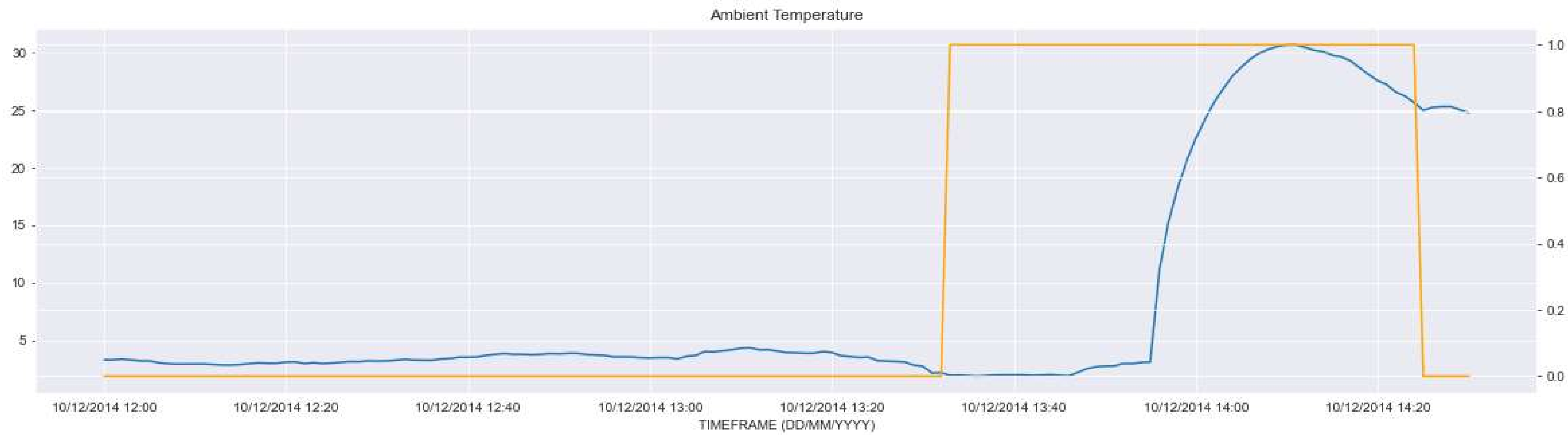

Volumetric Flow Meter 1
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0.0
0.2
0.4
0.6
0.8
1.0
10/12/2014 12:00
10/12/2014 12:20
10/12/2014 12:40
10/12/2014 13:00
10/12/2014 13:20
10/12/2014 13:40
10/12/2014 14:00
10/12/2014 14:20
TIMEFRAME (DD/MM/YYYY)


Volumetric Flow Meter 2
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
0.0
0.2
0.4
0.6
0.8
1.0
10/12/2014 12:00
10/12/2014 12:20
10/12/2014 12:40
10/12/2014 13:00
10/12/2014 13:20
10/12/2014 13:40
10/12/2014 14:00
10/12/2014 14:20
TIMEFRAME (DD/MM/YYYY)

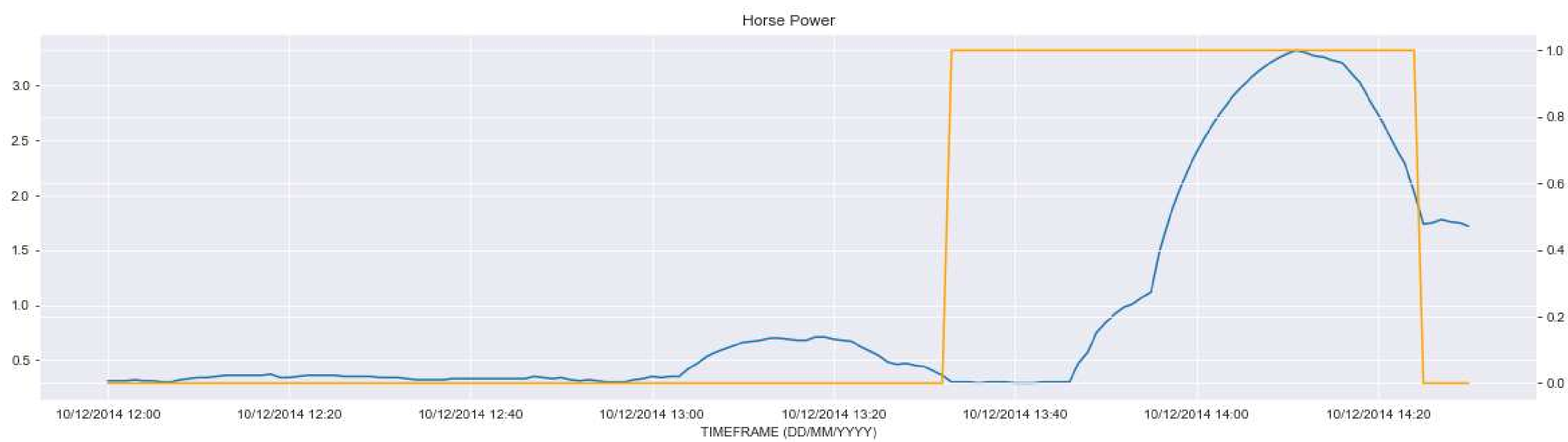

Pump Speed (RPM)
10
20
30
40
50
60
0.0
0.2
0.4
0.6
0.8
1.0
10/12/2014 12:00
10/12/2014 12:20
10/12/2014 12:40
10/12/2014 13:00
10/12/2014 13:20
10/12/2014 13:40
10/12/2014 14:00
10/12/2014 14:20
TIMEFRAME (DD/MM/YYYY)



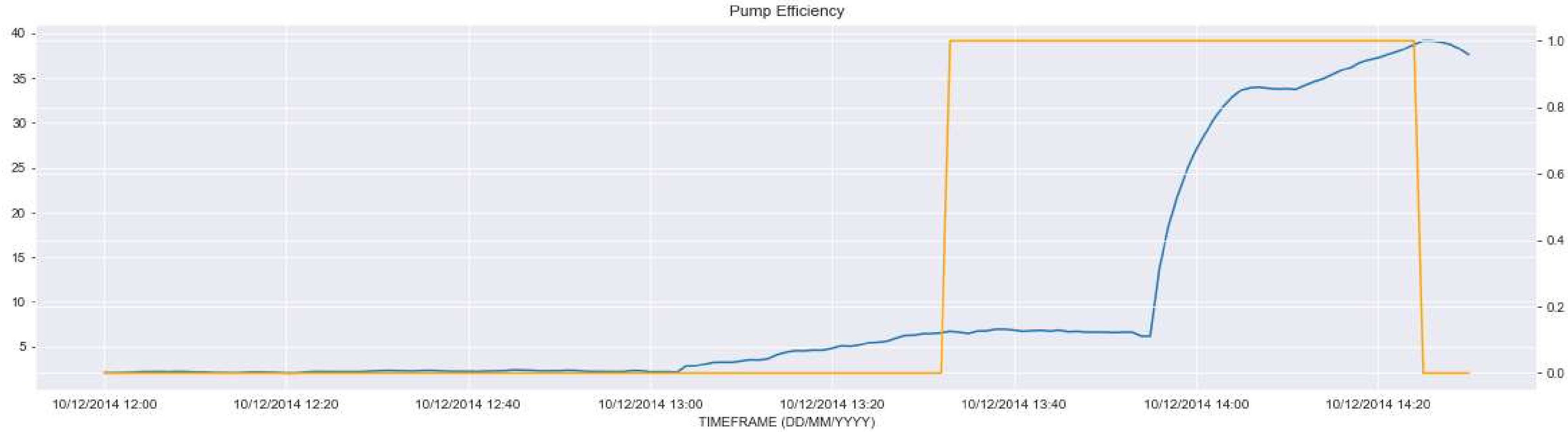
Pump Torque
120
100
80
60
40
20
1.0
0.8
0.6
0.4
0.2
0.0
10/12/2014 12:00
10/12/2014 12:20
10/12/2014 12:40
10/12/2014 13:00
10/12/2014 13:20
10/12/2014 13:40
10/12/2014 14:00
10/12/2014 14:20
TIMEFRAME (DD/MM/YYYY)

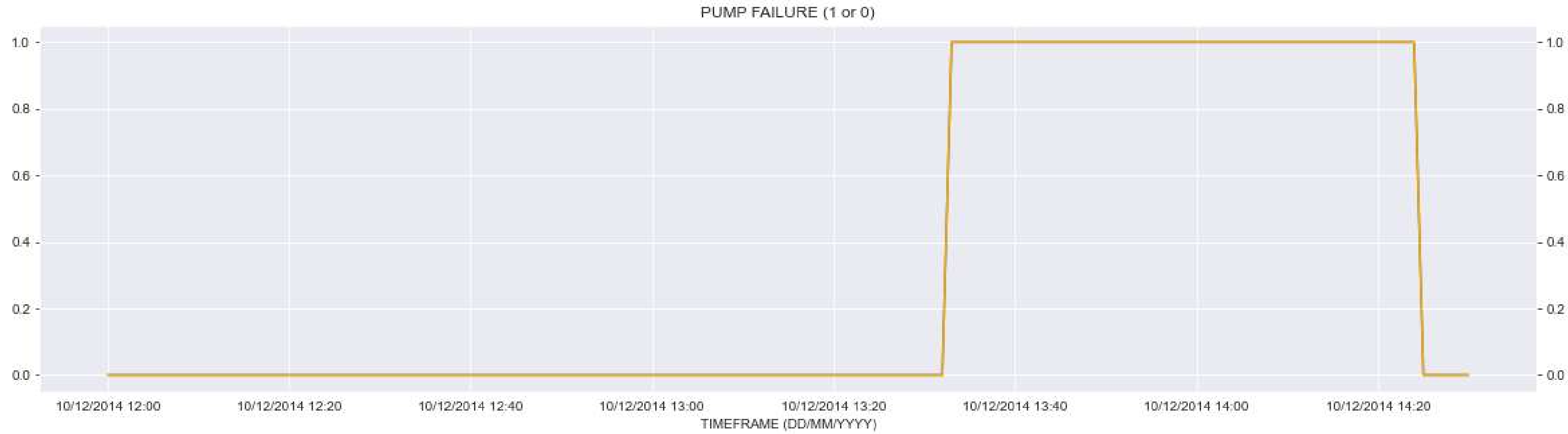


Ambient Temperature
30
25
20
15
10
5
1.0
0.8
0.6
0.4
0.2
0.0
10/12/2014 12:00
10/12/2014 12:20
10/12/2014 12:40
10/12/2014 13:00
10/12/2014 13:20
10/12/2014 13:40
10/12/2014 14:00
10/12/2014 14:20
TIMEFRAME (DD/MM/YYYY)



Horse Power
3.0
2.5
2.0
1.5
1.0
0.5
1.0
0.8
0.6
0.4
0.2
0.0
10/12/2014 12:00
10/12/2014 12:20
10/12/2014 12:40
10/12/2014 13:00
10/12/2014 13:20
10/12/2014 13:40
10/12/2014 14:00
10/12/2014 14:20
TIMEFRAME (DD/MM/YYYY)





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