outhern-water-corp-case-study-bbds

September 13, 2023

0.0.1 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.

0.0.2 Let's get started!

0.1 Part I: Descriptive Statistics

0.1.1 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.

0.2 Place your code here

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
import pandas as pd
import statsmodels.api as sm

%matplotlib inline
```

0.2.1 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.

[2]:
```
dataframe_1 = pd.read_csv('Desalination_Unit_File_001.csv',␣
 ↪parse_dates=['TIMEFRAME'])
dataframe_2 = pd.read_csv('Desalination_Unit_File_002.csv',␣
 ↪parse_dates=['TIMEFRAME'])
dataframe_3 = pd.read_csv('Desalination_Unit_File_003.csv',␣
 ↪parse_dates=['TIMEFRAME'])
combined_df = pd.concat([dataframe_1, dataframe_2, dataframe_3])

print(combined_df.describe())
print(combined_df.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            6997.000000
mean                      0.646718               0.009290
std                       0.755587               0.095941
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'>
Int64Index: 11998 entries, 0 to 2001
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)        6997 non-null   float64
 9   TIMEFRAME                    6998 non-null   datetime64[ns]
dtypes: datetime64[ns](1), float64(9)
memory usage: 1.0 MB
None
```

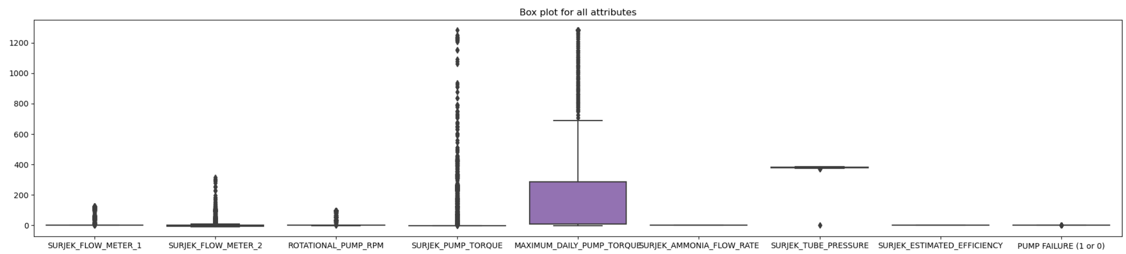
0.2.2 Step 3: Create a Boxplot

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.

0.2.3 Please put your code here

[3]:
```
fig, ax = plt.subplots(figsize=(25, 5))
sns.boxplot(data=combined_df, ax=ax)
ax.set_title('Box plot for all attributes')
plt.show()
```



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?

0.2.4 Step 4: Create a Filtered Boxplot

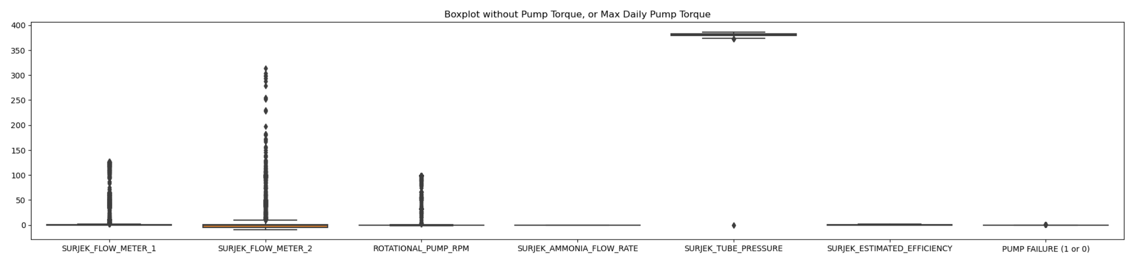
i) Create the same boxplot from Step 3, but this time, filter out SURJEK_PUMP_TORQUE and MAXIMUM_DAILY_PUMP_TORQUE. Create a new dataframe and apply a filter named **'dataframe_filt'**. Title this boxplot 'Boxplot without Pump Torque, or Max Daily Pump Torque'. We have provided the filter list for you.

Open-ended question:

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

0.3 Please put your code here

[4]:
```
#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)']
dataframe_filt = combined_df[filt]

fig, ax = plt.subplots(figsize=(25, 5))
sns.boxplot(data=dataframe_filt, ax=ax)
ax.set_title('Boxplot without Pump Torque, or Max Daily Pump Torque')
plt.show()

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



0.3.1 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.

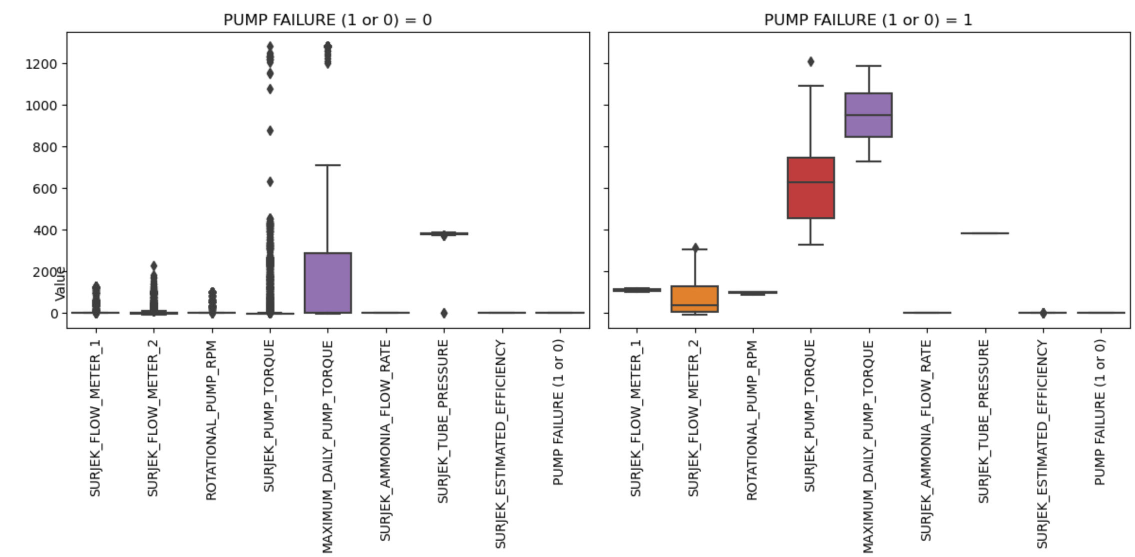
Open-ended Question:

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

0.4 Please put your code here

[5]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6), sharey=True)

data_failure_0 = combined_df[combined_df['PUMP FAILURE (1 or 0)'] == 0]
data_failure_1 = combined_df[combined_df['PUMP FAILURE (1 or 0)'] == 1]
```

```
sns.boxplot(data=data_failure_0, ax=ax1)
sns.boxplot(data=data_failure_1, ax=ax2)

ax1.set_title('PUMP FAILURE (1 or 0) = 0')
ax2.set_title('PUMP FAILURE (1 or 0) = 1')

ax1.set_xticklabels(ax1.get_xticklabels(), rotation=90)
ax2.set_xticklabels(ax2.get_xticklabels(), rotation=90)

fig.text(0.04, 0.5, 'Value', va='center', rotation='vertical')

plt.tight_layout()
plt.show()
```



0.4.1 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.

0.4.2 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.

0.5 Please put your code here

[6]:
```
Q1 = combined_df.quantile(0.25)
Q3 = combined_df.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
SURJEK_FLOW_METER_1              0.704162
SURJEK_FLOW_METER_2              5.748178
ROTATIONAL_PUMP_RPM              0.687240
SURJEK_PUMP_TORQUE               0.350032
MAXIMUM_DAILY_PUMP_TORQUE      276.315522
SURJEK_AMMONIA_FLOW_RATE         0.000000
SURJEK_TUBE_PRESSURE             3.662100
SURJEK_ESTIMATED_EFFICIENCY      1.240724
PUMP FAILURE (1 or 0)            0.000000
dtype: float64
```

0.5.1 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.

iii) 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.

0.6 Please put your code here

[7]:
```
#Below is the first part of the code
dataframe = pd.concat([combined_df])
df = dataframe.drop('TIMEFRAME', axis=1)
#---write your code below----------------
original_entries = len(df)
#We have provided the print line, you need to provide the calculation after the␣
↪quoted text:
print ("When we have not removed any outliers from the dataset, we have " +␣
↪str(original_entries) + " entries")
```

```
filtered_df = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).
 ↪any(axis=1)]

filtered_entries = len(filtered_df)
#We have provided the print line, you need to provide the calculation after the␣
 ↪quoted text:
print ("When we have removed outliers from the dataset, we have " +␣
 ↪str(filtered_entries) + " entries")

outliers_proportion = 1 - (filtered_entries / original_entries)
print ("The proportion of outliers which exist when compared to the dataframe␣
 ↪are: " + str(outliers_proportion))
```

```
When we have not removed any outliers from the dataset, we have 11998 entries
When we have removed outliers from the dataset, we have 8855 entries
The proportion of outliers which exist when compared to the dataframe are:
0.26196032672112024
```

0.6.1 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.

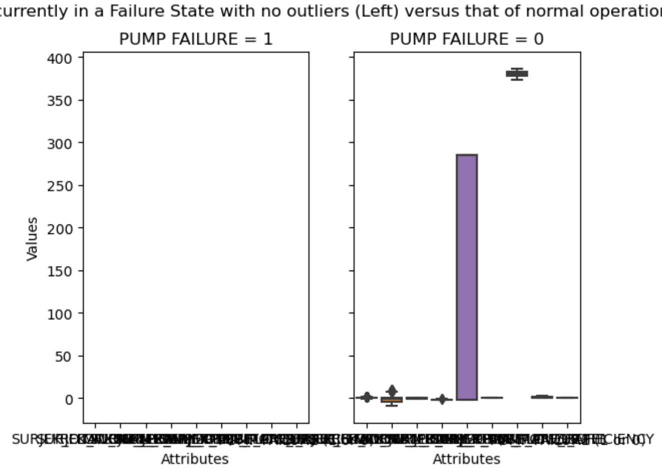
Note 2: Please display the two boxplots side-by-side. You can do this by creating a shared X axis or by creating two axes and looping through them while using the pyplot command.

0.7 Please put your code here

[8]:
```
#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)")
plt.rcParams['figure.figsize'] = (15,5)
#---write your code below-------------

# Boxplot when PUMP FAILURE is 1
sns.boxplot(data=filtered_df[filtered_df['PUMP FAILURE (1 or 0)'] == 1],␣
 ↪ax=axes[0])
```

```
axes[0].set_title("PUMP FAILURE = 1")
axes[0].set_xlabel('Attributes')
axes[0].set_ylabel('Values')

# Boxplot when PUMP FAILURE is 0
sns.boxplot(data=filtered_df[filtered_df['PUMP FAILURE (1 or 0)'] == 0], ax=axes[1])
axes[1].set_title("PUMP FAILURE = 0")
axes[1].set_xlabel('Attributes')

# Display the plots
plt.show()
```

BoxPlot when the Pump is currently in a Failure State with no outliers (Left) versus that of normal operations with no outliers (Right)



0.7.1 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.

0.7.2 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.

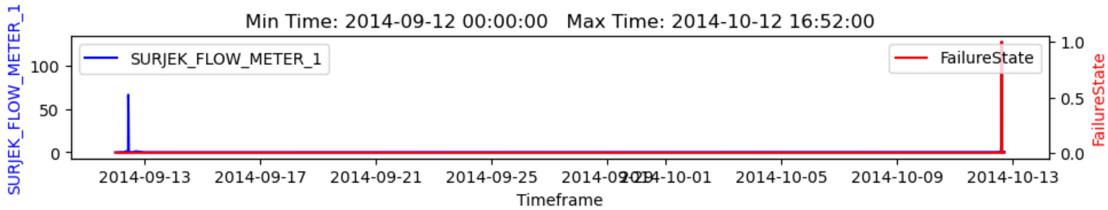
Note: For each plot, ensure that you have a dual axis set up so you can see the Pump Behaviour (0 or 1) on the second Y-axis, and the attribute (e.g. SURJEK_FLOW_METER_1) on the first Y-Axis. It might be helpful to give the failureState it's own color and add a legend to the axis to make it easier to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

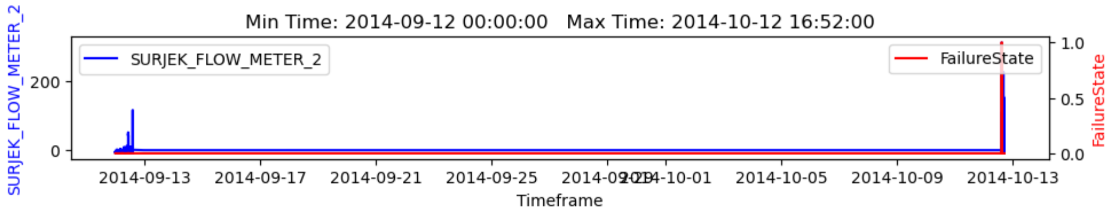
Check out this link to learn how to do this: https://matplotlib.org/gallery/api/two_scales.html

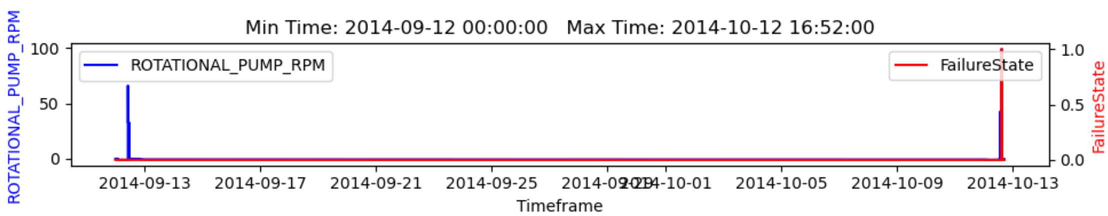
Note: Please ensure that the dataframe you are plotting contains all the outliers and that the Pump Failure Behaviour includes both the 0 and 1 State.

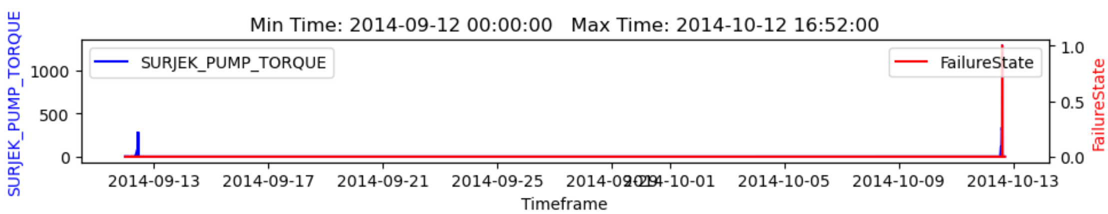
0.8 Please put your code here

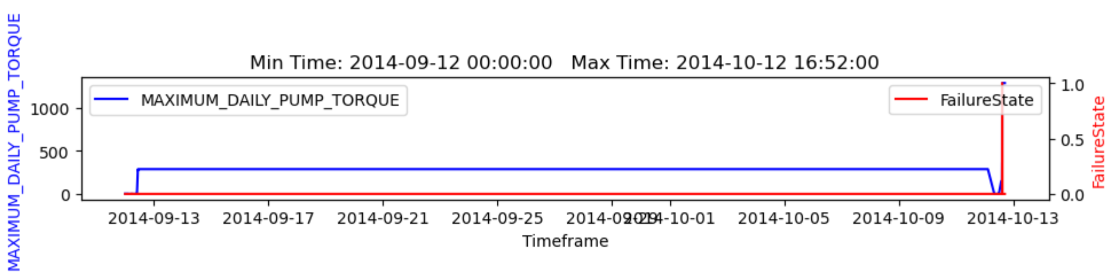
[9]:
```
#Below is the first part of the code
filt = ['SURJEK_FLOW_METER_1', 'SURJEK_FLOW_METER_2', 'ROTATIONAL_PUMP_RPM',
       'SURJEK_PUMP_TORQUE', 'MAXIMUM_DAILY_PUMP_TORQUE',
       'SURJEK_AMMONIA_FLOW_RATE', 'SURJEK_TUBE_PRESSURE',
       'SURJEK_ESTIMATED_EFFICIENCY']
filt2 = ['PUMP FAILURE (1 or 0)']
colList = combined_df[filt].columns
plt.rcParams['figure.figsize'] = (10,2)
#---write your code below-------

for column in colList:
    fig, ax1 = plt.subplots()
    ax2 = ax1.twinx()

    ax1.plot(combined_df['TIMEFRAME'], combined_df[column], color='b',␣
 ↪label=column)

    ax2.plot(combined_df['TIMEFRAME'], combined_df['PUMP FAILURE (1 or 0)'],␣
 ↪color='r', label='FailureState')

    ax1.set_xlabel('Timeframe')
    ax1.set_ylabel(column, color='b')
    ax2.set_ylabel('FailureState', color='r')
    ax1.legend(loc='upper left')
    ax2.legend(loc='upper right')

    plt.title("Min Time: " + str(combined_df['TIMEFRAME'].min()) + "   Max Time:
 ↪ " + str(combined_df['TIMEFRAME'].max()))
    plt.tight_layout()

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

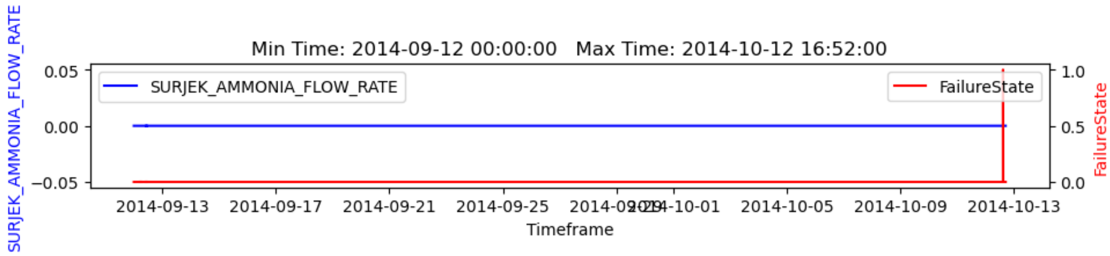


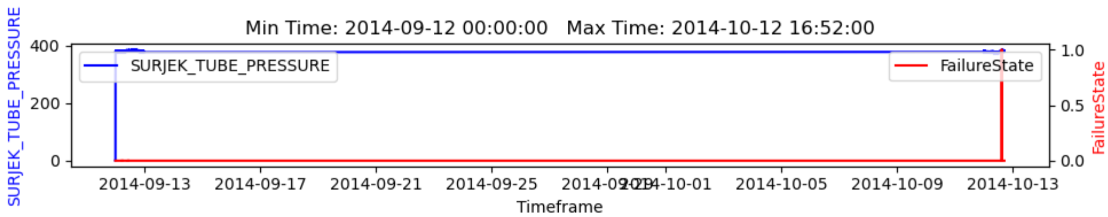

Min Time: 2014-09-12 00:00:00   Max Time: 2014-10-12 16:52:00
SURJEK_FLOW_METER_2
FailureState
Timeframe

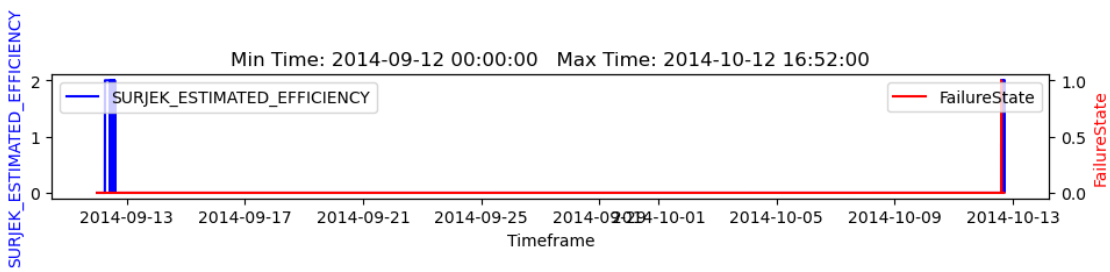

Min Time: 2014-09-12 00:00:00   Max Time: 2014-10-12 16:52:00
ROTATIONAL_PUMP_RPM
FailureState
Timeframe


Min Time: 2014-09-12 00:00:00   Max Time: 2014-10-12 16:52:00
SURJEK_PUMP_TORQUE
FailureState
Timeframe


Min Time: 2014-09-12 00:00:00   Max Time: 2014-10-12 16:52:00
MAXIMUM_DAILY_PUMP_TORQUE
FailureState
Timeframe







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!

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

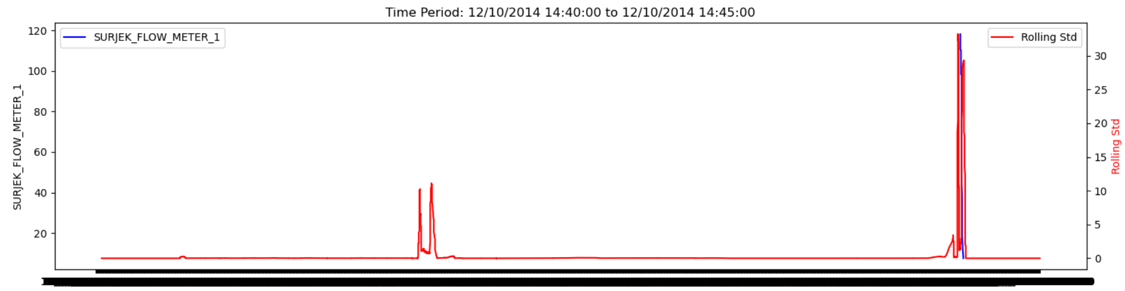
i) Apply a rolling standard deviation to the dataframe using a rolling window size of '30'.

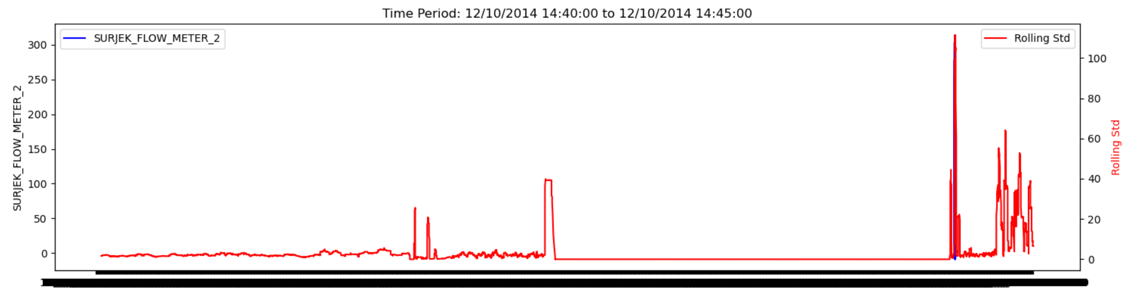
ii) Re-plot all variables for the time period 10/12/2014 14:40 to 10/12/2014 14:45, focusing specifically on the first Pump "Failure".

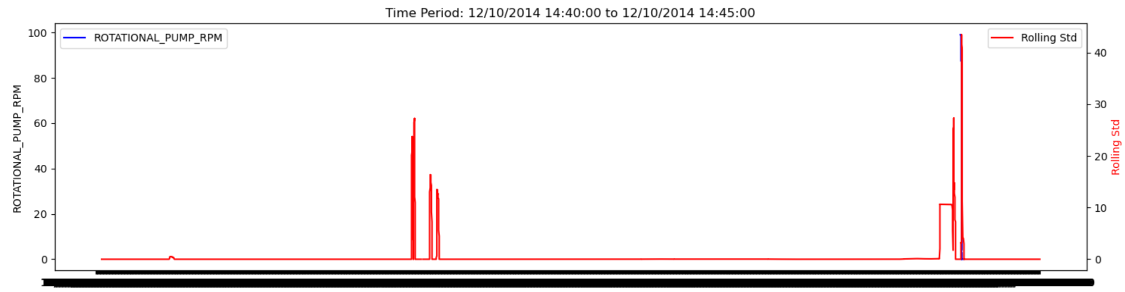
Open-ended Question: Do any particular variables seem to move in relation to the failure event?

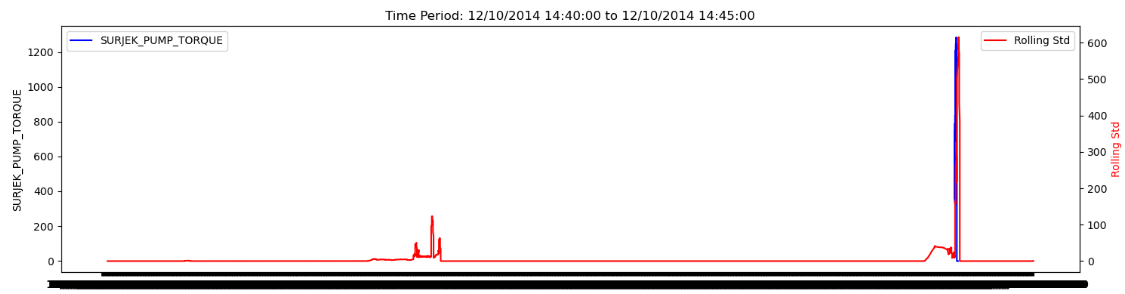
0.9 Please put your code here

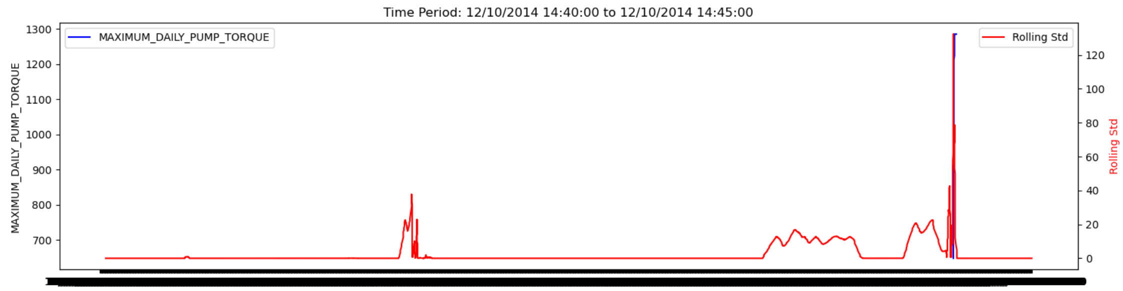
[10]:
```
#Below is the first part of the code
from datetime import datetime
dataframe = pd.concat([combined_df])
dataframe['TIMEFRAME'] = pd.to_datetime(dataframe['TIMEFRAME']).apply(lambda x:␣
↪x.strftime('%d/%m/%Y %H:%M:%S')if not pd.isnull(x) else '')
filt = ['SURJEK_FLOW_METER_1', 'SURJEK_FLOW_METER_2', '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']
colList = dataframe[filt].columns
plt.rcParams['figure.figsize'] = (15,4)
dataframe.set_index('TIMEFRAME', inplace=True)
#----write your code below-------

start_time = '12/10/2014 14:40:00'
end_time = '12/10/2014 14:45:00'

# Apply rolling standard deviation
rolling_std = dataframe[filt3].rolling(window=30).std()

# Filter data for the specified time period
filtered_data = dataframe.loc[start_time:end_time]

#Loop through the Plot
for col in filt3:
    fig, ax = plt.subplots()
    ax.plot(filtered_data.index, filtered_data[col], color='blue', label=col)
    ax.set_ylabel(col)
    ax2 = ax.twinx()
    ax2.plot(rolling_std.index, rolling_std[col], color='red', label='Rolling␣
↪Std')
    ax2.set_ylabel('Rolling Std', color='red')
    ax.legend(loc='upper left')
    ax2.legend(loc='upper right')
    plt.title(f'Time Period: {start_time} to {end_time}')
    plt.tight_layout()
    plt.show()
```

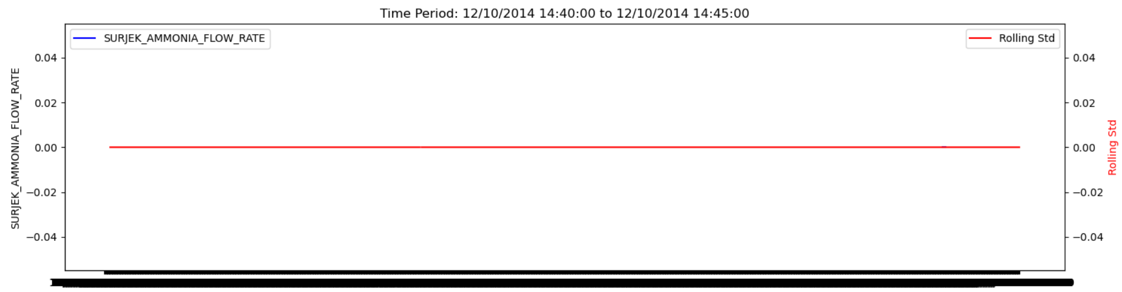

Time Period: 12/10/2014 14:40:00 to 12/10/2014 14:45:00
SURJEK_FLOW_METER_1
Rolling Std

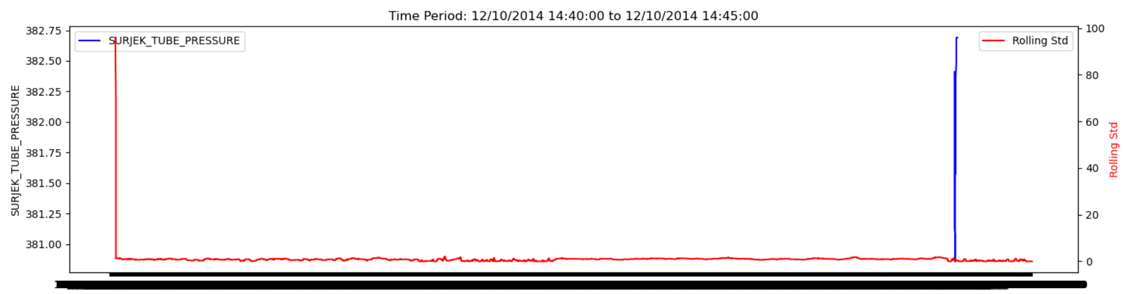

Time Period: 12/10/2014 14:40:00 to 12/10/2014 14:45:00
SURJEK_FLOW_METER_2
Rolling Std

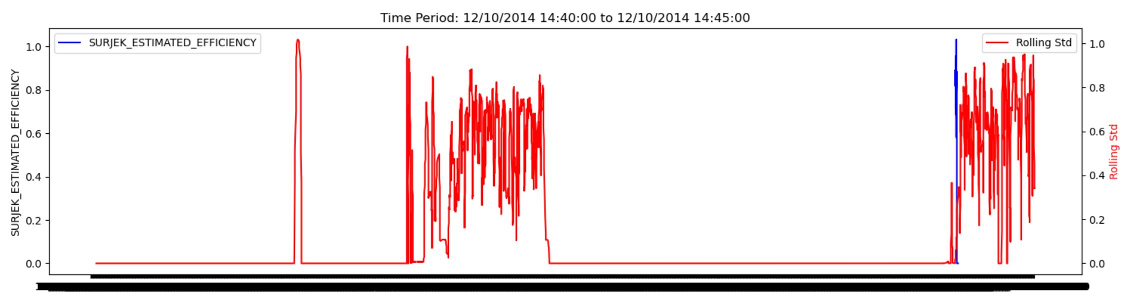

Time Period: 12/10/2014 14:40:00 to 12/10/2014 14:45:00
ROTATIONAL_PUMP_RPM
Rolling Std


Time Period: 12/10/2014 14:40:00 to 12/10/2014 14:45:00
SURJEK_PUMP_TORQUE
Rolling Std


Time Period: 12/10/2014 14:40:00 to 12/10/2014 14:45:00
MAXIMUM_DAILY_PUMP_TORQUE
Rolling Std


Time Period: 12/10/2014 14:40:00 to 12/10/2014 14:45:00
SURJEK_AMMONIA_FLOW_RATE
Rolling Std


Time Period: 12/10/2014 14:40:00 to 12/10/2014 14:45:00
SURJEK_TUBE_PRESSURE
Rolling Std


Time Period: 12/10/2014 14:40:00 to 12/10/2014 14:45:00
SURJEK_ESTIMATED_EFFICIENCY
Rolling Std

0.10 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.

0.10.1 Step 11: Create a Heatmap

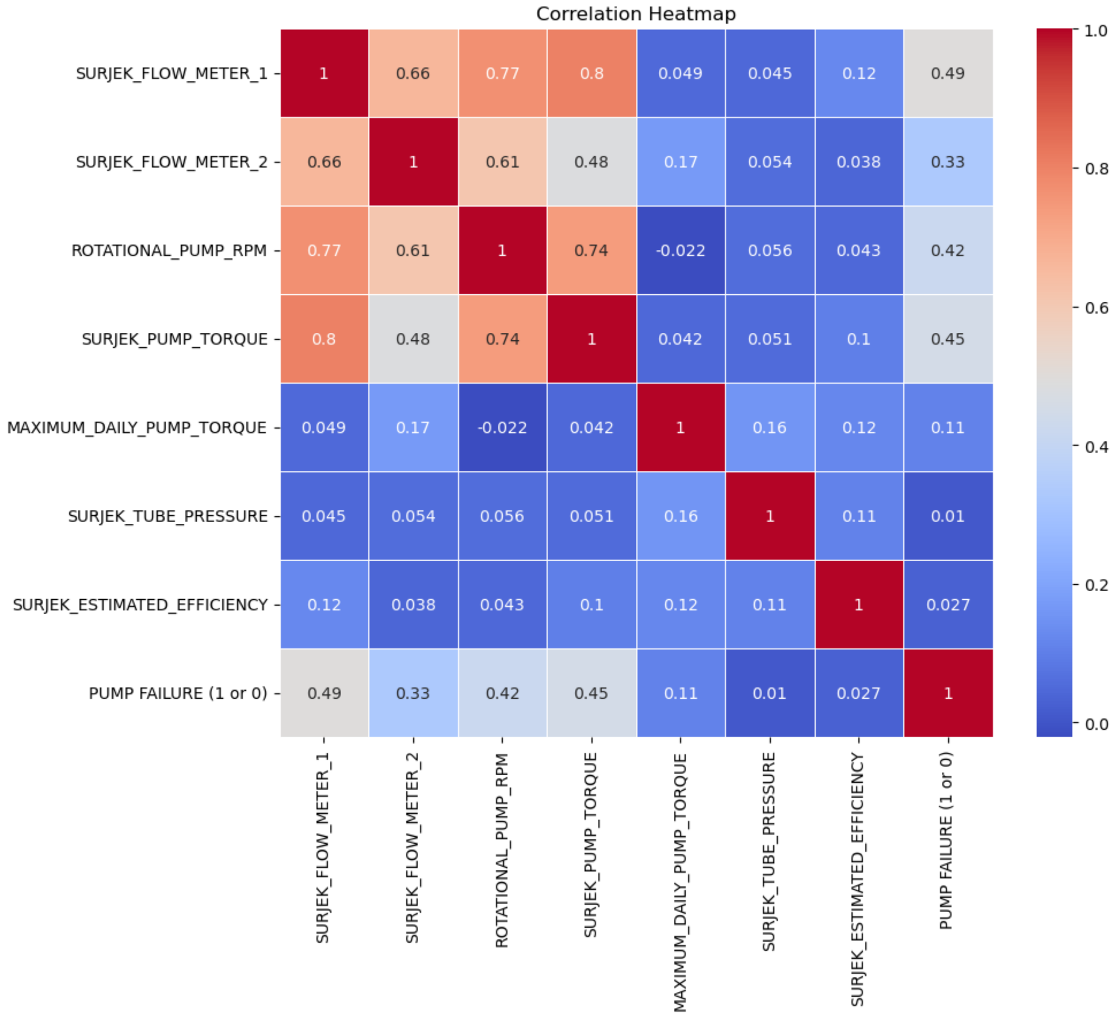
i) Using Seaborn's heatmap function, create a heatmap that clearly shows the correlations (including R Squared) for all variables (excluding those with consistent 0 values such as Ammonia Flow Rate).

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

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

0.11 Please put your code here

[13]:
```
#Below is the first part of the code
from datetime import datetime
dataframe = pd.concat([combined_df])
dataframe['TIMEFRAME'] = pd.to_datetime(dataframe['TIMEFRAME'], format="%d/%m/
↪%Y %H:%M:%S", infer_datetime_format=True )
dataframe.set_index('TIMEFRAME', inplace=True)

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

import seaborn as sns

# Calculate correlations
correlations = dataframe.corr()

# Create heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlations, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```



Open-ended Question:

Which variables seem to correlate with Pump Failure?

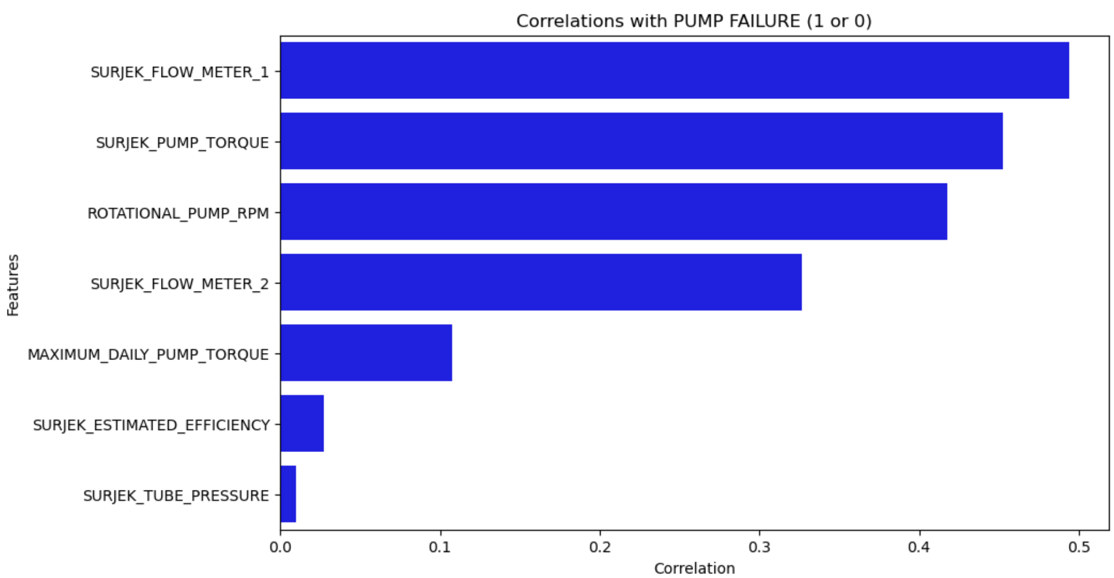
0.11.1 Step 12: Create a Barplot of Correlated Features

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

0.11.2 Please put your code here

[14]:
```
# Calculate correlations
correlations = dataframe.corr()["PUMP FAILURE (1 or 0)"]

# Select correlations with PUMP FAILURE (1 or 0)
corr_with_failure = correlations.drop("PUMP FAILURE (1 or 0)")
```

```python
# Sort correlations in descending order
corr_sorted = corr_with_failure.sort_values(ascending=False)

# Create barplot
plt.figure(figsize=(10, 6))
sns.barplot(x=corr_sorted.values, y=corr_sorted.index, color='blue')

# Set title and labels
plt.title("Correlations with PUMP FAILURE (1 or 0)")
plt.xlabel("Correlation")
plt.ylabel("Features")

# Show the plot
plt.show()
```



0.11.3 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.

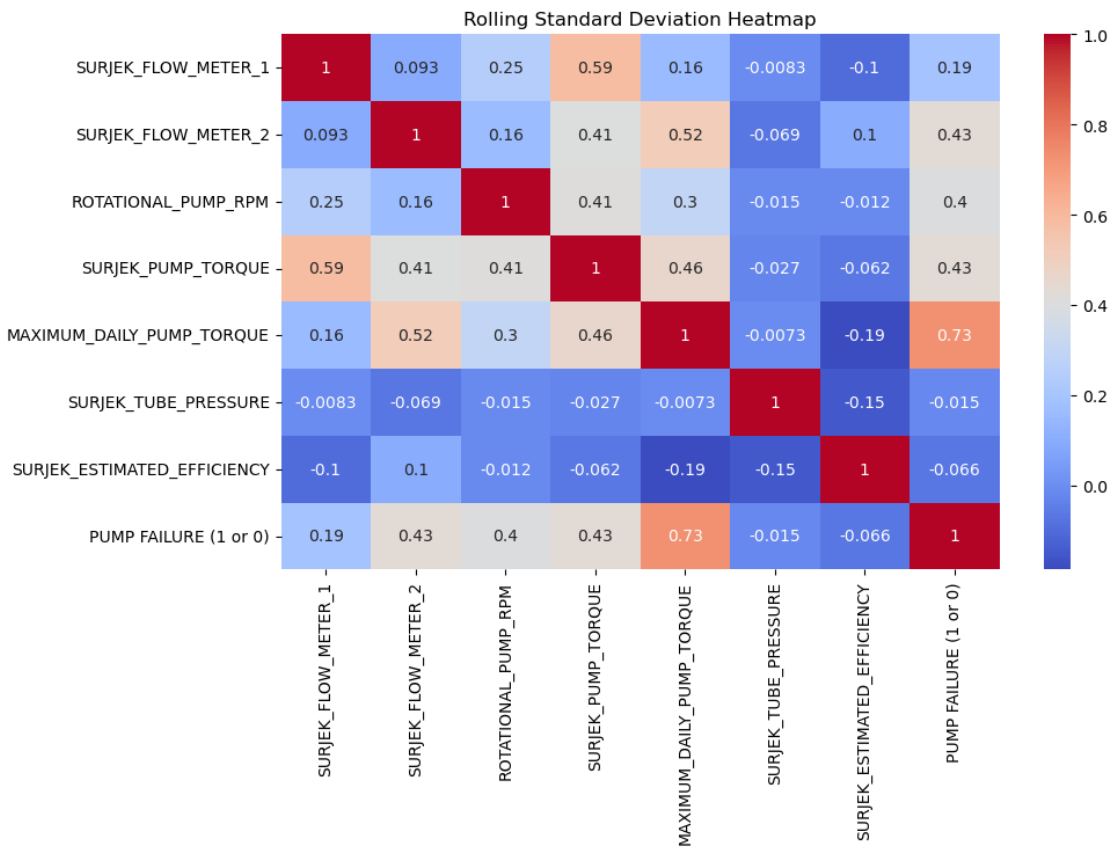
Do any variables stand out? If yes, list these out below your heatmap.

Note: We have provided the initial dataframe and filters.

0.12 Please put your code here

[15]:

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

# Apply rolling standard deviation with window size 30
rolling_std = dataframe[filt].rolling(window=30).std()

# Create heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(rolling_std.corr(), annot=True, cmap='coolwarm')

# Set title
plt.title("Rolling Standard Deviation Heatmap")

# Show the plot
plt.show()
```



0.12.1 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).

0.12.2 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.

0.13 Please put your code here

[17]:
```
combined_df = combined_df.dropna()
```

[26]:
```
#Answer for step i):
#Below is the first part of the code
dataframe_two = pd.concat([combined_df])
dependentVar = dataframe_two['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_TUBE_PRESSURE',
       'SURJEK_ESTIMATED_EFFICIENCY', 'PUMP FAILURE (1 or 0)']
#----write your code below------

# Select independent variables
independentVars = dataframe_two[filt[:-1]]  # Exclude the last variable (PUMP␣
 ↪FAILURE)

# Add constant term
independentVars = sm.add_constant(independentVars)

# Create OLS model
model = sm.OLS(dependentVar, independentVars)

# Fit the model
results = model.fit()

# Print R-squared value
print("R-squared:", results.rsquared)
```

```
R-squared: 0.2643556987574943
```

[28]:
```
#Answer for step ii):
#Below is the first part of the code

dataframe_two = pd.concat([combined_df])
dependentVar = dataframe_two['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_TUBE_PRESSURE',
       'SURJEK_ESTIMATED_EFFICIENCY', 'PUMP FAILURE (1 or 0)']
#----write your code below------

rolling_std = dataframe_two[filt].rolling(window=30).std()
```

```
# Select independent variables with rolling standard deviation
independentVars_rolling = rolling_std[filt[:-1]]

# Add constant column
independentVars_rolling = sm.add_constant(independentVars_rolling)

# Fit the OLS model with rolling standard deviation
model_rolling = sm.OLS(dependentVar, independentVars_rolling)
results_rolling = model_rolling.fit()

# Print R-squared
print("R-squared (with rolling std):", results_rolling.rsquared)
```

```
---------------------------------------------------------------------------
MissingDataError                          Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_68\2616668047.py in <module>
     18 
     19 # Fit the OLS model with rolling standard deviation
---> 20 model_rolling = sm.OLS(dependentVar, independentVars_rolling)
     21 results_rolling = model_rolling.fit()
     22 

~\Bureau\anaconda3\lib\site-packages\statsmodels\regression\linear_model.py in␣
 ↪__init__(self, endog, exog, missing, hasconst, **kwargs)
    888                    "An exception will be raised in the next version.")
    889             warnings.warn(msg, ValueWarning)
--> 890         super(OLS, self).__init__(endog, exog, missing=missing,

    891                                   hasconst=hasconst, **kwargs)
    892         if "weights" in self._init_keys:

~\Bureau\anaconda3\lib\site-packages\statsmodels\regression\linear_model.py in␣
 ↪__init__(self, endog, exog, weights, missing, hasconst, **kwargs)
    715         else:
    716             weights = weights.squeeze()
--> 717         super(WLS, self).__init__(endog, exog, missing=missing,

    718                                   weights=weights, hasconst=hasconst,␣
 ↪**kwargs)
    719         nobs = self.exog.shape[0]

~\Bureau\anaconda3\lib\site-packages\statsmodels\regression\linear_model.py in␣
 ↪__init__(self, endog, exog, **kwargs)
    189     """
    190     def __init__(self, endog, exog, **kwargs):
--> 191         super(RegressionModel, self).__init__(endog, exog, **kwargs)
    192         self._data_attr.extend(['pinv_wexog', 'wendog', 'wexog',␣
 ↪'weights'])
```

```
    193

~\Bureau\anaconda3\lib\site-packages\statsmodels\base\model.py in __init__(self, endog, exog, **kwargs)
    265
    266     def __init__(self, endog, exog=None, **kwargs):
--> 267         super().__init__(endog, exog, **kwargs)
    268         self.initialize()
    269

~\Bureau\anaconda3\lib\site-packages\statsmodels\base\model.py in __init__(self, endog, exog, **kwargs)
     90         missing = kwargs.pop('missing', 'none')
     91         hasconst = kwargs.pop('hasconst', None)
---> 92         self.data = self._handle_data(endog, exog, missing, hasconst,
     93                                       **kwargs)
     94         self.k_constant = self.data.k_constant

~\Bureau\anaconda3\lib\site-packages\statsmodels\base\model.py in _handle_data(self, endog, exog, missing, hasconst, **kwargs)
    130
    131     def _handle_data(self, endog, exog, missing, hasconst, **kwargs):
--> 132         data = handle_data(endog, exog, missing, hasconst, **kwargs)
    133         # kwargs arrays could have changed, easier to just attach here
    134         for key in kwargs:

~\Bureau\anaconda3\lib\site-packages\statsmodels\base\data.py in handle_data(endog, exog, missing, hasconst, **kwargs)
    671
    672     klass = handle_data_class_factory(endog, exog)
--> 673     return klass(endog, exog=exog, missing=missing, hasconst=hasconst,
    674                  **kwargs)

~\Bureau\anaconda3\lib\site-packages\statsmodels\base\data.py in __init__(self, endog, exog, missing, hasconst, **kwargs)
     84         self.const_idx = None
     85         self.k_constant = 0
---> 86         self._handle_constant(hasconst)
     87         self._check_integrity()
     88         self._cache = {}

~\Bureau\anaconda3\lib\site-packages\statsmodels\base\data.py in _handle_constant(self, hasconst)
    130             exog_max = np.max(self.exog, axis=0)
    131             if not np.isfinite(exog_max).all():
--> 132                 raise MissingDataError('exog contains inf or nans')
    133             exog_min = np.min(self.exog, axis=0)
```

```
    134                 const_idx = np.where(exog_max == exog_min)[0].squeeze()

MissingDataError: exog contains inf or nans
```

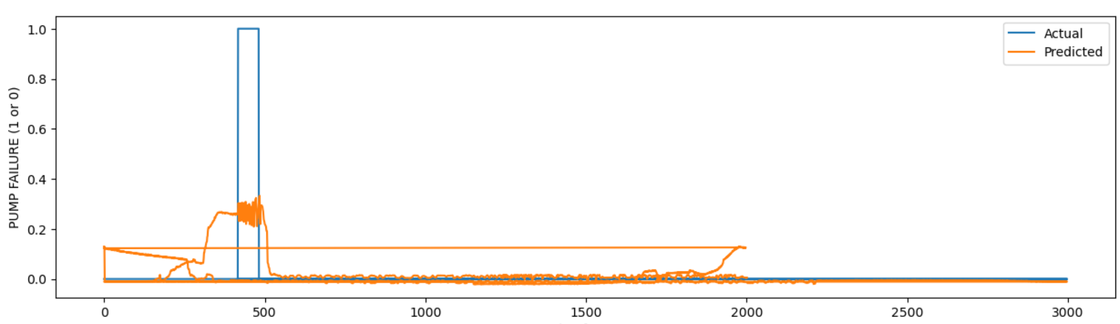
Great job creating those regressive equations! You've reached the final step of this case study! ### Step 15: Validate Predictions i) Use the regression equation you created in the previous step and apply the .predict() function to the dataframe to see whether or not your model 'picks' up the Pump Failure 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.

0.14 Please put your code here

[29]:
```
#Below is the first part of the code
plt.rcParams['figure.figsize'] = (15,4)
#----write your code below------
#ax.legend(bbox_to_anchor=(1.04,1), loc="upper left")


# Use the regression equation to make predictions
predictions = results.predict(independentVars)

# Plot the predictions and actual values
plt.plot(dataframe_two.index, dependentVar, label='Actual')
plt.plot(dataframe_two.index, predictions, label='Predicted')

# Label the axes and add a legend
plt.xlabel('Timeframe')
plt.ylabel('PUMP FAILURE (1 or 0)')
plt.legend()

# Show the plot
plt.show()
```



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

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