Kalman Filter + Benching

Python Scripts

main_kalman_filter.py Read in the SLM CSV

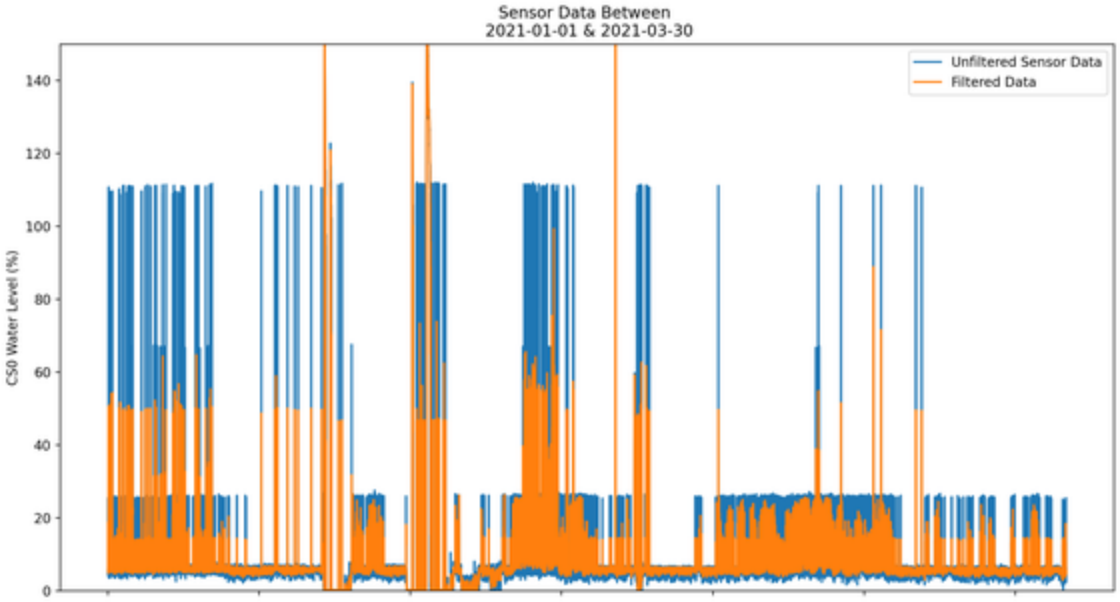
kalman.py Contains the mathematics to filter original data

Kalman Filter Parameters

```
# ---------------------------------------- Filtering: PARAMETERS :
(START) --------------------------------------
"""
    The Parameters for the Filter are outlined below.

    Q: The Process Covariance weights the final Kalman Output either
towards the actual measurement or Kalman
       Prediction. The lower this value is, for example, 0.005, the
less noise/volatility in the output.
       You must use judgement when setting this variable.

    A: Process Innovation: I tend to leave as 1 (has no affect). If >1
then the graph will drift upwards by the scaling
       factor.

    x: We make an arbitrary guess at the beginning. After a certain
amount of data, the model increases to the correct
       value. This is similar to the Posterior in Bayesian Updating.

    P: Again this is arbitrary and will update to the correct values
after multiple data is passed into the Filter.

"""
# Initialise the Kalman Filter
A = 1       # No process innovation - use 1
C = 1       # Measurement
B = 0       # No control input
Q = 0.3     # Process covariance
R = 1       # Measurement covariance
x = 100     # Initial state_estimate
P = 1       # Initial covariance/ prob_estimate
```

Q: The Process Covariance is the main parameter for tuning. Personally I would leave this at 0.3.

When Q is 0.3



When Q is 0.0005

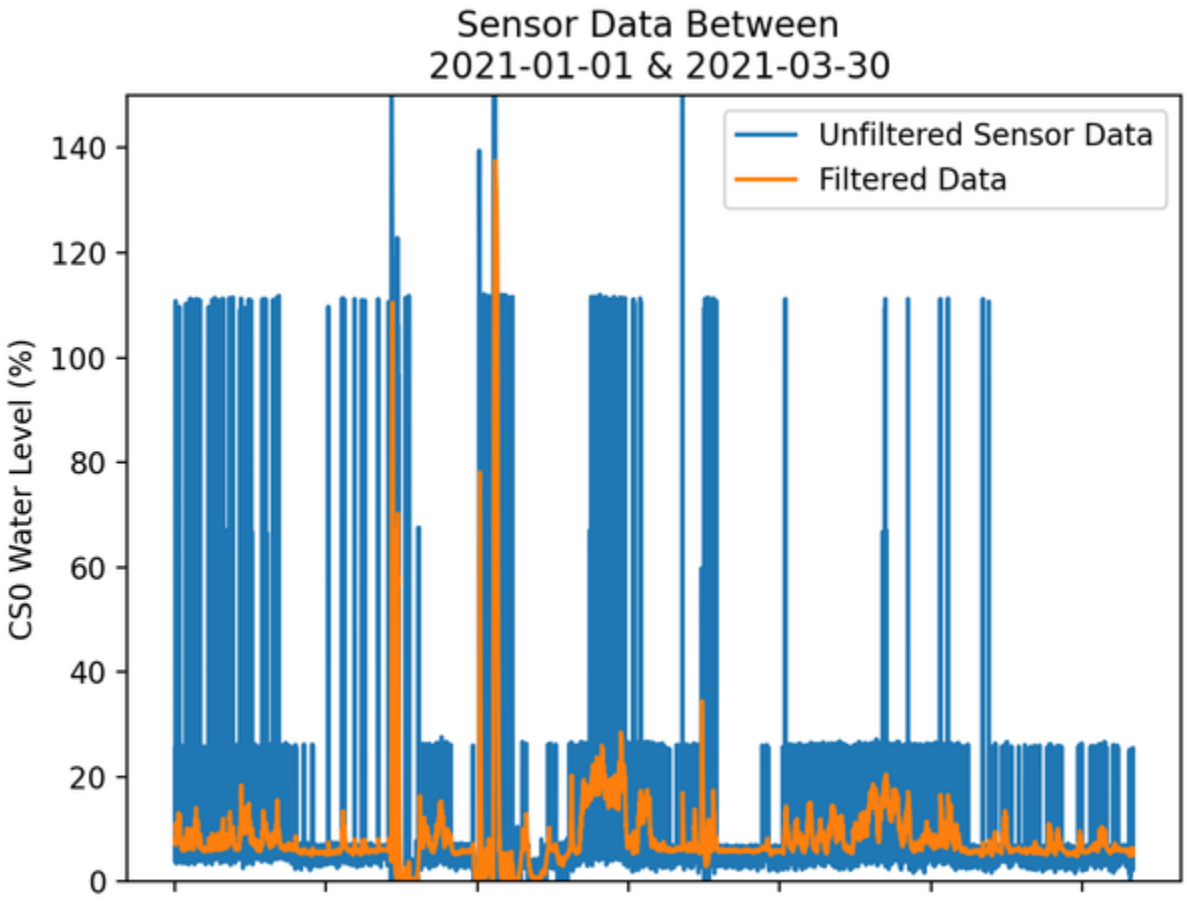
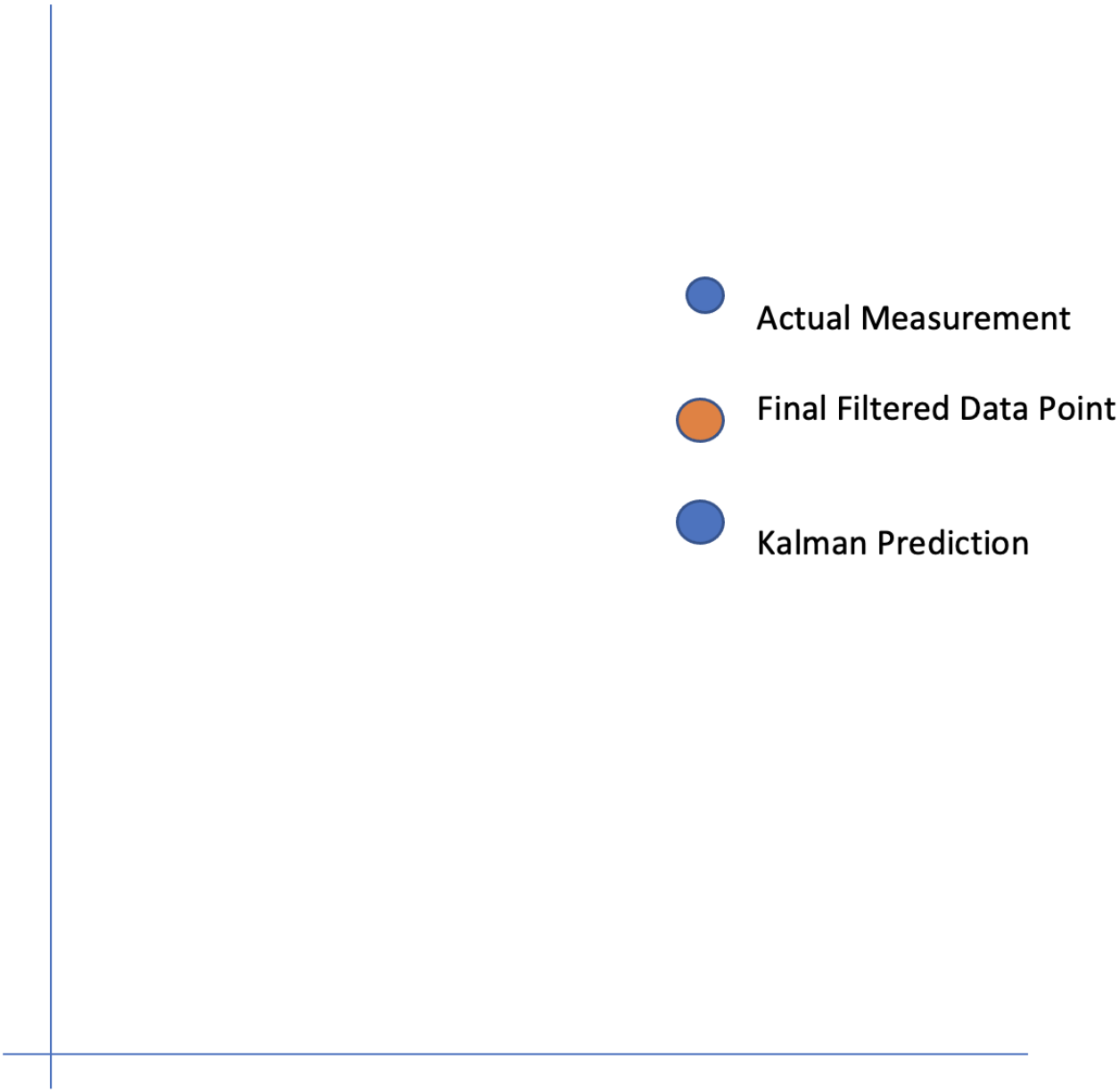


Illustration of the Filter logic.



Lower Q Final Filtered Data Point moves closer to the Kalman Prediction

Higher Q Final Filtered Data Point moves closer to Actual Measurement.

Initialise Kalman Instance

```
# Initialise The Kalman Object With Parameters
kalman_filter = SingleStateKalmanFilter(A, B, C, x, P, Q, R)
```

Ingest SLM Data File (real time will presumably be set to 'False')

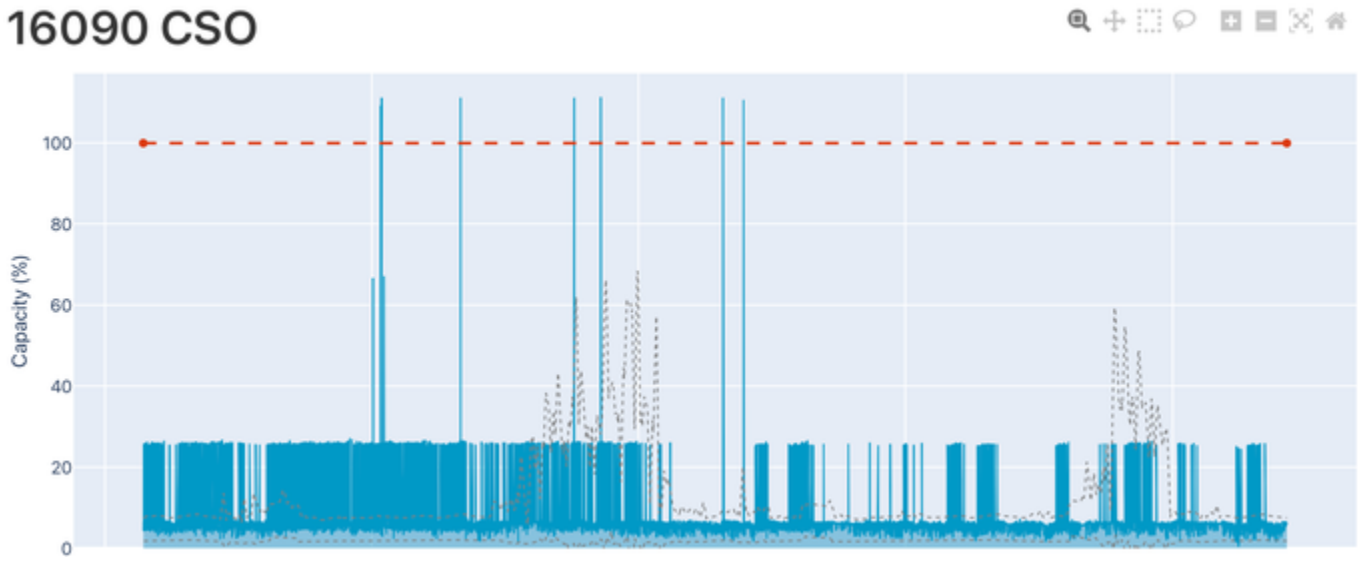
```
if real_time:
    """ If Real_Time is True then read in real time data from the
Crontab. Else Just Run all the
        data through the Filter.
    """
    csv = 'kalman_real_time.csv'
else:
    csv = '16090_CSO_SLM_data.csv'
```

Therefore the Filter will be applied to the entire SLM data set, as opposed to the new data coming in from the Crontab.

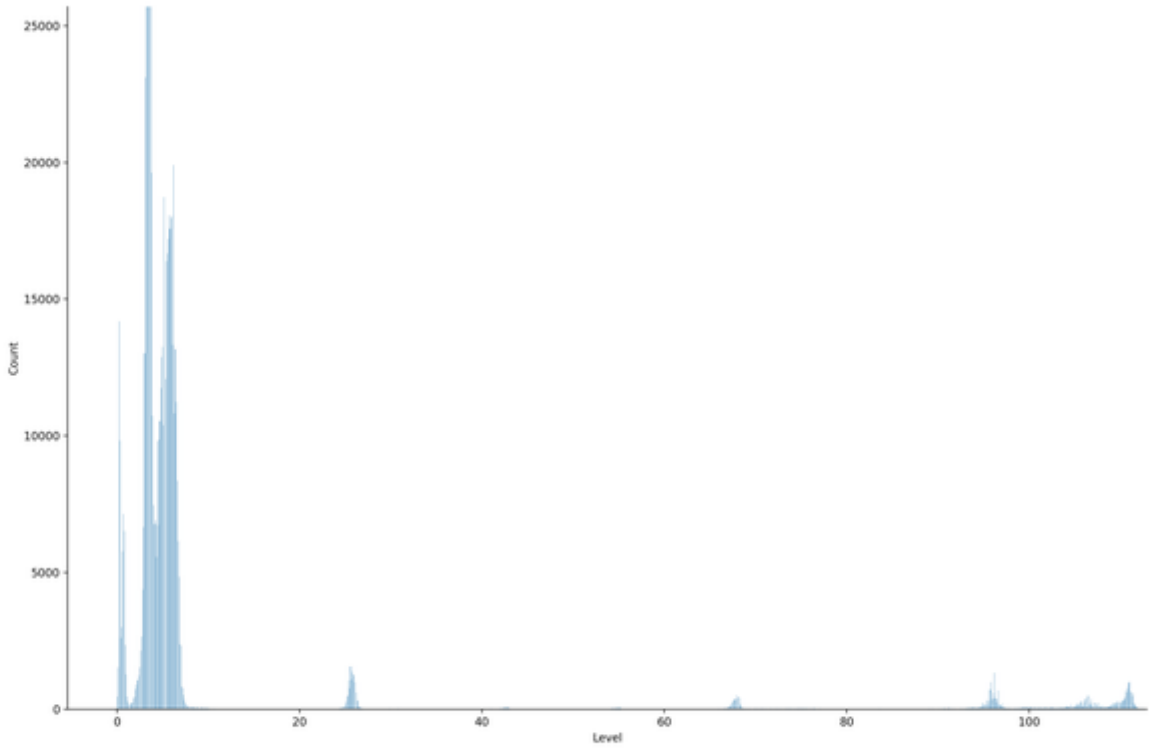
Check for Benching

Set:

```
show_distribution = True
```



We can see a lot of Benching at 25%, therefore we should expect to see a Gaussian Distribution around that point, which is exactly what we see in the distribution graph below (there are also further benching locations):



To strip out Benching, set Benching_Tool to True.

```
Benching_Tool = True
# Decide Count or Instances of a certain value appearing for which we
wich to erase.
#This is a Threshold Value. The Higher the Count, the more likely
benching is at this location.
count_to_erase = 50
if Benching_Tool:
    df_benching_loc = df['Level'].value_counts()
    df_benching_loc = df_benching_loc.to_frame()
    df_benching_loc.reset_index(level=0, inplace=True)
    df_benching_loc.rename(columns={'index': 'Levels'}, inplace=True)
    df_benching_loc.rename(columns={'Level': 'Count'}, inplace=True)
    # Choose Benching Corridors : Hint: Run show_distribution plot to
identify benching regions

    df_benching_loc = df_benching_loc[((df_benching_loc['Levels'] >=
24.) & (df_benching_loc['Levels'] <= 26.5)) \
                                      | ((df_benching_loc['Levels'] >=
95) & (df_benching_loc['Levels'] <= 97))  \
                                      | ((df_benching_loc['Levels'] >=
95) & (df_benching_loc['Levels'] <= 97))  \
                                      | ((df_benching_loc['Levels'] >=
108) & (df_benching_loc['Levels'] <= 112))\
                                      | ((df_benching_loc['Levels'] >=
66) & (df_benching_loc['Levels'] <= 68))]
```

Manually Insert Benching Locations:

```
df_benching_loc = df_benching_loc[((df_benching_loc['Levels'] >= 24.) &
(df_benching_loc['Levels'] <= 26.5)) \
                                      | ((df_benching_loc['Levels'] >=
95) & (df_benching_loc['Levels'] <= 97))  \
                                      | ((df_benching_loc['Levels'] >=
95) & (df_benching_loc['Levels'] <= 97))  \
                                      | ((df_benching_loc['Levels'] >=
108) & (df_benching_loc['Levels'] <= 112))\
                                      | ((df_benching_loc['Levels'] >=
66) & (df_benching_loc['Levels'] <= 68))]
```

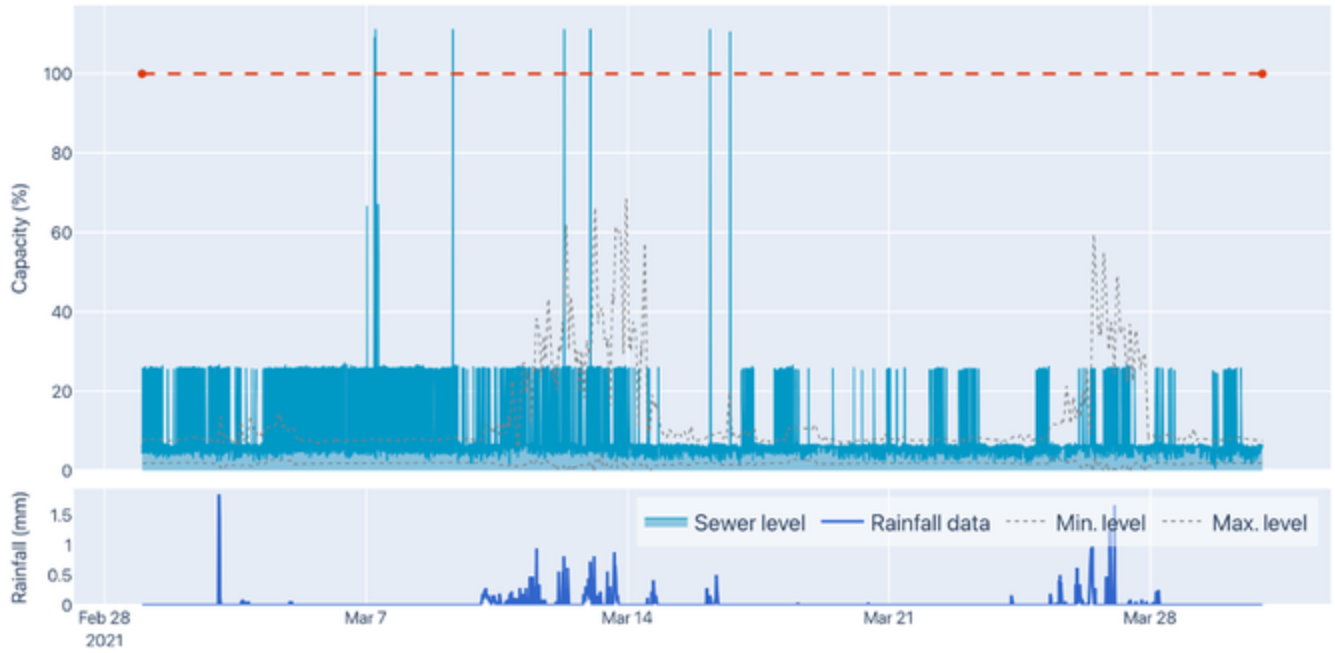
count_to_erase is a threshold value for which we strip out values. If count_to_erase is 50, and 25.2 appears 52 times, then all values with 25.2 will be pulled out of the SLM Data File. If the count for 25.2 is 40, then it will remain untouched.

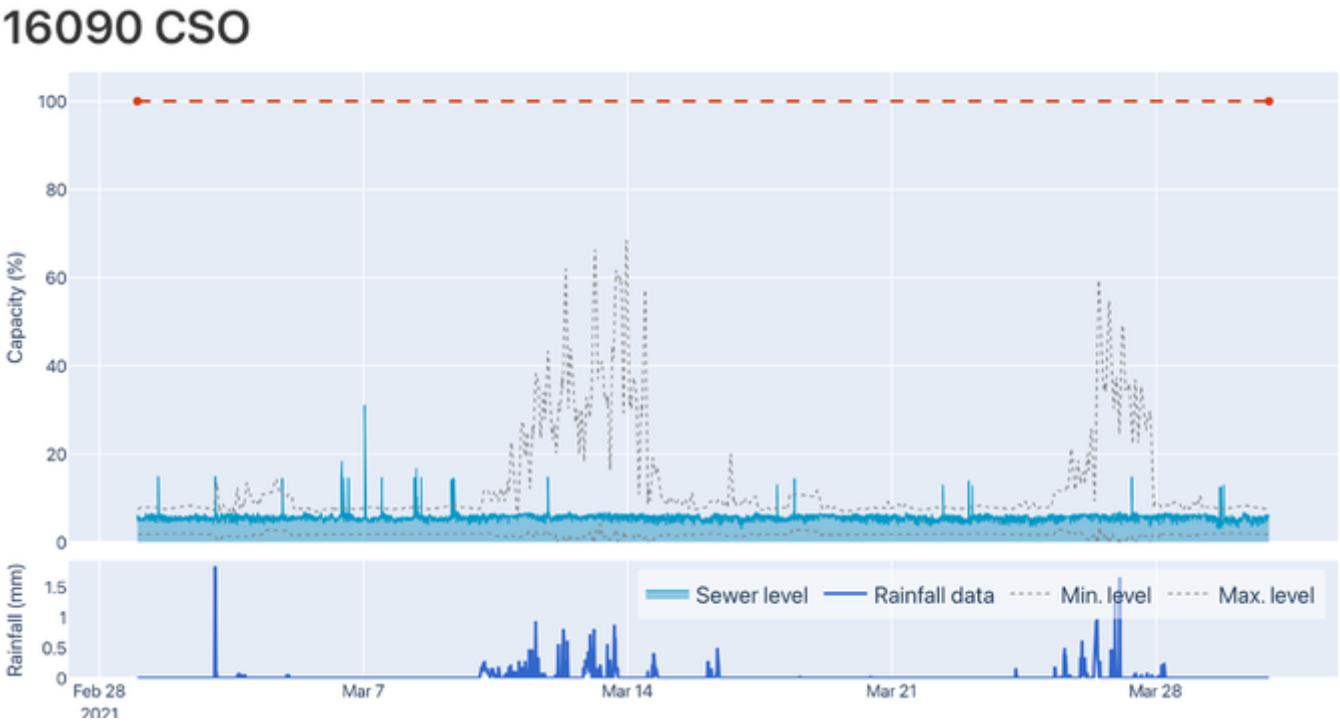
We are looking to strip out values that are at the peak of the Gaussian Distribution in the suspected Benching location (24 26.5), which is a proxy for values which have a higher probability of being sensor faults/benching.

Our data now represents a Swiss cheese with thousands of suspected benching values deleted.

The Kalman Filter when faced with absent data is capable of imputing (think Kalman Prediction in the graph above). I have kept this simple, however, it can be calibrated to have drift etc. Therefore we can plaster over the holes in our Swiss cheese using the Filter in lighting speed.

Results:





Filtering

The main file will call kalman.py which simply contains the mathematics below.

Initialize $\hat{\mathbf{x}}_{0|0}$ and $\mathbf{P}_{0|0}$.

At each iteration $k = 1, \ldots, n$

Predict

Predicted (a priori) state estimate

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k \mathbf{u}_k$$

Predicted (a priori) estimate covariance

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^{\mathrm{T}} + \mathbf{Q}_k$$

Update

Innovation or measurement residual

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1}$$

Innovation (or residual) covariance

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^{\mathrm{T}} + \mathbf{R}_k$$

Optimal ***Kalman gain***

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^{\mathrm{T}} \mathbf{S}_k^{-1}$$

Updated (a posteriori) state estimate

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k$$

Updated (a posteriori) estimate covariance

$$\mathbf{P}_{k|k} = (I - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1}$$

DO NOT TOUCH THIS FILE, ELSE:

Seaborn To View Data

Set:

`Sea_Plot = True`

Create Filtered Data CSV

Set create_csv to True. Also ensure the file_name is correct, in this case example the filtered data will be named `16090_CSO_SLM_data_KF.csv`

```
# ---------------------------------------- Create Filtered Data CSV
(START)----------------------------------------
create_csv = True
if create_csv:
    with open('16090_CSO_SLM_data_KF.csv', 'w') as f:
        for i, key in enumerate(kalman_filtered_data_dict.keys()):
            if i == 0:
                f.write("%s,%s,%s,%s\n" % (key,
kalman_filtered_data_dict[key][0], kalman_filtered_data_dict[key][1],
kalman_filtered_data_dict[key][2]))
            else:
                f.write("%s,%.2f,%.2f,%i\n"%(key,
kalman_filtered_data_dict[key][0],kalman_filtered_data_dict[key][1], 1
if kalman_filtered_data_dict[key][0] > 100 else 0))
# ---------------------------------------- Create Filtered Data CSV
(END)----------------------------------------
```