Python Final Project
Sensor Disagree Plotting

Justin Moss

Objective:

- Import aircraft Horizontal Stabilizer Position Sensor
- Manipulate data for analysis
- Analyze sensor data and prepare for predictive analysis
- Determine when a sensor will fail with machine learning model

Quick Notes:

A sensor is deemed to have failed when it has a .3° or greater difference between either of the other two sensors.

Not all objectives could be accomplished due to the scope of the data imported.

Instead, the objectives were modified over the course of the project to better suit the available data.

Step 1:

Import libraries

Read in the data

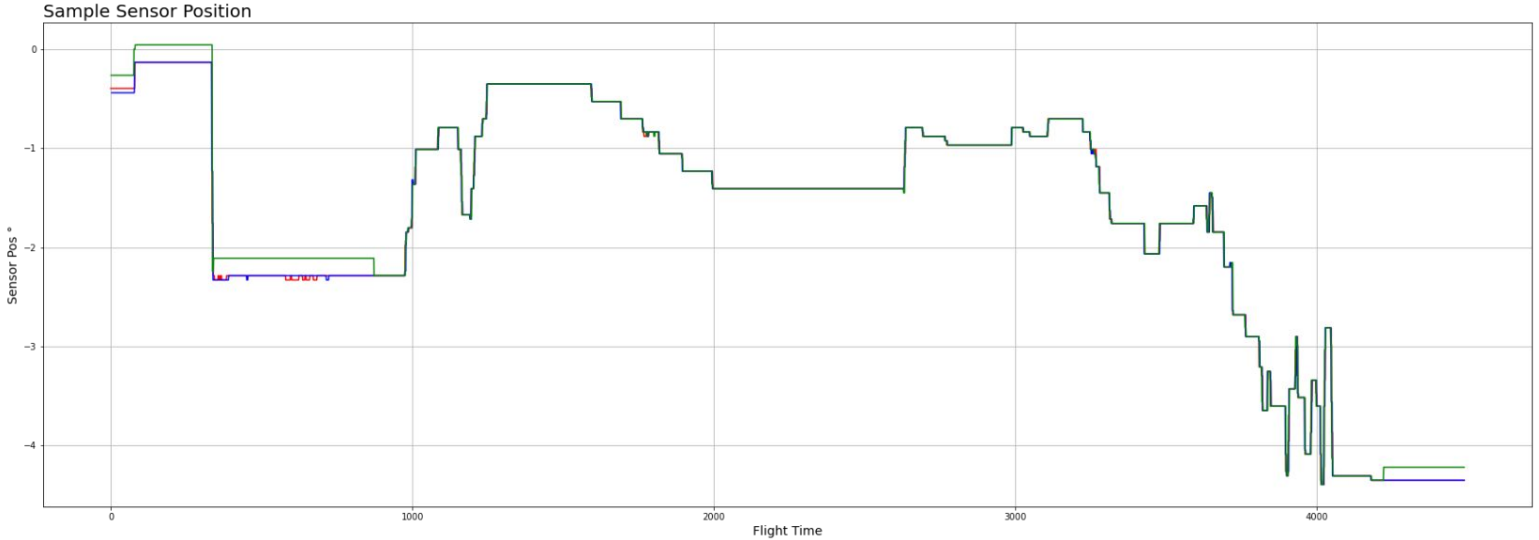
Remove/rename columns as required

```python
# Required Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import time
import multiprocessing
```

```python
# Read in data to list
datalist = (os.listdir('ga-final-data/Stab Pos'))
dataset = [pd.read_csv(f'ga-final-data/Stab Pos/{i}').ffill() for i in datalist]

# Remove Unnecessary Columns
dataset = [i.drop(columns=[
    'Baro-Corrected Altitude (ft)',
    'Pressure Altitude (ft)',
    'ENG N1-ACTUAL - LEFT (%RPM)',
    'ENG N1-ACTUAL - RIGHT (%RPM)',
], axis=1) for i in dataset]

# Rename Columns to be more manageable
dataset = [i.rename(columns= {
    'Stab Pos -A/P FCC L (Deg)':'pos_l',
    'Stab Pos -A/P FCC C (Deg)':'pos_c',
    'Stab Pos -A/P FCC R (Deg)':'pos_r',
    'Stab Pos R/C Disagree (Deg)': 'r/c_disagree',
    'Stab Pos L/C Disagree (Deg)': 'l/c_disagree',
    'Stab Pos L/R Disagree (Deg)': 'l/r_disagree'
}) for i in dataset]
```

Let's have a look

```
# Plot a sample Dataframe and save plot to file
fig, axs = plt.subplots(1, 1, figsize=(30, 10))
axs.plot(dataset[0]['pos_l'], color='red')
axs.plot(dataset[0]['pos_c'], color='blue')
axs.plot(dataset[0]['pos_r'], color='green')
axs.set_xlabel('Flight Time', size=14)
axs.set_ylabel('Disagree °', size=14)
axs.set_title('Sample Sensor Position', size=20, loc='left')
axs.grid()
plt.savefig('GA-Python-Final/Sample Sensor Position.png')
```

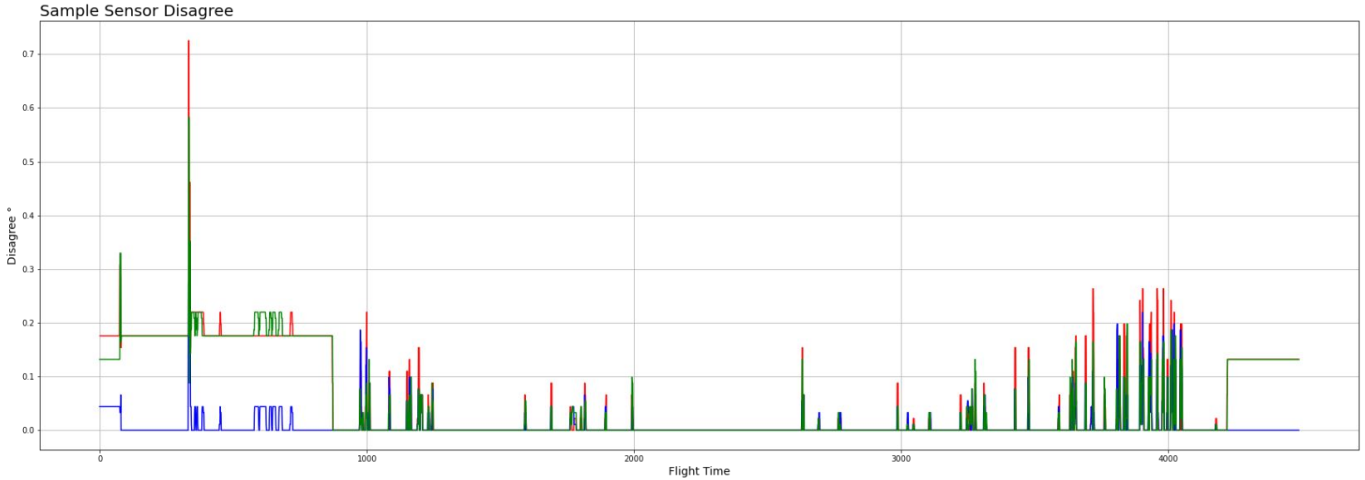


What does the in-built “disagree” value look like?

```python
# Plot a sample disagree Dataframe from source and save plot to file
fig, axs = plt.subplots(1, 1, figsize=(30, 10))
axs.plot(dataset[0]['r/c_disagree'], color='red')
axs.plot(dataset[0]['l/c_disagree'], color='blue')
axs.plot(dataset[0]['l/r_disagree'], color='green')
axs.set_xlabel('Flight Time', size=14)
axs.set_ylabel('Disagree °', size=14)
axs.set_title('Sample Sensor Disagree', size=20, loc='left')
axs.grid()
plt.savefig('GA-Python-Final/Sample Sensor Disagree.png')
```

Here, we’re looking at just the first dataframe in the set.

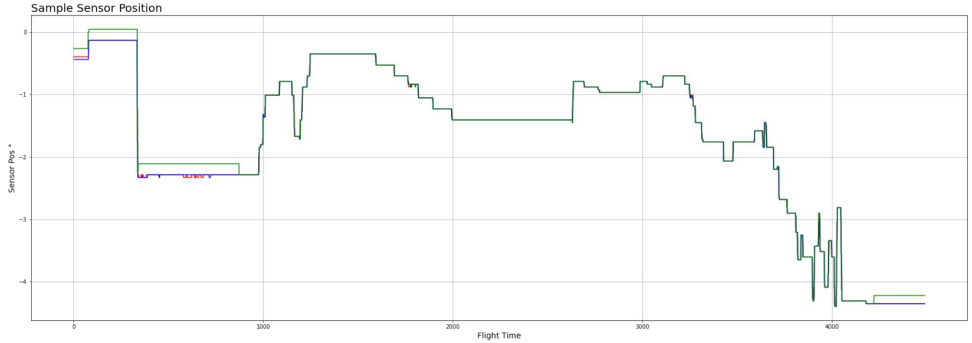
Note the spike to .7°

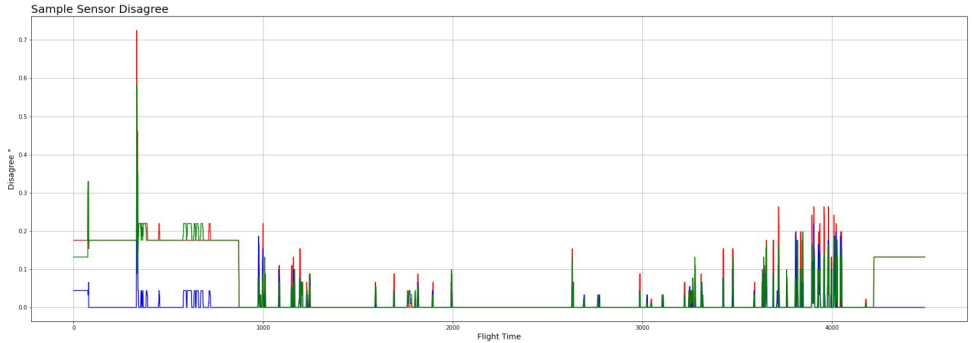


That’s not a good look…

Lets see those spikes side-by-side:

We can see that the spike is not representative of the sensor positions. This is likely because the samples are taken every four seconds, and not at the same time. So, as one sensor transitions, the other will lag for an entire sample period.



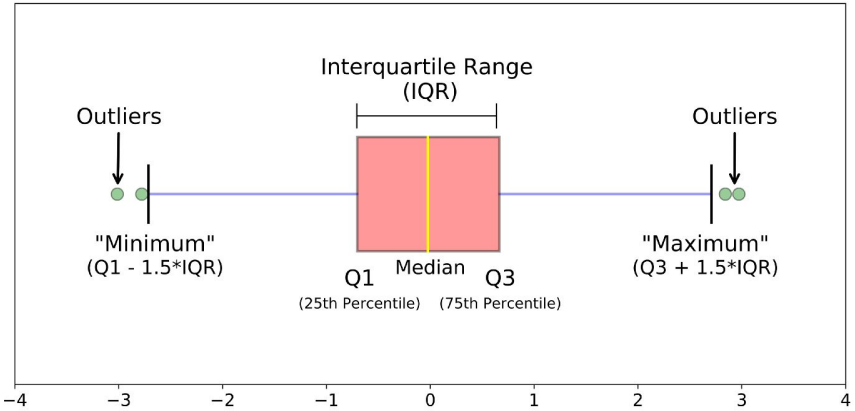


So, what now?

Using peak values to determine when a sensor has failed is insufficient. Calculations and some data-scrubbing will be required to handle this. My first instinct was to write a function that would drop a row if a value exceeded its previous iteration by more than double that previous value. I had difficulty with this and so I went a different way.

Box plots?

No, I didn’t use a box-plot directly. However, I used some math from how they work.



We’ll use the “maximum” value, which discludes outliers.

However...

In performing the following calculations, this is where I first noticed that the in-built “disagree” values were not correct, and not only because of the sampling rates. The math is outright incorrect. So, I had to find the “true” disagree values.

What does that look like?

We need to calculate the “maximum,” like this:

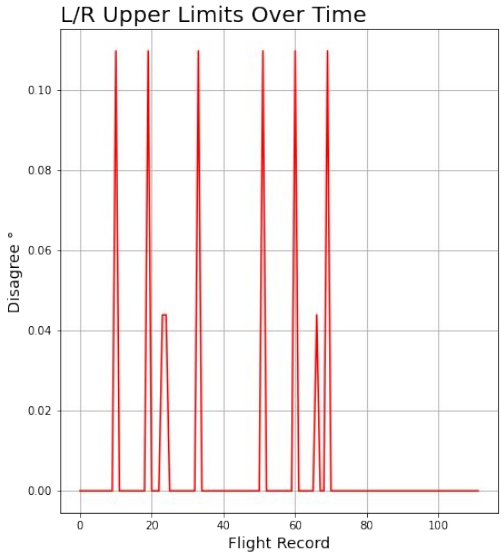
```python
# Returns upper end of Box+Whisker Plot
def get_topend(series):
    q1 = series.quantile(.25)
    q3 = series.quantile(.75)
    iqr = q3-q1
    upper = q3+(1.5*iqr)
    return upper
```

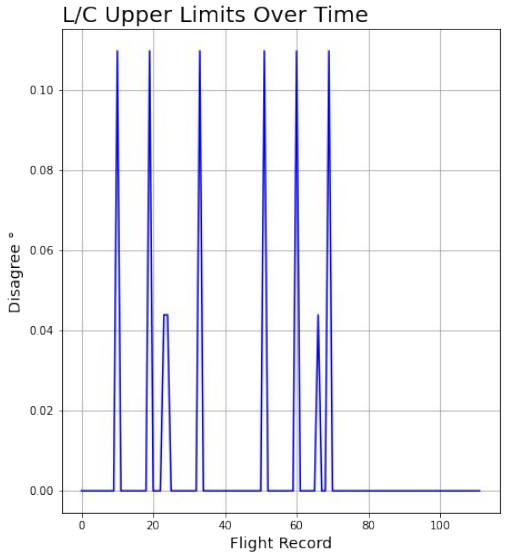
Then we find our true disagree values both apply the function to our data, like this:

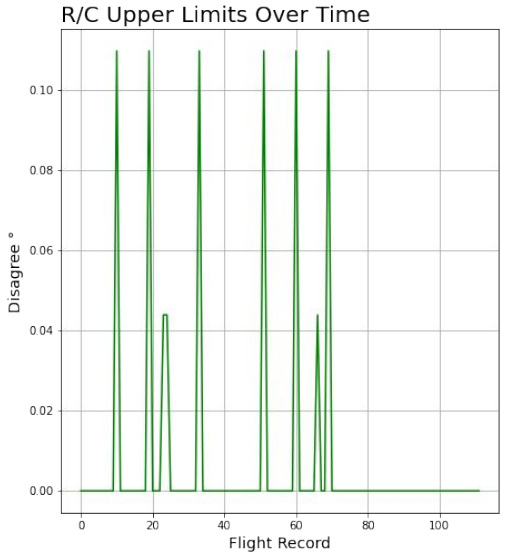
```python
# Find true difference between sensors
for frame in dataset:
    frame['true l/c disagree'] = abs(frame['pos_l']-frame['pos_c'])
    frame['true l/r disagree'] = abs(frame['pos_l']-frame['pos_c'])
    frame['true r/c disagree'] = abs(frame['pos_l']-frame['pos_c'])

# Gathers all true upper ends into a single Dataframe
upper_rc = pd.DataFrame([get_topend(i['true r/c disagree']) for i in dataset])
upper_lc = pd.DataFrame([get_topend(i['true l/c disagree']) for i in dataset])
upper_lr = pd.DataFrame([get_topend(i['true l/r disagree']) for i in dataset])
upperdf = upper_lc
upperdf.rename(columns={0:'upper l/c'}, inplace=True)
upperdf['upper l/r'] = upper_lr
upperdf['upper r/c'] = upper_rc

# Plots upper ends
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize = (24, 8))
ax1.plot(upperdf['upper l/r'], color='red')
ax2.plot(upperdf['upper l/c'], color='blue')
ax3.plot(upperdf['upper r/c'], color='green')
ax1.set_xlabel('Flight Record', size=14)
ax2.set_xlabel('Flight Record', size=14)
ax3.set_xlabel('Flight Record', size=14)
ax1.set_ylabel('Disagree °', size=14)
ax2.set_ylabel('Disagree °', size=14)
ax3.set_ylabel('Disagree °', size=14)
ax1.set_title('L/R Upper Limits Over Time', size=20, loc='left')
ax2.set_title('L/C Upper Limits Over Time', size=20, loc='left')
ax3.set_title('R/C Upper Limits Over Time', size=20, loc='left')
ax1.grid()
ax2.grid()
ax3.grid()

plt.savefig('GA-Python-Final/Calculated Sensor Disagree.png');
```

Pretty Pictures!

I had to make these separate plots because they are identical and so the last plotted line is the only visible line.





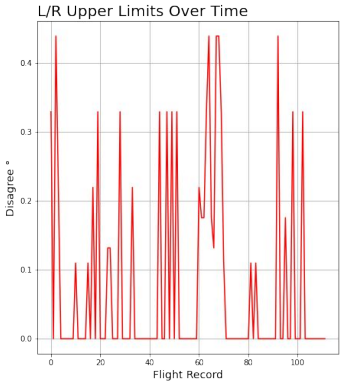


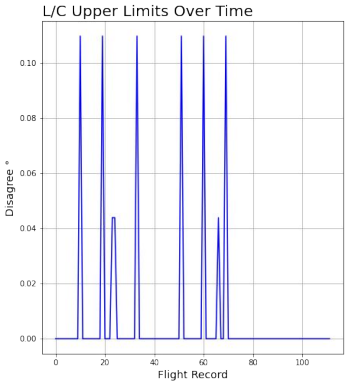
Wait! Oops...

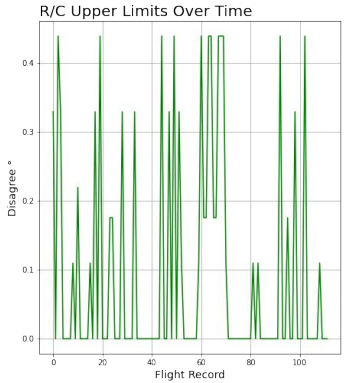
That's not supposed to look like that...
(I caught that while reviewing the Slides presentation)

```
for frame in dataset:
    frame['true l/c disagree'] = abs(frame['pos_l']-frame['pos_c'])
    frame['true l/r disagree'] = abs(frame['pos_l']-frame['pos_c'])
    frame['true r/c disagree'] = abs(frame['pos_l']-frame['pos_c'])
```

Those are identical metrics, so let's go again like this:








```
# Find true difference between sensors
for frame in dataset:
    frame['true l/c disagree'] = abs(frame['pos_l']-frame['pos_c'])
    frame['true l/r disagree'] = abs(frame['pos_l']-frame['pos_r'])
    frame['true r/c disagree'] = abs(frame['pos_r']-frame['pos_c'])

# Gathers all true upper ends into a single Dataframe
upper_rc = pd.DataFrame([get_topend(i['true r/c disagree']) for i in dataset])
upper_lc = pd.DataFrame([get_topend(i['true l/c disagree']) for i in dataset])
upper_lr = pd.DataFrame([get_topend(i['true l/r disagree']) for i in dataset])
upperdf = upper_lc
upperdf.rename(columns={0:'upper l/c'}, inplace=True)
upperdf['upper l/r'] = upper_lr
upperdf['upper r/c'] = upper_rc

# Plots upper ends
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize = (24, 8))
ax1.plot(upperdf['upper l/r'], color='red')
ax2.plot(upperdf['upper l/c'], color='blue')
ax3.plot(upperdf['upper r/c'], color='green')
ax1.set_xlabel('Flight Record', size=14)
ax2.set_xlabel('Flight Record', size=14)
ax3.set_xlabel('Flight Record', size=14)
ax1.set_ylabel('Disagree °', size=14)
ax2.set_ylabel('Disagree °', size=14)
ax3.set_ylabel('Disagree °', size=14)
ax1.set_title('L/R Upper Limits Over Time', size=20, loc='left')
ax2.set_title('L/C Upper Limits Over Time', size=20, loc='left')
ax3.set_title('R/C Upper Limits Over Time', size=20, loc='left')
ax1.grid()
ax2.grid()
ax3.grid()

plt.savefig('GA-Python-Final/Calculated Sensor Disagree.png');
```

Objection!

What’s with all those zero values?

Well, it turns out that, by and large across all of the flights, the 75th quantile is usually at or around zero. As a result, the math we did to find those maximum values based on quantile numbers returns a zero.

We must adapt!

We *want* outliers?

Yes! The outliers are the only values we can use at all! But how do we extract them?

We'll make a function, but I couldn't manage to work it out as a single function, so we'll do one for all three of our true disagree columns.

These work, but they are clunky and could use improvement for sure.

```
# Define lists for averages of the outliers
outlier_avg_lr = []
outlier_avg_lc = []
outlier_avg_rc = []
```

```
# Create functions for extracting the outliers for all disagree values and appending the average of these
# values to their appropriate lists for all dataframes
def get_outlier_avg_lr(series):
    for frame in dataset:
        outliers_lr = []
        for x in frame[series]:
            if x > get_topend(frame[series]):
                outliers_lr.append(x)
        if len(outliers_lr) == 0:
            continue
        else:
            outlier_avg_lr.append(sum(outliers_lr)/len(outliers_lr))

def get_outlier_avg_lc(series):
    for frame in dataset:
        outliers_lc = []
        for x in frame[series]:
            if x > get_topend(frame[series]):
                outliers_lc.append(x)
        if len(outliers_lc) == 0:
            continue
        else:
            outlier_avg_lc.append(sum(outliers_lc)/len(outliers_lc))

def get_outlier_avg_rc(series):
    for frame in dataset:
        outliers_rc = []
        for x in frame[series]:
            if x > get_topend(frame[series]):
                outliers_rc.append(x)
        if len(outliers_rc) == 0:
            continue
        else:
            outlier_avg_rc.append(sum(outliers_rc)/len(outliers_rc))
```

So what do we have here?

We have an egregiously long processing period is what we have.

I did one of the functions and it took about 34 minutes. But I goofed! I put the data into the wrong list and had to re-run it. So, I just did them all at once.

I tried to use some multiprocessing, but I couldn't tell you whether or not it actually worked.

Yes, that's 6205 seconds of processing time. This comes out to 1.7236hrs.

```
# Attempt to multiprocess all three functions at once
# This process went for roughly 1.75hrs
start = time.time()
p1 = multiprocessing.Process(get_outlier_avg_lr('true l/r disagree'))
p2 = multiprocessing.Process(get_outlier_avg_lc('true l/c disagree'))
p3 = multiprocessing.Process(get_outlier_avg_rc('true r/c disagree'))
p1.start()
p2.start()
p3.start()
end = time.time()
print(int(end - start))
```

```
6205
```

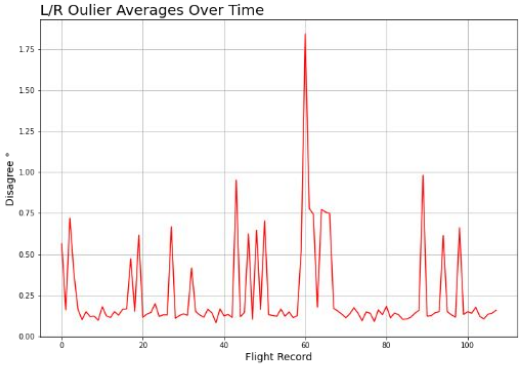
```
# Bring all of the outlier averages into a single dataframe
outlier_avg_lr_df = pd.DataFrame(outlier_avg_lr)
outlier_avg_lc_df = pd.DataFrame(outlier_avg_lc)
outlier_avg_rc_df = pd.DataFrame(outlier_avg_rc)
outlier_avg_df = outlier_avg_lr_df.rename(columns = {0: 'average l/r outliers'})
outlier_avg_df['average l/c outliers'] = outlier_avg_lc_df
outlier_avg_df['average r/c outliers'] = outlier_avg_rc_df
```

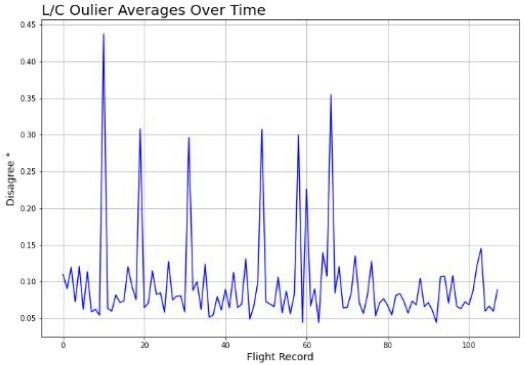
The Final Plot?

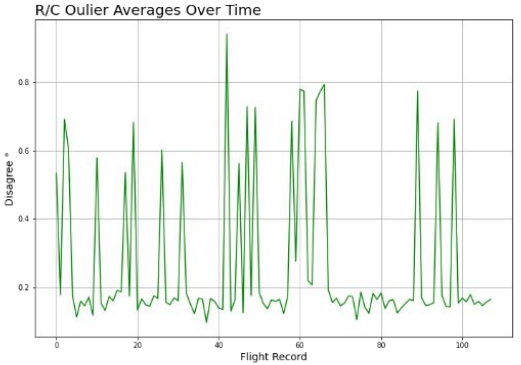
Not quite. If we are looking at the averages of the outliers, we get some nasty looking results.

So how about the .75 Quantile of the outliers?

```
# Plots the outlier averages over time
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize = (40, 8))
ax1.plot(outlier_avg_df['average l/r outliers'], color='red')
ax2.plot(outlier_avg_df['average l/c outliers'], color='blue')
ax3.plot(outlier_avg_df['average r/c outliers'], color='green')
ax1.set_xlabel('Flight Record', size=14)
ax2.set_xlabel('Flight Record', size=14)
ax3.set_xlabel('Flight Record', size=14)
ax1.set_ylabel('Disagree °', size=14)
ax2.set_ylabel('Disagree °', size=14)
ax3.set_ylabel('Disagree °', size=14)
ax1.set_title('L/R Oulier Averages Over Time', size=20, loc='left')
ax2.set_title('L/C Oulier Averages Over Time', size=20, loc='left')
ax3.set_title('R/C Oulier Averages Over Time', size=20, loc='left')
ax1.grid()
ax2.grid()
ax3.grid()

plt.savefig('GA-Python-Final/Calculated Outlier Average of Sensor Disagree.png');
```







Crunching, crunching, crunching…

We'll just let it crunch some numbers for another almost two hours…

```
outliers_quantile_lr = []
outliers_quantile_lc = []
outliers_quantile_rc = []
```

```
# Gets the .75 quantile of all the outlier values for all dataframes
def get_outlier_quantile_lr(series):
    for frame in dataset:
        outliers_lr = []
        for x in frame[series]:
            if x > get_topend(frame[series]):
                outliers_lr.append(x)
        if len(outliers_lr) == 0:
            continue
        else:
            outliers_quantile_lr.append(pd.DataFrame(outliers_lr).quantile(.75))

def get_outlier_quantile_lc(series):
    for frame in dataset:
        outliers_lc = []
        for x in frame[series]:
            if x > get_topend(frame[series]):
                outliers_lc.append(x)
        if len(outliers_lc) == 0:
            continue
        else:
            outliers_quantile_lc.append(pd.DataFrame(outliers_lc).quantile(.75))

def get_outlier_quantile_rc(series):
    for frame in dataset:
        outliers_rc = []
        for x in frame[series]:
            if x > get_topend(frame[series]):
                outliers_rc.append(x)
        if len(outliers_rc) == 0:
            continue
        else:
            outliers_quantile_rc.append(pd.DataFrame(outliers_rc).quantile(.75))
```

```
# Attempt to multiprocess all three functions at once
# This process went for roughly 1.75hrs
start = time.time()
p1 = multiprocessing.Process(get_outlier_quantile_lr('true l/r disagree'))
p2 = multiprocessing.Process(get_outlier_quantile_lc('true l/c disagree'))
p3 = multiprocessing.Process(get_outlier_quantile_rc('true r/c disagree'))
p1.start()
p2.start()
p3.start()
end = time.time()
print(int(end - start))
```

```
6134
```

```
# Bring all of the outlier averages into a single dataframe
outlier_quantile_lr_df = pd.DataFrame(outliers_quantile_lr)
outlier_quantile_lc_df = pd.DataFrame(outliers_quantile_lc)
outlier_quantile_rc_df = pd.DataFrame(outliers_quantile_rc)
outlier_quantile_df = outlier_avg_lr_df.rename(columns = {0: '.75 l/r outliers'})
outlier_quantile_df['.75 l/c outliers'] = outlier_avg_lc_df
outlier_quantile_df['.75 r/c outliers'] = outlier_avg_rc_df
```

```
# Plots the .75 quantile over time
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize = (40, 8))
ax1.plot(outlier_quantile_df['.75 l/r outliers'], color='red')
ax2.plot(outlier_quantile_df['.75 l/c outliers'], color='blue')
ax3.plot(outlier_quantile_df['.75 r/c outliers'], color='green')
ax1.set_xlabel('Flight Record', size=14)
ax2.set_xlabel('Flight Record', size=14)
ax3.set_xlabel('Flight Record', size=14)
ax1.set_ylabel('Disagree °', size=14)
ax2.set_ylabel('Disagree °', size=14)
ax3.set_ylabel('Disagree °', size=14)
ax1.set_title('L/R Oulier .75 Quantiles Over Time', size=20, loc='left')
ax2.set_title('L/C Oulier .75 Quantiles Over Time', size=20, loc='left')
ax3.set_title('R/C Oulier .75 Quantiles Over Time', size=20, loc='left')
ax1.grid()
ax2.grid()
ax3.grid()

plt.savefig('GA-Python-Final/Calculated Outlier .75 Quantile of Sensor Disagree.png');
```

Tim's Rescue

It turns out I'm a goober, and there was a much less resource-intensive way of doing those calculations. However, I'm new so Tim helped me see the error of my ways. Here's the revised code:

```
# Create functions for extracting the outliers for all disagree values and appending the average o
# values to their appropriate lists for all dataframes
def get_outlier_avg_lr(series):
    for frame in dataset:
        outliers_lr = []
        topend = get_topend(frame[series])
        for x in frame[series]:
            if x > topend:
                outliers_lr.append(x)
        if len(outliers_lr) == 0:
            continue
        else:
            outlier_avg_lr.append(sum(outliers_lr)/len(outliers_lr))

def get_outlier_avg_lc(series):
    for frame in dataset:
        outliers_lc = []
        topend = get_topend(frame[series])
        for x in frame[series]:
            if x > topend:
                outliers_lc.append(x)
        if len(outliers_lc) == 0:
            continue
        else:
            outlier_avg_lc.append(sum(outliers_lc)/len(outliers_lc))

def get_outlier_avg_rc(series):
    for frame in dataset:
        outliers_rc = []
        topend = get_topend(frame[series])
        for x in frame[series]:
            if x > topend:
                outliers_rc.append(x)
        if len(outliers_rc) == 0:
            continue
        else:
            outlier_avg_rc.append(sum(outliers_rc)/len(outliers_rc))
```

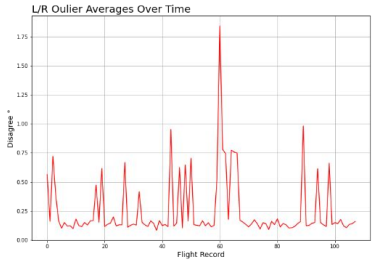
Note how the "topend" calculation takes place outside of the "for x" loop. Before, this calculation was being repeated for each cell of the .csv, essentially forcing the code to perform this math >150k times.

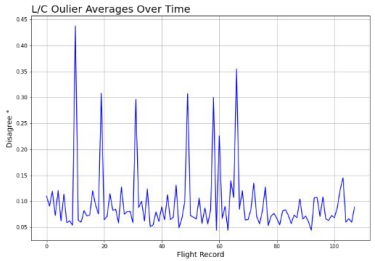
```
# Gets the .75 quantile of all the outlier values for all dataframes
def get_outlier_quantile_lr(series):
    for frame in dataset:
        outliers_lr = []
        topend = get_topend(frame[series])
        for x in frame[series]:
            if x > topend:
                outliers_lr.append(x)
        if len(outliers_lr) == 0:
            continue
        else:
            outliers_quantile_lr.append(pd.DataFrame(outliers_lr).quantile(.75))

def get_outlier_quantile_lc(series):
    for frame in dataset:
        outliers_lc = []
        topend = get_topend(frame[series])
        for x in frame[series]:
            if x > topend:
                outliers_lc.append(x)
        if len(outliers_lc) == 0:
            continue
        else:
            outliers_quantile_lc.append(pd.DataFrame(outliers_lc).quantile(.75))

def get_outlier_quantile_rc(series):
    for frame in dataset:
        outliers_rc = []
        topend = get_topend(frame[series])
        for x in frame[series]:
            if x > topend:
                outliers_rc.append(x)
        if len(outliers_rc) == 0:
            continue
        else:
            outliers_quantile_rc.append(pd.DataFrame(outliers_rc).quantile(.75))
```

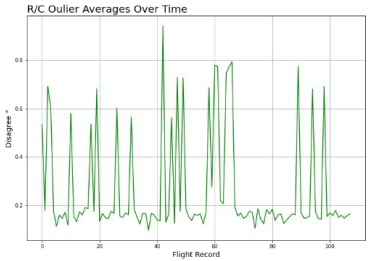
It looks... exactly the same?

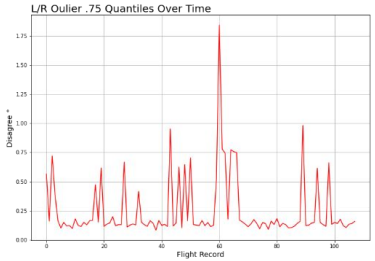
Yes, if there are differences, they're in too fine of a metric to be observed in these graphs. But, for now, we're finished trying to manipulate the dataset.

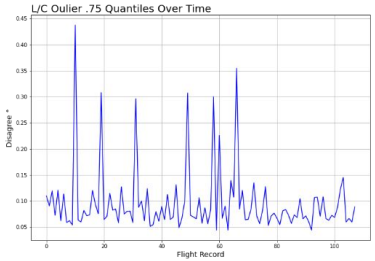
How about some observations? We can see that the baseline values in the L/R and R/C graphs are almost twice as high as the L/C graph. This leads us to believe that the Right Sensor is either compromised or will be soon. I believe there's more that could be done here, but we need more data.

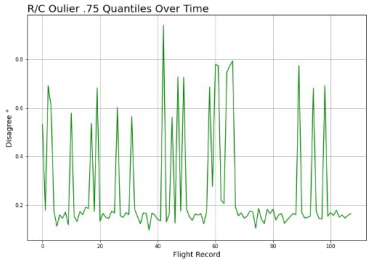












Conclusions

I can determine unequivocally that in order to make a predictive model, I need a metric ton of additional data. I would also need to know when these sensors are replaced by maintenance so I can slice and dice the data accordingly.

We can see that the small spikes between the Left and Center sensors are within our tolerance range, but the Right sensor to either of the other two are particularly high, perhaps indicating a bad sensor or one that will be soon.

I'm still not decided on which metric - either maximums, outliers, or outlier quantiles - are best to utilize for a future model. But having all of them together here is useful tool for making this decision in the future.

I am thrilled to have accomplished what I've done here already, and I'm greatly looking forward to developing this and other parameters into models we can use to predict the future!