Assignment 3 Report

Prepared by: Foo Whye Keat

7432628

Initial input

```
for line in events_in:
        event_count +=1
        components = line.strip().split(':')

        event_name = components[0]
        event_type = components[1]
        min_value = components[2] if components[2] else None
        max_value = components[3] if components[3] else None
        weight = components[4]

        event_details.append([event_name,event_type,min_value,max_value,weight])
```

The initial inputs are being stored in an array list as array types. Since the information is delimited with a semi colon, we can split the lines by the semi colon and store the information as an array. Since the values in a single line is fixed (event name: event type: min : max: weight), we can call on the corresponding indexes to get the data we need. Once this data is stored in variables, I then append it as an array into the array list.

```
stat_count = 0
for line in stats_in:
        stat_count +=1
        components = line.strip().split(':')

        event_name = components[0]
        mean = components[1]
        standard_dev = components[2]

        stat_details.append([event_name,mean,standard_dev])
```

The same is done for extracting data from the stats.txt file. The information in a single line is fixed (event name: mean: standard deviation) so once we split it the information we can call on the corresponding indexes to get the data I need. The input engine function will return this two array lists for the program to carry on

Inconsistencies

Majority of the inconsistencies are based on data integrity, which means ensuring that the data that we are told we are getting is actually accurate. We begin with the inconsistencies that I checked for.

```
    events_in = open(events,'r')
    num_events = events_in.readline().strip()

    stats_in = open(stats, 'r')
    num_stats = stats_in.readline().strip()


    event_details = []
    stat_details = []
    #checking that the given number of events matches the given
:r of stats
    if num_stats == num_events:
```

The simplest one to check for is ensuring that the number of "events" is the same in the events.txt and stats.txt file. This can be done by checking the number in the first line in both files, the program then carries on if the numbers match.

However, it is very easy to manipulate the number, meaning the number can say that there are 5 events in total in the txt file when there isn't. Because of this, I also implemented a line counter that counts how many lines have been read in for the events.txt and stats.txt file. This counter is then compared after all the lines in both files have been read in to ensure that the initial event number is correct, the actual number of events is correct and both files actually have the same number of events.

```
if ((len(event_name) == 0) or (event_type not in ['C','D']) or (len(weight) == 0)):
        return False,False
```

Next, we are checking that the data extracted from each line is correct. We check that there must be an event name in the line, the only acceptable types of data is either continuous or discrete and that there must be a weight to an event. Min and Max values are left out because they can be empty and if they are empty, it means there isn't any floor or limit to it.

Other inconsistencies

Checking that the order of the events are the same in the events.txt and stats.txt file. Given my code is running based on the assumption that the first event in the event.txt file corresponds to the data in the first event it is necessary to check that the event name and data matches in the stats.txt and events.txt file. This validation is instead done in my simulation engine when the data is being handled.

Activity sim and logs

The process used to generate events approximately consistent with the particular distribution is done based on Z-score. Z-score is a measure of how far a particular data points is from the mean group of data points expressed in terms of standard deviation. The formula to get z-score will be as follows, Z = ((target - mean)/standard deviation). In my program, I start by using a random number generator to generate me a set of numbers based on the number of days that is chosen by the user. From this set of random numbers, I then calculate the z-score of each number. Since I am given the mean and standard deviation of each event, I am then able work backwards by multiplying the z-score by the desired standard deviation and then adding the desired mean. This transformation scales and shifts the distribution to match the desired mean and standard deviation. This produces a target value for me that is very close to the parameters bounded by the baseline mean and standard deviation. This can be done for both continuous and discrete data. In my program I am using 2 different random number generators, one set for discrete data and one set for continuous data, this provides and additional form of randomness. This not only makes sure that data generated is approximately consistent with the particular distribution but also that the data is realistic.

```
random_numbers_d = random.sample(range(10001),days)
random_numbers_c = random.sample(range(10001),days)

#average or random numbers
average_d = calculate_avg(random_numbers_d)
stand_dev_d = calculate_standard_dev(random_numbers_d)

average_c = calculate_avg(random_numbers_c)
stand_dev_c = calculate_standard_dev(random_numbers_c)

#get z score for the individual random values
z_score_1 = []
z_score_2 = []
for x in random_numbers_d:
        z = (x-average_d)/stand_dev_d
        z_score_1.append(z)

for x in random_numbers_c:
        z = (x-average_c)/stand_dev_c
        z_score_2.append(z)
```

Generation of 2 sets of z-scores, one for discrete and the other for continuous data

```
targets = [stat_name]
for day in range(days):
        if cont_or_dis == 'D':
                #target is discrete, round it down
                target_raw = (z_score_1[day]*float(stat[2]))+float(stat[1])
                target = math.floor(target_raw)

                if((min_value == None) and (max_value)):
                        if (target > int(max_value)):
                                return False, False
                if((min_value) and (max_value == False)):
                        if (target < int(min_value)):
                                return False, False
                if((min_value) and (max_value)):
                        if ((target > int(max_value)) and (target < int(min_value))):
                                return False, False
```

Generation of the target values. The target value is either floored if the data is discrete or rounded of to 2 decimal points if the data is continuous. The if statements are to check whether the data produced is within maximum and minimum values that is provided in the event.txt file. This is technically an inconsistency check as the data given in the stats.txt file could be wrong, and this is a check for that. The same is done for generation of the continuous data type.

Logs

The program outputs the data in the command line interface for the user to view. It also outputs the data as a .txt file into the folder where the programme is being stored. First is the baseline_data.txt file that is generated by the activity simulation engine.

```
1 Day 1
2 Logins: 4
3 Time online: 126.69
4 Emails sent: 10
5 Emails opened: 12
6 Emails deteled: 7
7 ================================
8 Day 2
9 Logins: 4
10 Time online: 151.78
11 Emails sent: 10
12 Emails opened: 12
13 Emails deteled: 7
14 ================================
15 Day 3
16 Logins: 4
```

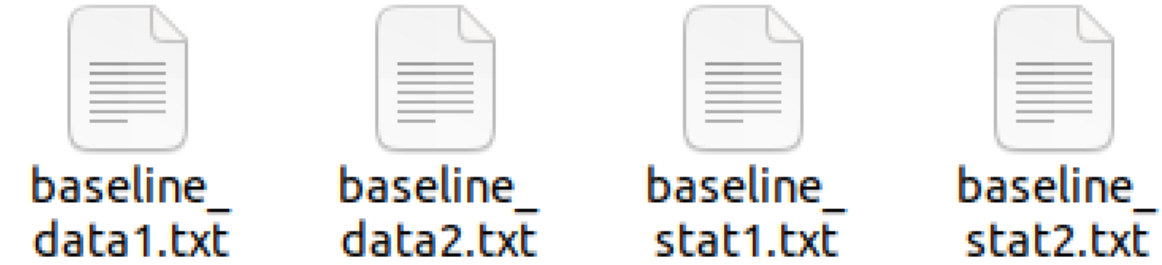
This is the data generated for each individual day. It includes the day, the events and the different target values that have been generated by the simulator engine.

Next is the baseline_stat.txt file that is generated by the analysis engine

```
1 Simulated baseline stats based on 10 days
2 ============================================
3 Event Name: Logins, [4, 4, 4, 1, 3, 5, 5, 1, 5, 3]
4 Total: 35
5 Mean: 3.5, Standard Deviation: 1.43
6 ========================================
7 Event Name: Time online, [126.69, 151.78, 189.0, 130.76, 139.98, 173.84, 114.41, 134.88,
8 Total: 1505.0100000000002
9 Mean: 150.5, Standard Deviation: 25.0
10 ========================================
11 Event Name: Emails sent, [10, 10, 11, 5, 9, 12, 13, 4, 13, 8]
```

This is a summarisation of the data that has been generated by the activity simulation engine. It shows the event name, the data generated, the sum of that total, the mean, and the standard deviation for the event.

Each file that is generated is unique and paired.



Baseline_data1.txt is paired with baseline_stat1.txt and so on. The first file will be the one generated by the simulation engine, every subsequent file that is generated will be through the alert engine and the numbers will follow suit. This ensures that data in the logs are not lost or overridden by accident and provides the user with a clear and concise way to view the data generated.

Analysis engine was just to analyse the baseline_data and output baseline_stat.

Alert Engine

The alert engine takes in a new statistical input and time period so that a new set of simulated data can be generated. This still stays within the parameters of the event.txt file assigned at the beginning of the program.

Since I just needed to extract data from the statistics file I didn't call my input engine function to do this for me.

Example of no anomalies found

```
Baseline simulated data and statistic generation completed
Open baseline_stat.txt and baseline_data.txt to view

No anomalies found for the 10 days simulation

Please enter a new stats.txt file followed by the number of days
Or enter 'exit' to quit simulation
```

Example of anomalies found