Initial Input|Event Identifier

```
class EventIdentifier:
    def __init__(self, eventName, eventType, minValue, maxValue, weight):
        self.eventName = eventName
        self.eventType = eventType
        self.minValue = minValue
        self.maxValue = maxValue
        self.weight = weight
        self.mean = (minValue + maxValue) / 2
        self.standardDev = (maxValue - minValue) / 2
```

Events are read from Events.txt using the read_event_text function and stored in a list.
Each event is stored as an instance of the EventIdentifier class.
The class stores event attributes: eventName, eventType (C for continuous, D for discrete), minValue, maxValue, and weight. Additionally, it calculates mean and standardDev based on the minValue and maxValue.

```
    for event in event_types:
        if event.get_event_name() == event_name:
            event.set_mean(mean_value)
            event.set_standard_dev(standard_dev_value)
```

Statistics are read from Stats.txt using the read_stats_text function.
The stats file contain Event name:mean:standard deviation
For each Event name in events will check with stats event name
Before setting the mean value and standard deviation in each specific Event

```
# Check if the counts match
if events_count != stats_count:
    print(
        f"Error: The number of events in Events.txt ({events_count}) does not match the number in Stats.txt ({stats_count}).")
    sys.exit(-1)
```

```
if not match_found:
    print(f"Error: Event name '{event_name}' in Stats.txt does not match any event in Events.txt.")
    sys.exit(-1)
```

This is the check to check for the first line that contains the number of events being monitored if both Events.txt and Stats.txt are the same. The read_stats_text function also checks the events is the same as Events.txt if not an error message will show.

Potential consistency checks that are not included in the program:

Weights in Events.txt must be specified
Mean and Standard Deviation in Stats.txt must be specified

Activity Engine

In order to generate events in accordance with the statistics outlined in Stats.txt, the random.gauss() method is employed. This method, by default, generates a normally distributed double value with a mean of 0.0 and a standard deviation of 1.0. To adapt random.gauss() to our specific statistical requirements, the formula used is: mean + (standardDeviation * random.gauss(0, 1)).

```
def get_rand_value(self):
    while True:
        value = self.mean + self.standardDev * random.gauss( mu: 0,  sigma: 1)
        if self.eventType == 'D':
            value = round(value)
        if self.minValue <= value <= self.maxValue:
            return value
```

```
Enter StatsFile and Days in the format (StatsFile.txt Days || Or press 1 to quit): newStats.txt 5
```

```python
class ActivitySimuEngine:
    def __init__(self, eventStats):
        self.eventStats = eventStats

    2 usages
    def random_time(self):
        hour = random.randint( a: 0,  b: 23)
        minute = random.randint( a: 0,  b: 59)
        second = random.randint( a: 0,  b: 59)
        return f"{hour:02d}-{minute:02d}-{second:02d}"

    2 usages
    def logs_generator(self, days, prefix):
        for i in range(days):
            print(f"Stimulating Events for Day {i} ...")

            event_log = []
            for event in self.eventStats:
                if event.eventType == 'C':
                    value = event.get_rand_value()
                    event_log.append(f"{self.random_time()}:{event.eventName}:{value:.2f}")
                elif event.eventType == 'D':
                    events = round(event.get_rand_value())
                    for _ in range(events):
                        event_log.append(f"{self.random_time()}:{event.eventName}:1")

            event_log.sort()
            with open(f"{prefix}{i}.txt", 'w') as file:
                for line in event_log:
                    file.write(line + '\n')

        print(f"Completed Stimulating Events for {days} Days\n")
```

The process:
The constructor takes eventStats, a list of EventIdentifier objects, as an input.

Generates a random timestamp with hour, minute, and second components. This is used to timestamp the events in the logs

Iterates over each day, for each day, it creates an empty list event_log to store the events of that day.

Iterates over each event type in eventStats:
If the event is continuous (eventType == 'C'), it generates a random value within the defined range and appends it to event_log with the event name and timestamp

Else If the event is discrete (eventType == 'D'), it generates a round number of occurrences of that event and adds each occurrence to event_log with the event name and timestamp.

Sorts the event_log to ensure events are listed in chronological order.

Writes the events of the day to a file named with the given prefix and the day number.
File name : {prefix}_{daycounter}
Format: Time | Event name | value

```
00-02-14:Emails deteled:1
01-12-22:Time online:169.93
01-21-19:Emails deteled:1
02-07-47:Emails opened:1
02-20-09:Emails sent:1
02-49-11:Logins:1
03-17-10:Emails deteled:1
03-40-59:Emails opened:1
04-24-39:Emails opened:1
06-00-35:Emails sent:1
06-20-56:Emails sent:1
06-46-54:Emails deteled:1
07-06-21:Logins:1
09-15-04:Logins:1
09-30-59:Emails deteled:1
09-44-49:Emails opened:1
11-21-02:Emails sent:1
11-30-55:Emails sent:1
12-16-01:Emails opened:1
13-08-30:Emails opened:1
14-12-19:Emails opened:1
14-50-46:Emails opened:1
15-22-18:Emails sent:1
15-50-55:Emails opened:1
16-45-47:Emails sent:1
17-40-06:Emails deteled:1
18-36-44:Logins:1
18-47-18:Emails opened:1
19-33-12:Emails opened:1
```

Timestamp is generated random generated numbers for hours being in the range of 0-23, minutes and seconds in the range of 0-59.

Analysis Engine

```python
class AnalysisEngine:
    def __init__(self, eventStats):
        self.eventStats = eventStats

    2 usages
    def event_day_calculation(self, filename):
        event_counts = [0] * len(self.eventStats)

        with open(filename, 'r') as file:
            for line in file:
                delimited = line.strip().split(":")

                if len(delimited) != 3:
                    continue

                eventName = delimited[1]
                eventValue = float(delimited[2])
                for j, event in enumerate(self.eventStats):
                    if event.get_event_name() == eventName:
                        event_counts[j] += eventValue

        return event_counts
```

```python
def logs_processor(self, days, prefix):
    all_event_counts = []
    for i in range(days):
        event_counts = self.event_day_calculation(f"{prefix}_{i + 1}.txt")
        all_event_counts.append(event_counts)

        with open(f"{prefix}_{i + 1}_stats.txt", 'w') as file:
            for j, event in enumerate(self.eventStats):
                file.write(f"{event.get_event_name()}:{event_counts[j]}\n")

    calculated_events = EventIdentifier.clone_events(self.eventStats)
    with open(f"total_{prefix}_stats.txt", 'w') as file, open(f"{prefix}_events.txt", 'w') as file1:
        for j, event in enumerate(self.eventStats):
            sum_counts = sum(counts[j] for counts in all_event_counts)
            mean = sum_counts / days

            sum_diffs = sum((counts[j] - mean) ** 2 for counts in all_event_counts)
            std = math.sqrt(sum_diffs / days)

            calculated_events[j].set_mean(mean)
            calculated_events[j].set_standard_dev(std)

            print(f"Stats generated for {event.get_event_name()}:\nMean: {mean:.2f}\nStandard Deviation: {std:.2f}\n")
            file.write(f"{event.get_event_name()}:{mean:.2f}:{std:.2f}\n")

            # Additional calculations
            max_value = int(mean + std)
            min_value = int(round(mean - std))

            cont_value = round(min_value + (max_value - min_value) * random.random(), 2)
            disc_value = random.randint(min_value, max_value + 1)

            print(f"Continuous Baseline of {event.get_event_name()}: {cont_value}")
            print(f"Discrete Baseline of {event.get_event_name()}: {disc_value}\n")
            print("=================================================================\n")
            file1.write(f"{event.get_event_name()}:C:{min_value}:{max_value}:{cont_value}\n")
            file1.write(f"{event.get_event_name()}:D:{min_value}:{max_value}:{disc_value}\n")

    print("Analysis of Events Completed\n")
    return calculated_events
```

Generates a log with the same format as Stats.txt

{prefix}_{count} : {total_event_base_day}_{1} is per day events
{prefix} : {total_events_base_day} is the sum of all the base day events

```
Logins:3.20:1.47
Time online:134.19:23.40
Emails sent:11.00:2.61
Emails opened:10.80:5.15
Emails deteled:6.40:0.80
```

Alert Engine

This section is where by the newStatsfile is entered, this is to check consistency/anomaly between the live data and baseline data.

```python
class AlertEngine:
    def __init__(self, baseStats):
        self.baseStats = baseStats

    1 usage
    def alert_detector(self, days, prefix):
        analysis_engine = AnalysisEngine(self.baseStats)

        sum_weights = sum(event.get_weight() for event in self.baseStats)
        threshold = 2 * sum_weights
        print(f"Start alert analysis, detecting anomaly\nAnomaly threshold: {threshold}")
        alert = False

        for i in range(days):
            day_totals = analysis_engine.event_day_calculation(f"{prefix}_{i + 1}.txt")

            anomaly = 0
            for j, base_stat in enumerate(self.baseStats):
                temp = abs(
                    (day_totals[j] - base_stat.get_mean()) * base_stat.get_weight() / base_stat.get_standard_dev())
                anomaly += temp

            if anomaly > threshold:
                print(f"Day {i + 1}: Anomaly detected || weight: {anomaly:.2f}")
                alert = True
            else:
                print(f"Day {i + 1}: No Anomaly detected || weight: {anomaly:.2f}")

        if not alert:
            print("No Anomaly detected")
```

```
temp = abs(
    (day_totals[j] - base_stat.get_mean()) * base_stat.get_weight() / base_stat.get_standard_dev())
anomaly += temp
```

temp is the absolute value of the deviation of the day's total events (day_totals[j]) from the baseline mean (base_stat.get_mean()), multiplied by the event's weight (base_stat.get_weight()), and normalized by the event's standard deviation (base_stat.get_standard_dev())

The anomaly variable is a sum of all these temp values for a given day. It accumulates the weighted, normalized deviations for each event type, giving a composite measure of how much that day's event pattern deviates from the norm.

```
threshold = 2 * sum_weights
```

Threshold for detecting an intrusion is 2 × (*Sums of weights*)

```
Analysis of Events Completed

Start alert analysis, detecting anomaly
Anomaly threshold: 18.0
Day 1: Anomaly detected || weight: 21.37
Day 2: Anomaly detected || weight: 19.38
Day 3: Anomaly detected || weight: 20.52
Day 4: Anomaly detected || weight: 27.18
Day 5: No Anomaly detected || weight: 9.58
Enter new Stats.txt and Days in the format (new Stats.txt Days || Or press 0 to quit):
```

Output

Python A3.py Events.txt Stats.txt 5

```
PS B:\SIM UOW\CSCI262\Assignment\CSCI262_A3_ChowKeiren_7233450> python A3.py Events.txt Stats.txt 5
Generating Events from Stats.txt based on number of Days

Stimulating Events for 5 Days
Completed Events Stimulation for 5 Days

Analyzing Events from Stats.txt based on number of Days

Stats generated for Logins:
Mean: 4.00
Standard Deviation: 1.10

Continuous Baseline of Logins: 3.95
Discrete Baseline of Logins: 6

========================================================================

Stats generated for Time online:
Mean: 156.32
Standard Deviation: 13.18

Continuous Baseline of Time online: 143.48
Discrete Baseline of Time online: 157

========================================================================

Stats generated for Emails sent:
Mean: 9.40
Standard Deviation: 2.06

Continuous Baseline of Emails sent: 7.22
Discrete Baseline of Emails sent: 7

========================================================================

Stats generated for Emails opened:
Mean: 12.80
Standard Deviation: 3.87

Continuous Baseline of Emails opened: 15.82
Discrete Baseline of Emails opened: 13

========================================================================

Stats generated for Emails deleted:
Mean: 7.20
Standard Deviation: 2.14

Continuous Baseline of Emails deleted: 5.6
Discrete Baseline of Emails deleted: 9

========================================================================

Analysis of Events Completed

Enter new Stats.txt and Days in the format (new Stats.txt Days || Or press 0 to quit):
```

New Stats.txt 5

```
Enter new Stats.txt and Days in the format (new Stats.txt Days || Or press 0 to quit): newStats.txt 5
newStats.txt 5
Generating Events from new Stats.txt based on number of Days

Stimulating Events for 5 Days
Completed Events Stimulation for 5 Days

Analyzing Events from new Stats.txt based on number of Days

Stats generated for Logins:
Mean: 5.60
Standard Deviation: 2.15

Continuous Baseline of Logins: 6.93
Discrete Baseline of Logins: 7

======================================================================

Stats generated for Time online:
Mean: 207.95
Standard Deviation: 8.57

Continuous Baseline of Time online: 212.73
Discrete Baseline of Time online: 200

======================================================================

Stats generated for Emails sent:
Mean: 13.80
Standard Deviation: 4.45

Continuous Baseline of Emails sent: 13.52
Discrete Baseline of Emails sent: 11

======================================================================

Stats generated for Emails opened:
Mean: 15.20
Standard Deviation: 2.79

Continuous Baseline of Emails opened: 16.48
Discrete Baseline of Emails opened: 13

======================================================================

Stats generated for Emails deleted:
Mean: 10.60
Standard Deviation: 2.24

Continuous Baseline of Emails deleted: 8.95
Discrete Baseline of Emails deleted: 9

======================================================================

Analysis of Events Completed

Start alert analysis, detecting anomaly
Anomaly threshold: 18.0
Day 1: Anomaly detected || weight: 21.37
Day 2: Anomaly detected || weight: 19.38
Day 3: Anomaly detected || weight: 20.52
Day 4: Anomaly detected || weight: 27.18
Day 5: No Anomaly detected || weight: 9.58
Enter new Stats.txt and Days in the format (new Stats.txt Days || Or press 0 to quit):
```