Initial Input

To make it clearer, I separated the user input into separated lines.

```
PS D:\SIM-Wollongong\CSCI262-Systems Security\Assignment\A3\A3> python main.py
Welcome to IDS
Enter filename for Events: Events.txt

Enter filename for Stats: Stats.txt

Enter the number of Days: 10
```

Some validations with regards to the three inputs above have been implemented.
For both Events and Stats file, the program checks whether the file entered exists within the system's directory. If not, the program will prompt the user to enter another file.
For the number of days, the days entered must be an integer and must be more than 0.

```
Welcome to IDS
Enter filename for Events: WrongFile.txt
Please enter the correct Events File again
Enter filename for Events: Events.txt

Enter filename for Stats: WrongFile.txt
Please enter the correct Stats File again
Enter filename for Stats: Stats.txt

Enter the number of Days: 12a
Please enter a whole number
Enter the number of Days: 10
```

After reading the two files, the program will store the data in memory, in a list, in the following format:

```
From Events.txt: ['5', 'Logins:0:0:100:3:', 'Time online:0:0:1440:2:', 'Emails sent:0:0:100:1:', 'Emails opened:0:0:100:1:', 'Emails deleted:0:0:100:2:']
From Stats.txt: ['5', 'Logins:4:1.5:', 'Time online:150.5:25.00:', 'Emails sent:10:3:', 'Emails opened:12:4.5:', 'Emails deleted:7:2.25:']
```

A simple initial consistency check between the number of events in each file is performed afterwards.

```python
# check for inconsistencies between Events and Stats file
def consistencyCheck(eventData, statsData):
    noOfEventData = int(eventData[0])
    noOfStatsData = int(statsData[0])

    # different number of data
    if noOfEventData != noOfStatsData:
        print("Number of data is inconsistent")

    # loop through array of data and check the consistency between each event
    for i in range(1, noOfEventData):
        # different event
        if eventData[i].split(":")[0] != statsData[i].split(":")[0]:
            print(f"Inconsistencies found in line {i + 1}")
            return False
```

```
PROBLEMS  OUTPUT  TERMINAL  JUPYTER  DEBUG CONSOLE

------------------------------------------------------------
Checking for inconsistencies between Events.txt and Stats.txt...
------------------------------------------------------------
No inconsistencies found
```

If the files pass this check, the program will go on the process the data extracted from the events and stats file, while performing some validation along the way for example, if the event type is discrete, the min/max values must be an integer.

Activity Simulation Engine and the Logs

Activity Simulation

Program will generate data as close as possible to the mean and standard deviation specified in the Stats file. (please zoom in)

```python
# generate set of data as close to mean and stdev
def generateData(mean, standardDeviation, days, minimum, maximum, eventType):
    while True:
        n = s.NormalDist(mu = mean, sigma = standardDeviation)
        samples = n.samples(days)

        # keep value within min/max bounds
        for index in range(len(samples)):
            if eventType == "D": # discrete event
                samples[index] = round(samples[index]) # round value to integer

            if eventType == "C": # continuous event
                samples[index] = round(samples[index], 2) # round value to 2 decimal places

            # check for out of bounds
            if samples[index] < minimum or samples[index] > maximum:
                continue

        if days >= 10:
            # keep mean and standard deviation within 0.5% +/-
            if s.mean(samples) < mean * 0.95 or s.mean(samples) > mean * 1.05 or s.stdev(samples) < standardDeviation * 0.95 or s.stdev(samples) > standardDeviation * 1.05:
                continue # continue looping and generating new sample data
            else:
                return samples # return generated sample data
        else:
            # keep mean and standard deviation within 1% +/-
            if s.mean(samples) < mean * 0.9 or s.mean(samples) > mean * 1.1 or s.stdev(samples) < standardDeviation * 0.9 or s.stdev(samples) > standardDeviation * 1.1:
                continue # continue looping and generating new sample data
            else:
                return samples # return generated sample data
```

I made use of the statistics import from Python to access statistics.NormalDist method which takes in the mean and standard deviation as arguments to generate a set of data based on the number of days specified. Because the lesser the data, the harder it is to generate close approximates, so for days less than 10, I allow a 10% +/- deviation from the specified mean and standard deviations. For days more than 10, I allow a 5% +/- deviation instead.
While the generated samples fall outside the stated deviation above, the program will continue generating new samples until the parameters have been met.

Logs File

For the logs file, the first thing is the day number, followed by the number of events, and then the events itself. For each event, it includes the name of the event, the event type, and the total number for that specific event. After the last event, I included a new line to separate each day more clearly

logs.txt

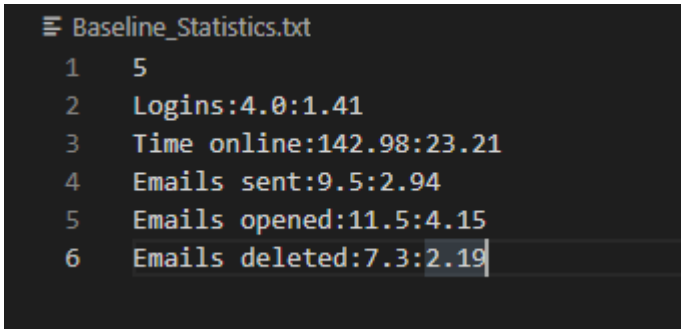
```
Day 1
5
Logins:D:4:
Time online:C:167.59:
Emails sent:D:12:
Emails opened:D:5:
Emails deleted:D:10:

Day 2
5
Logins:D:4:
Time online:C:127.32:
Emails sent:D:6:
Emails opened:D:16:
Emails deleted:D:6:
```

Analysis Engine

This part of the program will first read in the log file generated in the previous step, then returns the data extracted from the logs file and then output the data into another text file which I titled as "Baseline_Statistics.txt".

The format for the baseline statistics file generated follows the same format for the first Stats file that was read in during the 'Initial Input' part of the program.



Because the format for the logs file is standardised, I can use the pattern to read and store the data based on the number of events.

```
with open(filename, "r") as fin:
    # total lines per day in logs #
    # first line - Day number
    # second line - number of events
    # inbetween = number of events
    # last line - \n
    # total number of lines per day = 3 + number of events
```

While the program attempts to output the relevant data to the baselines statistics file, it will calculate the mean, and standard deviation of the data extracted from the logs file. The mean and standard deviation will then be stored in memory for future use.

Alert Engine

The program will run this function to read the new logs file to get the daily totals and use the values to calculate the anomaly counter.

```python
# for each event:
#   [total - mean(from Baseline.txt)] / stddev(from Baseline.txt) * weight(from Events.txt)
def anomalyCounter(filename, weight, mean, stddev):
    print("Currently calculating daily totals...\n.\n.\n.")

    # read new logs file and capture its data
    data = readNewLogs(filename) # data = [[], [], ..., []]

    # track the daily totals
    dailyCounter = []

    for index, value in enumerate(data): # loop through the number of days
        counter = 0 # track counter for daily events
        for i, v in enumerate(value): # loop through the number of events
            counter += round(((abs((float(v) - mean[i])) / stddev[i]) * weight[i]), 2)

        dailyCounter.append(counter)

    print("Daily totals calculated!\n")
    return dailyCounter
```

The program will then run the next function which is to put the anomaly counter up against the threshold to too whether it exceeds the threshold.

```python
# check for anomaly
def flagging(data, threshold):
    print("Currently checking for anomalies...\n.\n.\n.")

    flagged = []

    for i in range(len(data)):
        alert = False
        if data[i] > threshold:
            flagged.append(i + 1) # day number
            alert = True

        print(f"Day {i + 1} anomaly count = {round(data[i], 2)} {'--- FLAGGED' if alert == True else ''}")

    print("\n")

    # alert
    if len(flagged) != 0:
        print("ALERT! Anomalies detected!")
        print("--------------------------")
        for index, value in enumerate(flagged):
            print(f"Day {value} has been flagged!")
        print("\n")
    else:
        print("Congratulations! There are no anomalies!\n")
```

If the anomaly counter exceeds the threshold, the program will show the days flagged in the command line below:

```
Commencing analysis for logs2.txt...

Daily totals calculated!


Threshold: 18
Currently checking for anomalies...
.
.
.
Day 1 anomaly count = 25.94 --- FLAGGED
Day 2 anomaly count = 9.25
Day 3 anomaly count = 9.28
Day 4 anomaly count = 13.3
Day 5 anomaly count = 9.03
Day 6 anomaly count = 12.87
Day 7 anomaly count = 7.96
Day 8 anomaly count = 9.46
Day 9 anomaly count = 16.29
Day 10 anomaly count = 12.74
Day 11 anomaly count = 10.04
Day 12 anomaly count = 22.43 --- FLAGGED
Day 13 anomaly count = 22.44 --- FLAGGED
Day 14 anomaly count = 12.62
Day 15 anomaly count = 15.34
Day 16 anomaly count = 11.38
Day 17 anomaly count = 19.06 --- FLAGGED
Day 18 anomaly count = 17.04
```