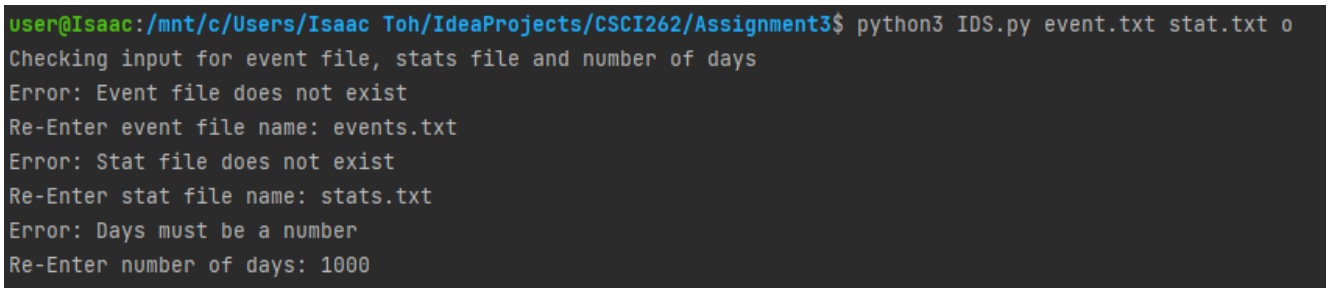
Initial Input

As per the requirements of the initial input, user will only need to input the above command in order to run the program.



Program will check for the following things:

Ensuring that both the event file and the stat file exists within the system directory where the program exists in. If the entered event or stat file name does not exist, then the program will prompt the user to enter another filename. The user will also have to enter a valid number that is larger than 0 without any included ascii characters.

```
1 row of event file: Logins: D : 0 : 100 : 3
2 row of event file: Time Online: C : 0 : 1440 : 2
3 row of event file: Emails Sent: D : 0 : 100 : 1
4 row of event file: Emails Opened: D : 0 : 100 : 1
5 row of event file: Emails Deleted: D : 0 : 100 : 1


1 row of stats file: Logins: 4 : 1.5
2 row of stats file: Time Online: 150.5 : 25.00
3 row of stats file: Emails Sent: 19 : 3
4 row of stats file: Emails Opened: 12 : 4.5
5 row of stats file: Emails Deleted: 7 : 2.25
Checking Done
```

Program will read and display all information from the event and stats file.

```
2 usages  Isaac
def checkNumEvents(events, stats):
    #Check if number of events in event file and stats file are same
    if len(events) != len(stats):
        print("Error: Number of events in event file and stats file are not same")
        exit()

2 usages  Isaac
def checkEventNames(events, stats):
    #Check if event names in event file and stats file are same
    for i in range(len(events)):
        eventName = events[i].split(':')[0].lower()
        statName = stats[i].split(':')[0].lower()
        if eventName != statName:
            print("Error: Event names in event file and stats file are not same")
            exit()
    print("Checking Done")
```

Checks done will be done to ensure that there are the same number of events in both file and if either file does not have any data in the columns needed, the program will prompt the user to enter a value into it.

```
with open(event_file, "r") as event_file:
    allevents = event_file.readlines()

for i in range(len(allevents)):
    eventNames.append(allevents[i].split(':')[0])
    eventTypes.append(allevents[i].split(':')[1])
    eventMinimum.append(allevents[i].split(':')[2])
    eventMaximum.append(allevents[i].split(':')[3])
    eventWeights.append(allevents[i].split(':')[4])
    if eventNames[i] == "":
        print("Error: Event name is empty")
        eventNames[i] = input(f"Enter event name for {i} row: ")
    if eventTypes[i] == "":
        print("Error: Event type is empty")
        eventTypes[i] = input(f"Enter event type for {eventNames[i]} row: ")
    if eventMinimum[i] == "":
        print("Error: Event minimum is empty")
        eventMinimum[i] = input(f"Enter event minimum for {eventNames[i]} row: ")
    if eventMaximum[i] == "":
        print("Error: Event maximum is empty")
        eventMaximum[i] = input(f"Enter event maximum for {eventNames[i]} row: ")
    if eventWeights[i] == "":
        print("Error: Event weight is empty")
        eventWeights[i] = input(f"Enter event weight for {eventNames[i]} row: ")
    print(f"{i + 1} row of event file: {eventNames[i]}: {eventTypes[i]} : {eventMinimum[i]} : {eventMaximum[i]} : {eventWeights[i]} ")

return eventNames, eventTypes, eventMinimum, eventMaximum, eventWeights
```

```
with open(stats_file, "r") as stats_file:
    allstats = stats_file.readlines()

print("\n")
for j in range(len(allstats)):
    statNames.append(allstats[j].split(':')[0])
    statMeans.append(allstats[j].split(':')[1])
    statSDs.append(allstats[j].split(':')[2])
    if statNames[j] == "":
        print("Error: Stat name is empty")
        statNames[j] = input(f"Enter stat name for {j} row: ")
    if statMeans[j] == "":
        print("Error: Stat mean is empty")
        statMeans[j] = input(f"Enter stat mean for {statNames[j]} row: ")[i]
    if statSDs[j] == "":
        print("Error: Stat SD is empty")
        statSDs[j] = input(f"Enter stat SD for {statNames[j]} row: ")
    print(f"{j + 1} row of stats file: {statNames[j]}: {statMeans[j]} : {statSDs[j]} ")

return statNames, statMeans, statSDs
```

Activity Simulation Engine and the Logs

Activity Simulation

Program will be used to generate data as close as possible to the mean and standard deviation specified in the stats file.

```
def generateEvents(eventName, eventType, eventMin, eventMax, statMean, statSD, days):
    event = [[eventName[j]] + [0 for i in range(days)] for j in range(len(eventName))]
    contZScore = generateZScore(3000, days)
    disZScore = generateZScore(1000, days)
    for i in range(len(eventName)):
        for j in range(1, days + 1):
            if eventType[i] == "C":
                #Function to generate continuous data
                event[i][j] = round(generateBaselineData(contZScore[j - 1], statMean[i],
                 statSD[i], eventMin[i], eventMax[i]), 2)
            else:
                #Function to generate discrete data
                event[i][j] = int(generateBaselineData(disZScore[j - 1], statMean[i], statSD[i],
                 eventMin[i], eventMax[i]))
    return event
```

```
def generateBaselineData(zScore, statMean, statSD, eventMin, eventMax):
    while True:
        event = (zScore * float(statSD)) + float(statMean)
        if float(eventMin) < event < float(eventMax):
            return event
```

In order to ensure random generation of data, for each day of data that has to be generated, a random integer of between 0-1000 for discrete data and 0-3000 for continuous data will be generated. This data will be used in the generation of a Z-score that will then be used to be multiplied against the inputted stat file's mean, standard deviation and be kept to between the minimum and maximum extracted from the event file.

The generateBaselineData function is then used to reverse the Z-score function to return a value that is adjusted by mean and standard deviation from the stat file.

Log File

For the logs file generated, it will look like thus below where there is a column for event name followed by day number then the events. Each event is listed with the event name, event type and value

```
Event Name, Day 1
Logins:D:3
Time Online:C:152.76
Emails Sent:D:17
Emails Opened:D:9
Emails Deleted:D:5


Event Name, Day 2
Logins:D:5
Time Online:C:178.32
Emails Sent:D:21
Emails Opened:D:15
Emails Deleted:D:8
```

Analysis Engine

Although the activity and logs engine writes out to the log file, the analysis engine will still read and take its data from the internal memory which is stored in a 2-dimensional list.
This is to prevent any possible formatting issues that may arise from rereading from a file.

The function will enter the obtain the mean and standard deviation of the generated events and place them into an array. The function will the carry on to find the individual Z scores for each day in each event and then get the Mean Z score for each event. After which, the mean, standard deviation and Mean Z-score will be returned to the IDS file. The Mean-Z score is calculated for the purposes of setting the baseline later on.

```python
def analyseData(events, days):
    #From events list, calculate the mean and SD for each even
    #For each event, calculate the Z score for each day
    #For each event, calculate the mean of Z scores
    mean = []
    sd = []

    print("----------------------------------------")
    print(f"Calculating mean and SD for each event")
    mean, sd = getMeanSD(events, mean, sd)

    print("----------------------------------------")
    print(f"Calculating Z score for each day in each event")
    zScore = getZScoreDayEvent(events, days, mean, sd)
    print("----------------------------------------")

    print(f"Calculating mean Z score for each event")
    print("----------------------------------------")
    meanZScore = getMeanZScore(zScore, events)
    print("----------------------------------------")

    return mean, sd, meanZScore
```

```python
def getMeanSD(events, mean, sd):
    for i in range(len(events)):
        if i == 1:
            #Round to 2 decimal places
            print(f"Hit {events[i][0]}")
            mean.append(np.round(np.mean(events[i][1:]), 2))
            sd.append(np.round(np.std(events[i][1:]), 2))
        else:
            #Round to integer
            mean.append(np.round(np.mean(events[i][1:]), 0))
            sd.append(np.round(np.std(events[i][1:]), 0))
    return mean, sd


3 usages  Isaac
def getZScoreDayEvent(events, days, mean, sd):
    zScore = []
    for i in range(len(events)):
        zScoreDay = []
        for j in range(1, days + 1):
            #New list that will be appended to zScore later
            zScoreDay.append(abs((events[i][j] - mean[i]) / sd[i]))
        zScore.append(zScoreDay)
    return zScore


1 usage  Isaac
def getMeanZScore(zScore, events):
    meanZScore = []
    for i in range(len(events)):
        meanZScore.append(np.mean(zScore[i]))
        print(f"Mean Z score for {events[i][0]}: {meanZScore[i]}")
    return meanZScore
```

After all this is done, the mean and standard deviation is then written to a file thresholdStats.txt

```
def logNewData(eventName, newmean, newsd, hasTakenInput):
    print("\n----------------------------------------")
    print("Writing to file")
    if hasTakenInput == 0:
        fileName = "thresholdStats.txt"
    else:
        fileName = f"liveStats{hasTakenInput}.txt"
    with open(fileName, 'w') as file:
        file.write("Event Name, Mean, SD\n")
        for i in range(len(eventName)):
            file.write(f"{eventName[i]}, {newmean[i]}, {newsd[i]}\n")
    print("Done writing to file")
    print("----------------------------------------")
```

Alert Engine

```
while True:
    print("\n----------------------------------------")
    print("Enter new stat file and days")
    stats_file, days, hasTakenInput = takeInput(hasTakenInput)

    newStatName, newStatMean, newStatSD = readStatsFiles(stats_file)

    newEvents = generateEvents(eventName, eventType, eventMin, eventMax, newStatMean,
     newStatSD, days)

    logEvents(newEvents, eventType, days, hasTakenInput)

    liveMean, liveSD = getMeanSD(newEvents, [], [])

    logNewData(eventName, liveMean, liveSD, hasTakenInput)
    hasTakenInput += 1

    dayZScore = getZScoreDayEvent(newEvents, days, threshHoldMean, threshHoldSD)

    print("\n----------------------------------------")
    print(f"Threshold: {sumWeight}")
    print("----------------------------------------")
    print("Alerts")
    flagAlerts(dayZScore, sumWeight, eventName, days)
    print("----------------------------------------")
    print("Done")
    while True:
        print("Do you want to continue? (y/n)")
        choice = input()
        if choice == "y":
            break
        elif choice == "n":
```

The program asks for an input of a new stat file that will be used to generate live stats that will be used to create new events to be compared against the threshold events and statistics. The image above depicts the behavior of the program after asking for a new stat file and it behaves very similarly with the baseline Stats portion of the program.

Before the flagAlerts method is called in the alert engine, the getZScoreDayEvent method from above is called again to analyse the new statistics that have been input by the user.

The way the program sets the threshold is by multiplying the weight of the event by the mean Z score attained from the threshold data and then summing it up.

```
def flagAlerts(dayZScore, thresHold, eventName, days):
    sumofDay = 0
    counter = 0
    for i in range(days):
        for j in range(len(eventName)):
            sumofDay = sumofDay + dayZScore[j][i]
        if sumofDay > thresHold:
            print(f"Day {i + 1} is flagged")
            counter += 1
        sumofDay = 0
    print(f"Total number of days flagged: {counter}")

2 usages  Isaac
def getthreshHold(meanZScore, weight):
    totalweight = 0.00
    for i in range(len(meanZScore)):
        totalweight = totalweight + (meanZScore[i] * np.float64(weight[i]))
    return 2 * totalweight
```

The program will run each days sum of Z-score against the threshold and if day's Z-score is higher than the threshold then it will flag it up.

```
---------------------------------------
Threshold: 12.752470421333333
---------------------------------------
Alerts
Day 126 is flagged: 13.252400000000002 > 12.752470421333333
Day 289 is flagged: 13.8568 > 12.752470421333333
Day 368 is flagged: 13.122799999999998 > 12.752470421333333
Day 421 is flagged: 12.879999999999999 > 12.752470421333333
Day 547 is flagged: 13.352799999999998 > 12.752470421333333
Day 608 is flagged: 12.7944 > 12.752470421333333
Day 658 is flagged: 13.369599999999998 > 12.752470421333333
```

Non-flags are not displayed.