METRISTICS

SOEN 6611: SOFTWARE MEASUREMENT
(GROUP C)

Anirudh Boddu	40225271
Ayushi Chaudhary	40224978
Riddhi Bhuva	40220969
Visnunathan Chidambaranathan	40230157
Yang Cao	26654029

November 23, 2023

Contents

List of Figures

Introduction

The purpose of descriptive statistics is to quantitatively describe a collection of data by measures of central tendency, measures of frequency, and measures of variability.

Let x be a random variable that can take values from a finite data set x1, x2, x3, ..., xn, with each value having the same probability.

- The minimum, m, is the smallest of the values in the given data set. (m need not be unique.)
- The maximum, M, is the largest of the values in the given data set. (M need not be unique.)
- The mode, o, is the value that appears most frequently in the given data set. (o need not be unique.)
- The median, d, is the middle number if n is odd, and is the arithmetic mean of the two middle numbers if n is even.
- The arithmetic mean, μ, is given by

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

- The mean absolute deviation, MAD, is given by

$$MAD = \frac{1}{n}\sum_{i=1}^{n} |x_i - \mu_i|$$

- The standard deviation, σ, is given by

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i - \mu^2}$$

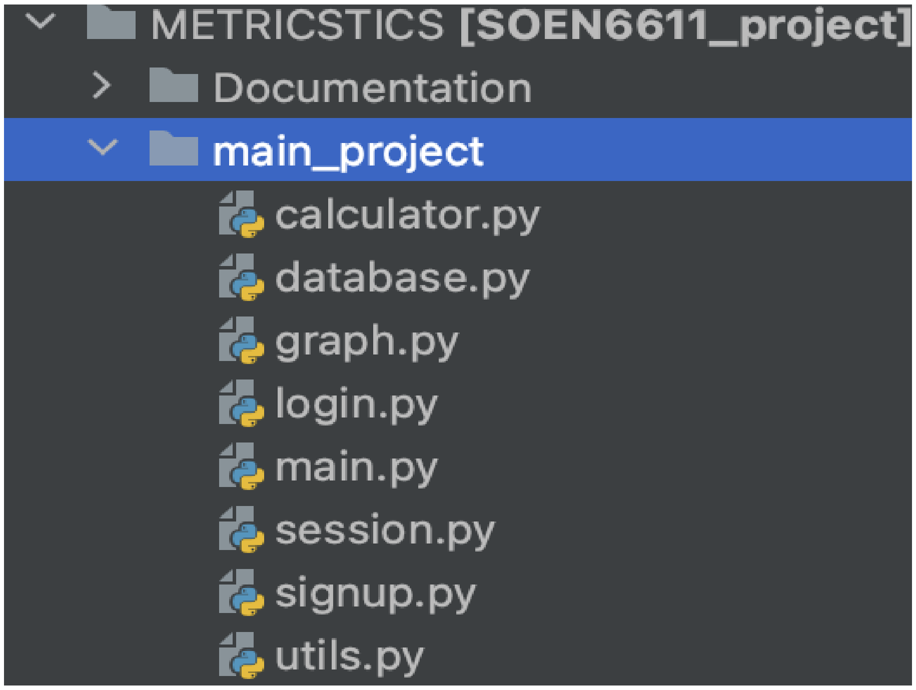
Project URL:
https://github.com/itshisher/METRICSTICS
https://www.overleaf.com/read/gxkjzdqfbndm

Chapter 1

The use of system

System documentation

Our system used object-orientation as the programming paradigm and Python as the programming language. The implementation of graphical user interface for METRICSTICS is using Tkinter. The main project includes eight python files, several csv files which are dataset generated by the system and results calculated by the system, also a log file which logs all the information while the system is running. The system is initialized by running the main.py.



```
METRICSTICS [SOEN6611_project]
  Documentation
  main_project
    calculator.py
    database.py
    graph.py
    login.py
    main.py
    session.py
    signup.py
    utils.py
```

Figure 1: Python classes

After we run main.py, a login page pops out asking the user either to login, signup, or simply bypass the login page to use the system as a guest.

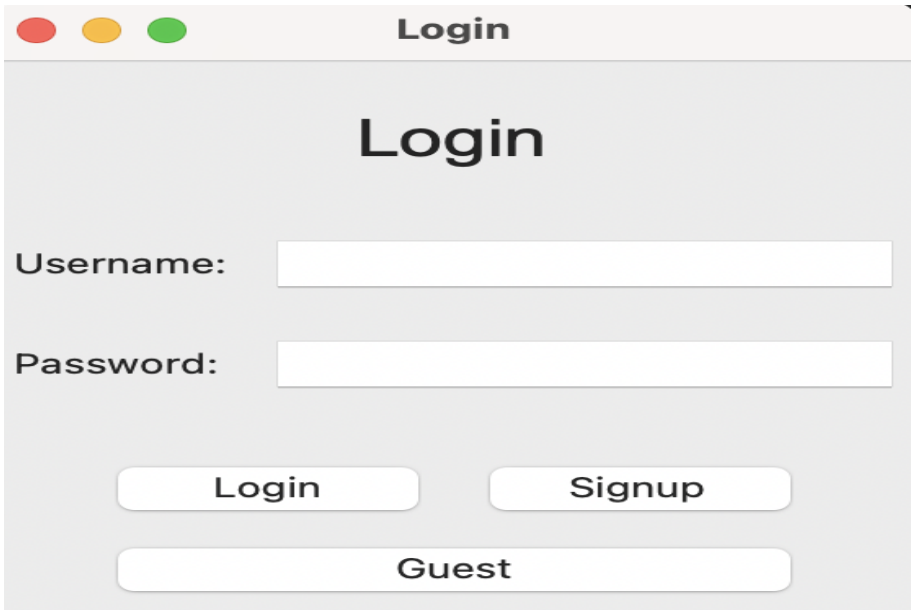
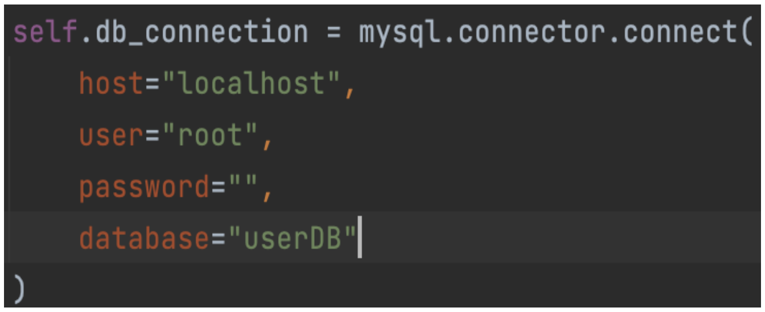


Figure 2: Login page

Right now, the system is connected to the local database with the settings shown below.

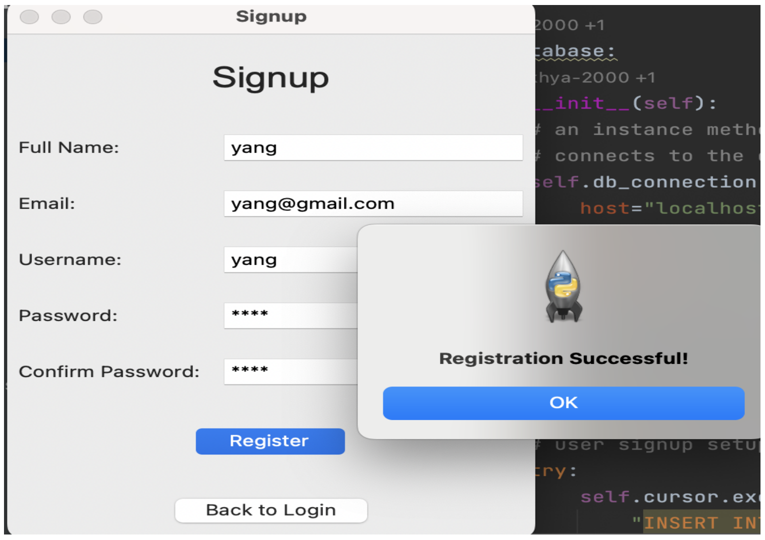


```
self.db_connection = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database="userDB"
)
```

Figure 3: Database connection

1.1 Sign up to use the system

Users need to click on the button **Signup** to create an new account. The system directs users to a new page to input user information and custmized password.



Signup

Full Name: yang

Email: yang@gmail.com

Username: yang

Password: ****

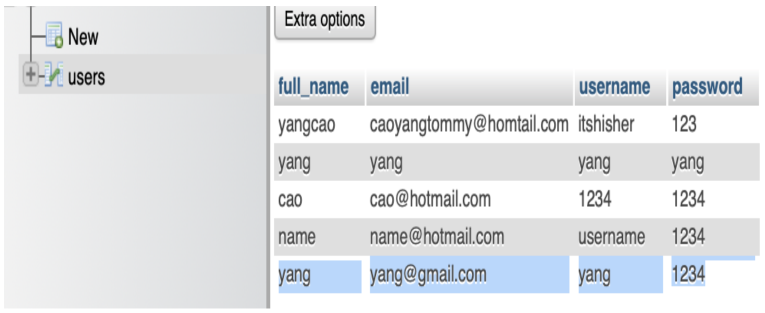
Confirm Password: ****

Registration Successful!

```
2000 +1
abase:
hya-2000 +1
__init__(self):
# an instance meth
# connects to the
self.db_connection
    host="localhos
# user signup setu
ry:
    self.cursor.ex
        "INSERT IN
```

Figure 4: User signup

Once the user input information is correct and password is verified, user information will be stored into the database.



full_name	email	username	password
yangcao	caoyangtommy@homtail.com	itshisher	123
yang	yang	yang	yang
cao	cao@hotmail.com	1234	1234
name	name@hotmail.com	username	1234
yang	yang@gmail.com	yang	1234

Figure 5: Database

Now this user can login to use the system.

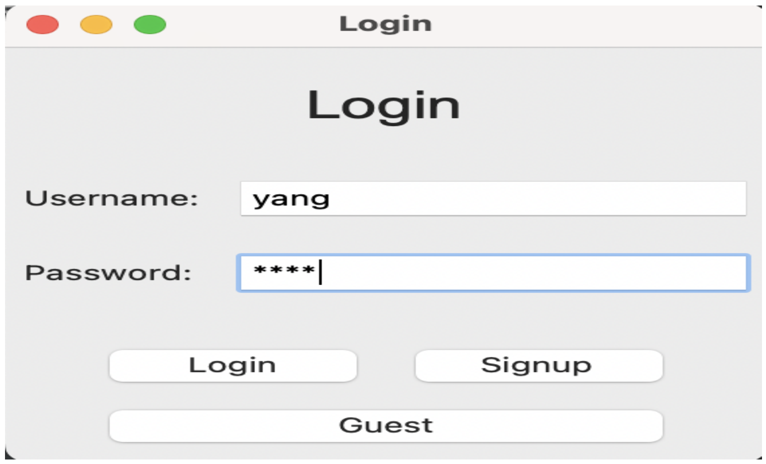


Figure 6: User login

After user login, the next actions are the same as in the step 1.2.

1.2 Login to use the system

If login successfully, the user will be directed to a new page with a view of the METRICSTRICS system. All functionalities of the system are included here and users with different roles can perform their inquiries.

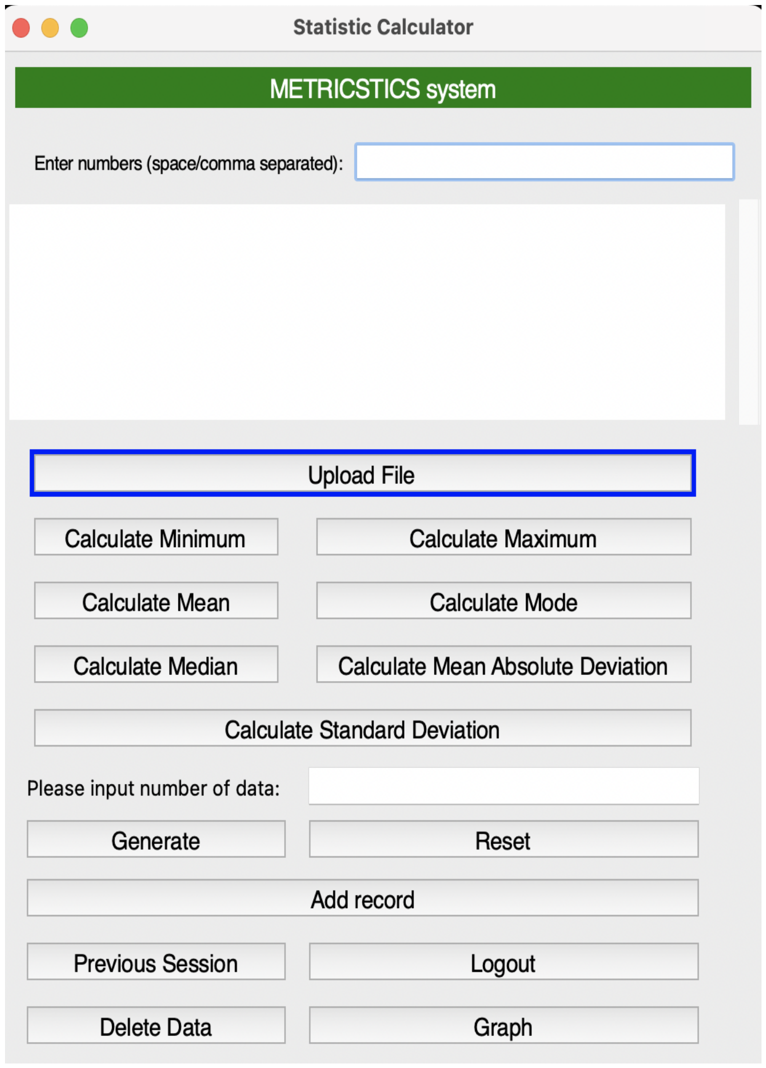
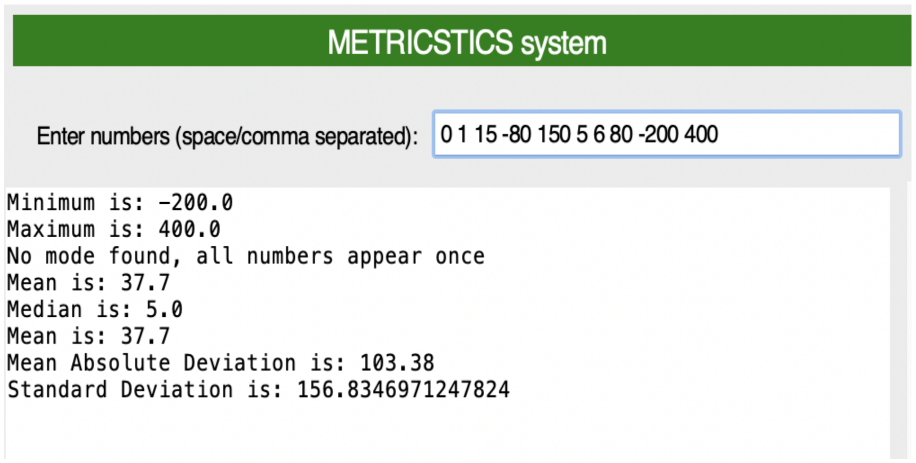


Figure 7: Main system

Following are the functionalities of our system.

1.2.1 User input data

Out system can perform seven descriptive statistics based on user input data of a wide range.



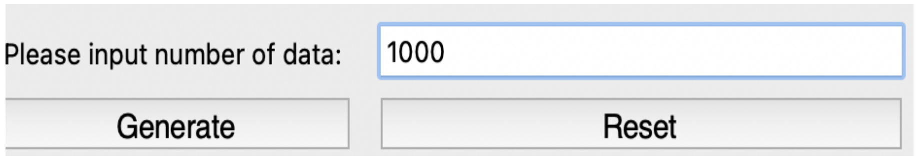
Enter numbers (space/comma separated): 0 1 15 -80 150 5 6 80 -200 400

```
Minimum is: -200.0
Maximum is: 400.0
No mode found, all numbers appear once
Mean is: 37.7
Median is: 5.0
Mean is: 37.7
Mean Absolute Deviation is: 103.38
Standard Deviation is: 156.8346971247824
```

Figure 8: Test of user input data

1.2.2 User generate dataset

Users can **generate** the dataset with their desired amount of data. Then they can upload it to the system to perform seven descriptive statistics. Here, our system default range for generating dataset is from -1000000 to 1000000.



Please input number of data: 1000

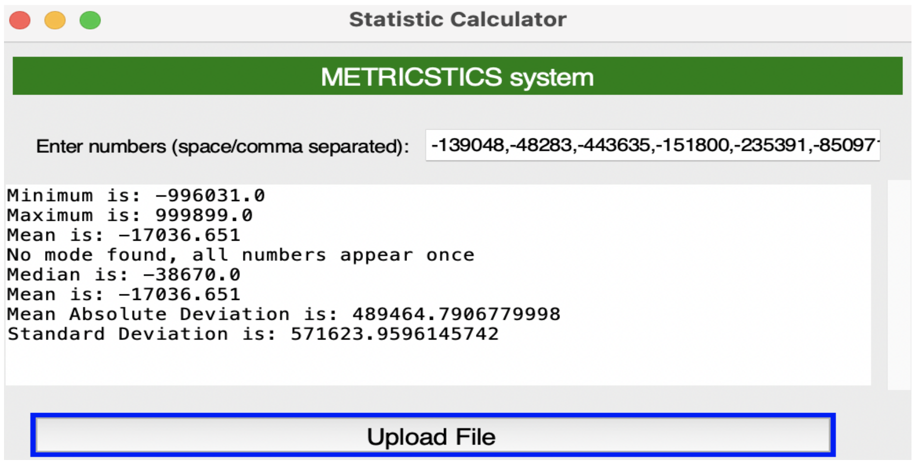
Figure 9: Dataset generated by the user

Logged in users will have a csv file named with Data_ followed by their usernames showing that different users will have different files generated by the system.

Figure 10: Generated file with specific username

1.2.3 Upload file

After a user generating a test dataset. He/she can choose to upload it to our system to perform seven descriptive statistics.



```
Minimum is: -996031.0
Maximum is: 999899.0
Mean is: -17036.651
No mode found, all numbers appear once
Median is: -38670.0
Mean is: -17036.651
Mean Absolute Deviation is: 489464.7906779998
Standard Deviation is: 571623.9596145742
```

Figure 11: Test uploaded dataset

1.2.4 Save session

Users can click on the **Add record** button to save the descriptive statistics to the local machine and view it in a csv file.

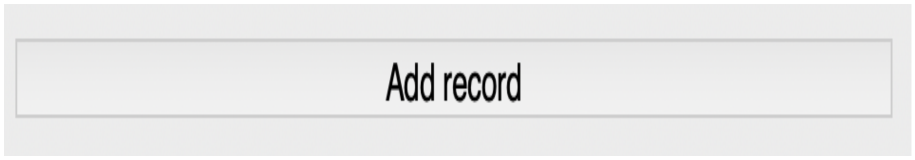
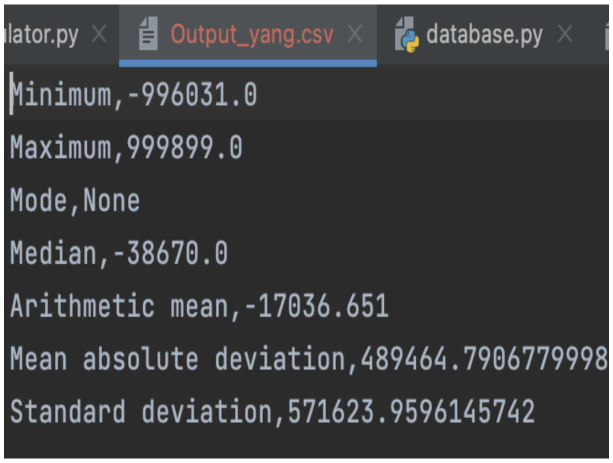


Figure 12: Add record button

Logged in users will have csv file name with Output_ followed by their usernames showing that different users will have different files saved by the system.

Figure 13: Test uploaded dataset



```
Minimum,-996031.0
Maximum,999899.0
Mode,None
Median,-38670.0
Arithmetic mean,-17036.651
Mean absolute deviation,489464.7906779998
Standard deviation,571623.9596145742
```

Figure 14: Saved results

1.2.5 Showing graph

After a user performs seven descriptive statistics and clicks on the Add record button to save the results, he/she can click on the **Graph** button to view results of descriptive statistics graphically. Then click on Back to Calculator to perform more calculations.

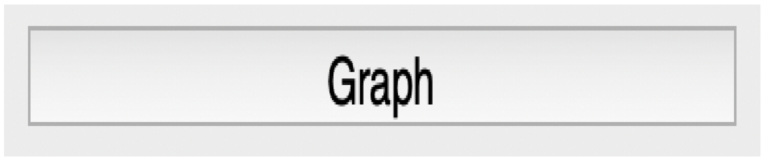


Figure 15: Graph button

Here, in each graph there is one data point since it's this user's first test. More tests are adding later.

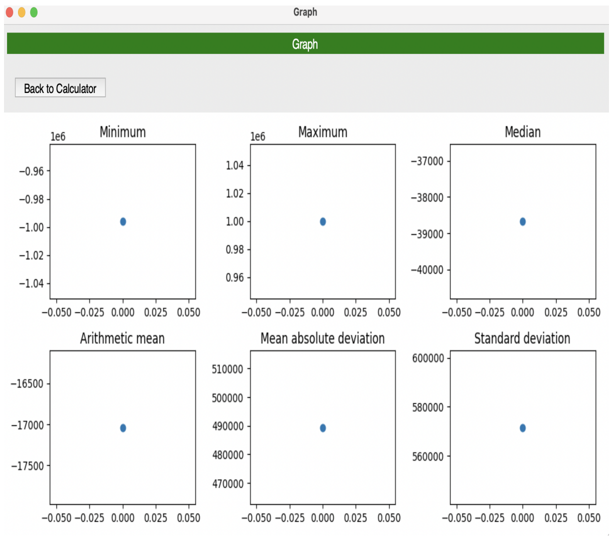


Figure 16: Graph for one test case

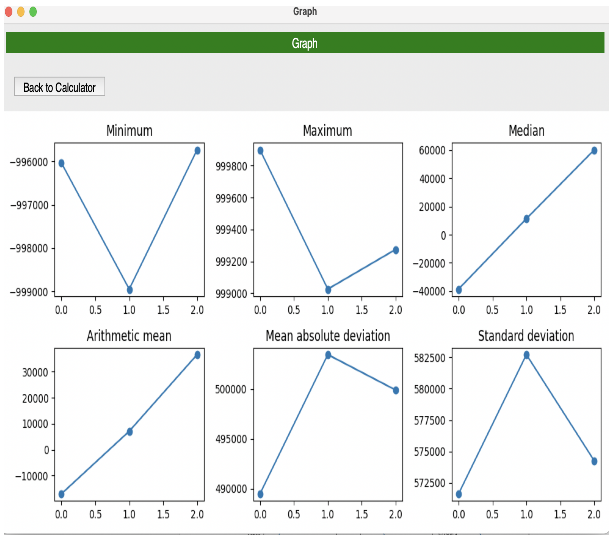


Figure 17: Graph for three test cases

1.2.6 Previous session

Pervious session function is available for users to view the calculations that they performed before by simply read previously saved descriptive statistics from the file generated in section 1.2.4.

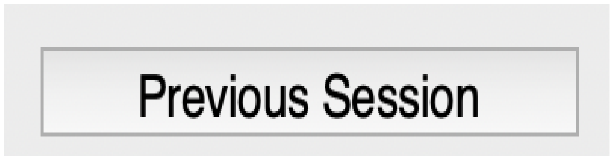
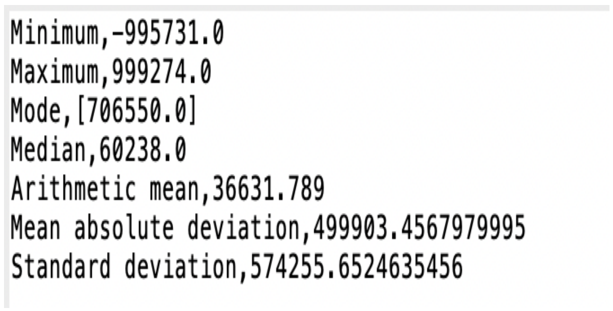


Figure 18: Previous session



```
Minimum,-995731.0
Maximum,999274.0
Mode,[706550.0]
Median,60238.0
Arithmetic mean,36631.789
Mean absolute deviation,499903.4567979995
Standard deviation,574255.6524635456
```

Figure 19: Results showing of previous session

1.2.7 Reset

Reset button to clear the user input and results that are showing in the screen.

Figure 20: Reset screen

1.2.8 Delete data

Delete data button to delete the data in local saved file that belongs to this particular user.

Figure 21: Delete data

1.2.9 User logout

Logout button for users to logout the system.

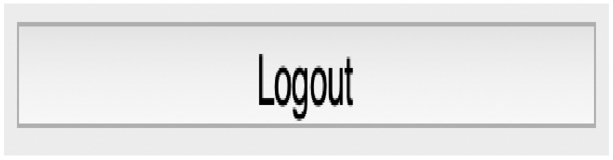
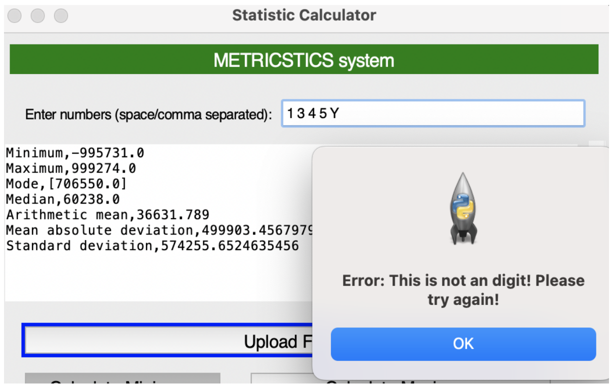


Figure 22: User logout

1.2.10 Error handling

If users input a non-digital character, an error message will prompt out.



Enter numbers (space/comma separated): 1 3 4 5 Y

```
Minimum,-995731.0
Maximum,999274.0
Mode,[706550.0]
Median,60238.0
Arithmetic mean,36631.789
Mean absolute deviation,499903.456797
Standard deviation,574255.6524635456
```

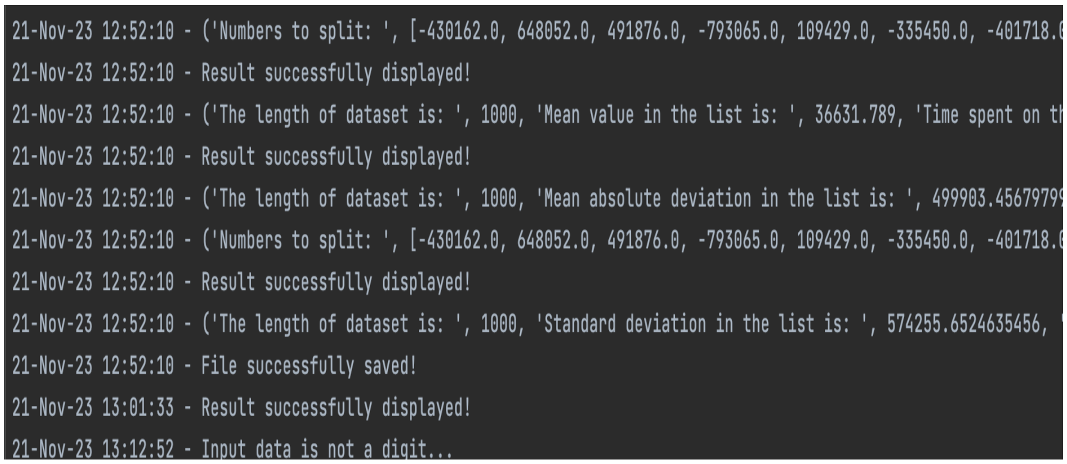
Error: This is not an digit! Please try again!

Figure 23: Error handling

1.2.11 Logging

A log file is also included to record actions performed by the user.

Figure 24: Log file

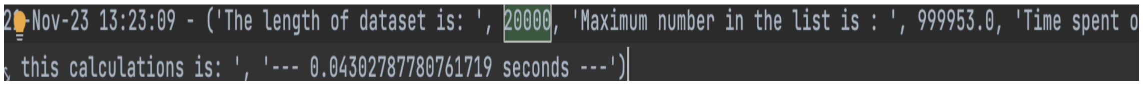


```
21-Nov-23 12:52:10 - ('Numbers to split: ', [-430162.0, 648052.0, 491876.0, -793065.0, 109429.0, -335450.0, -401718.
21-Nov-23 12:52:10 - Result successfully displayed!
21-Nov-23 12:52:10 - ('The length of dataset is: ', 1000, 'Mean value in the list is: ', 36631.789, 'Time spent on t
21-Nov-23 12:52:10 - Result successfully displayed!
21-Nov-23 12:52:10 - ('The length of dataset is: ', 1000, 'Mean absolute deviation in the list is: ', 499903.4567979
21-Nov-23 12:52:10 - ('Numbers to split: ', [-430162.0, 648052.0, 491876.0, -793065.0, 109429.0, -335450.0, -401718.
21-Nov-23 12:52:10 - Result successfully displayed!
21-Nov-23 12:52:10 - ('The length of dataset is: ', 1000, 'Standard deviation in the list is: ', 574255.6524635456, '
21-Nov-23 12:52:10 - File successfully saved!
21-Nov-23 13:01:33 - Result successfully displayed!
21-Nov-23 13:12:52 - Input data is not a digit...
```

Figure 25: Records of actions in the log file

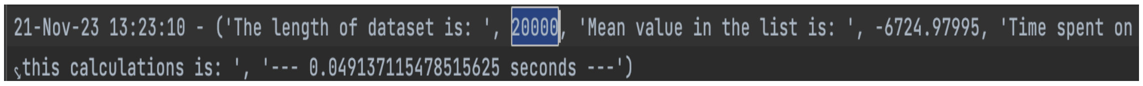
1.2.12 System performance

Our descriptive statistics do not make use of any reuse mechanism (such as built-in functions, libraries, or APIs) and we also implemented divide and conquer algorithms to make the calculation run smoothly. Users can generate the dataset with large amout of entries and the execution time is very fast and acceptable. Details can be viewed in scraper.log file. Several tests on execution tome of 20000 data is show below.



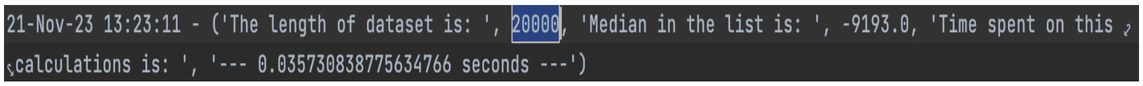
```
2 -Nov-23 13:23:09 - ('The length of dataset is: ', 20000, 'Maximum number in the list is : ', 999953.0, 'Time spent o
 this calculations is: ', '--- 0.04302787780761719 seconds ---')
```

Figure 26: Performance test 1



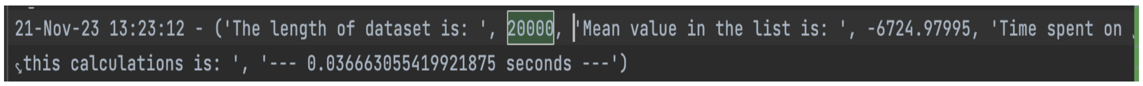
```
21-Nov-23 13:23:10 - ('The length of dataset is: ', 20000, 'Mean value in the list is: ', -6724.97995, 'Time spent on
this calculations is: ', '--- 0.049137115478515625 seconds ---')
```

Figure 27: Performance test 2



```
21-Nov-23 13:23:11 - ('The length of dataset is: ', 20000, 'Median in the list is: ', -9193.0, 'Time spent on this
calculations is: ', '--- 0.035730838775634766 seconds ---')
```

Figure 28: Performance test 3



```
21-Nov-23 13:23:12 - ('The length of dataset is: ', 20000, 'Mean value in the list is: ', -6724.97995, 'Time spent on
this calculations is: ', '--- 0.036663055419921875 seconds ---')
```

Figure 29: Performance test 4

1.3 Use the system as a guest

If users click on the **Guest** button, they can use the system directly and most of the functions in the system can be used. Except for some changes.

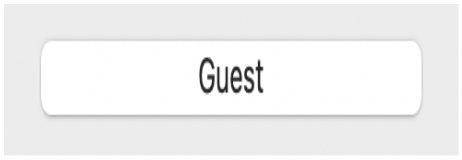


Figure 30: Guest

Guests will share one common Output_Guest file which is the descriptive statistics performed by the system and saved to local.

Figure 31: Common saved file for guests

Guests will share one common Data_Guest file which is the desired amount of data by user input.

Figure 32: Common output file for guests

Chapter 2

Descriptions of algorithms

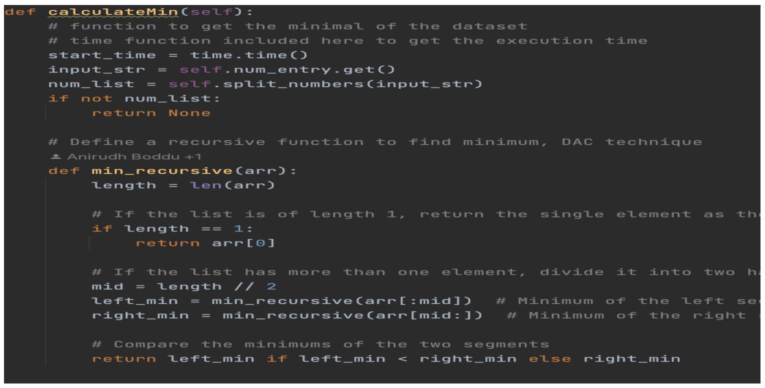
Below is the description of algorithms for seven descriptive statistics.

2.1 The minimum

This method is to find the minimum value within a list using a recursive algorithm. The process of comparison and selection continues until it reaches the base case. After, it returns the overall minimum value in the entire list.

This method calculates the minimum value for a given list (num_list) and displays it in a logging file like the length of the dataset, the minimum number found, and the time taken to perform the calculations.

This method used a divide-and-conquer algorithm, recursively dividing the list into smaller segments until it reaches the smallest units and then merging and comparing the results to find the overall minimum.



```
def calculateMin(self):
    # function to get the minimal of the dataset
    # time function included here to get the execution time
    start_time = time.time()
    input_str = self.num_entry.get()
    num_list = self.split_numbers(input_str)
    if not num_list:
        return None

    # Define a recursive function to find minimum, DAC technique
    Anirudh Boddu +1
    def min_recursive(arr):
        length = len(arr)

        # If the list is of length 1, return the single element as th
        if length == 1:
            return arr[0]

        # If the list has more than one element, divide it into two h
        mid = length // 2
        left_min = min_recursive(arr[:mid])  # Minimum of the left se
        right_min = min_recursive(arr[mid:])  # Minimum of the right

        # Compare the minimums of the two segments
        return left_min if left_min < right_min else right_min
```

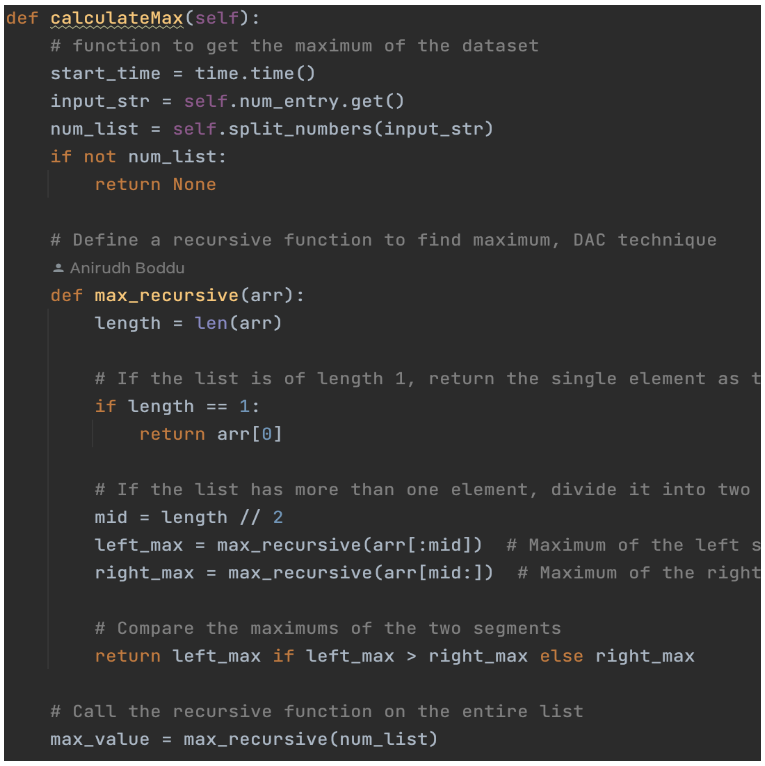
Figure 1: Calculate the minimum

2.2 The maximum

This method finds the maximum value within a provided dataset using a recursive algorithm. The process of comparison and selection continues until it reaches the base case. After, it returns the overall maximum value in the entire list.

The method displays the maximum value and logs information such as the length of the dataset, the maximum number found, and the time taken to perform the calculations.

This method used a divide-and-conquer algorithm, recursively dividing the list into smaller segments until it reaches the largest units and then merging and comparing the results to find the overall maximum.



```
def calculateMax(self):
    # function to get the maximum of the dataset
    start_time = time.time()
    input_str = self.num_entry.get()
    num_list = self.split_numbers(input_str)
    if not num_list:
        return None

    # Define a recursive function to find maximum, DAC technique
    Anirudh Boddu
    def max_recursive(arr):
        length = len(arr)

        # If the list is of length 1, return the single element as t
        if length == 1:
            return arr[0]

        # If the list has more than one element, divide it into two
        mid = length // 2
        left_max = max_recursive(arr[:mid])  # Maximum of the left s
        right_max = max_recursive(arr[mid:])  # Maximum of the right

        # Compare the maximums of the two segments
        return left_max if left_max > right_max else right_max

    # Call the recursive function on the entire list
    max_value = max_recursive(num_list)
```

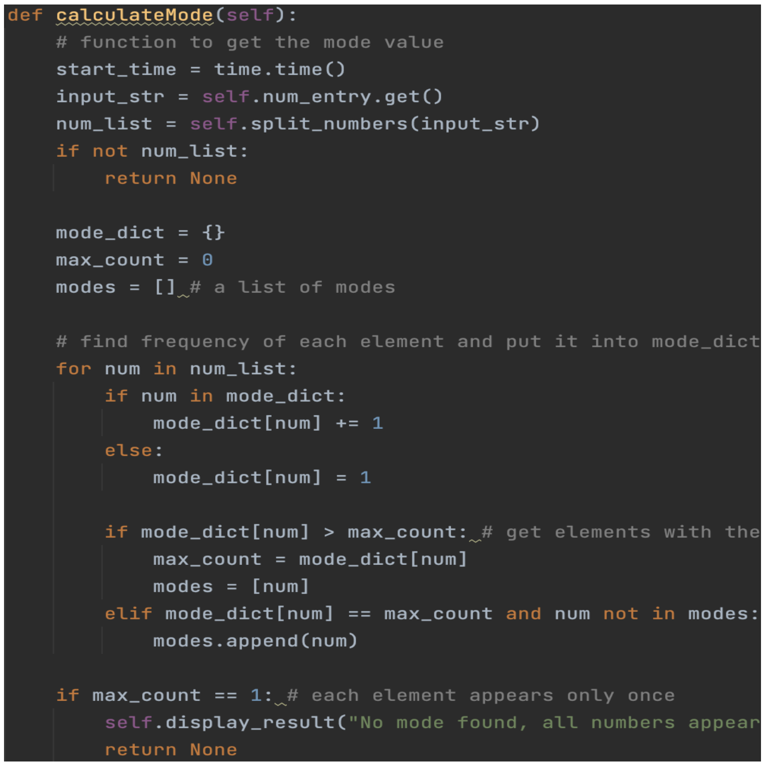
Figure 2: Calculate the minimum

2.3 The mode

This method utilizes a dictionary to count the frequency of each element in the dataset and then identifies the number(s) with the highest frequency as the mode(s).

This method keeps track of the max_count representing the maximum frequency encountered so far and updates the modes list to contain numbers that have the highest frequency (max_count). If multiple numbers have the same maximum frequency, it appends them to the modes list.

This method also includes Logs information such as the length of the dataset, the mode(s) found, and the time taken to perform the calculations.



```python
def calculateMode(self):
    # function to get the mode value
    start_time = time.time()
    input_str = self.num_entry.get()
    num_list = self.split_numbers(input_str)
    if not num_list:
        return None

    mode_dict = {}
    max_count = 0
    modes = [] # a list of modes

    # find frequency of each element and put it into mode_dict
    for num in num_list:
        if num in mode_dict:
            mode_dict[num] += 1
        else:
            mode_dict[num] = 1

        if mode_dict[num] > max_count: # get elements with the
            max_count = mode_dict[num]
            modes = [num]
        elif mode_dict[num] == max_count and num not in modes:
            modes.append(num)

    if max_count == 1: # each element appears only once
        self.display_result("No mode found, all numbers appear
        return None
```

Figure 3: Calculate the mode

2.4 The median

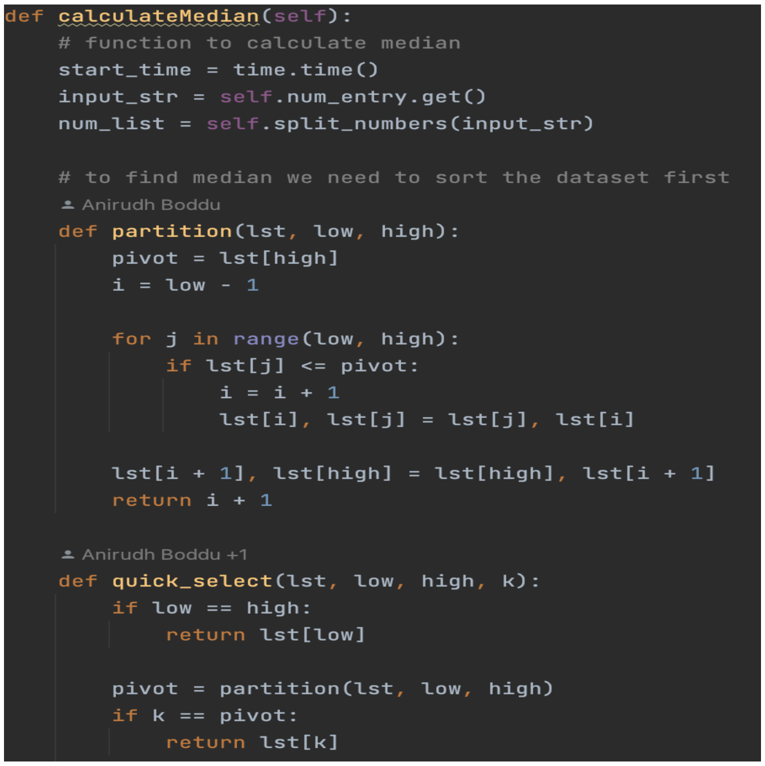
This method utilizes the quick select algorithm to efficiently find the median of the dataset without sorting the entire list, making it faster for large datasets compared to other sorting algorithms.

There are two inner methods:

- partition: Partitions the list based on a chosen pivot element.
- quick_select: Finds the kth smallest element in the list using partitioning and recursion.

After, this method finds the middle index of the dataset and uses quick_select to find the median value.

Again, the result of calculated median value and logs information are shown in the log file.



```
def calculateMedian(self):
    # function to calculate median
    start_time = time.time()
    input_str = self.num_entry.get()
    num_list = self.split_numbers(input_str)

    # to find median we need to sort the dataset first
    Anirudh Boddu
    def partition(lst, low, high):
        pivot = lst[high]
        i = low - 1

        for j in range(low, high):
            if lst[j] <= pivot:
                i = i + 1
                lst[i], lst[j] = lst[j], lst[i]

        lst[i + 1], lst[high] = lst[high], lst[i + 1]
        return i + 1

    Anirudh Boddu +1
    def quick_select(lst, low, high, k):
        if low == high:
            return lst[low]

        pivot = partition(lst, low, high)
        if k == pivot:
            return lst[k]
```

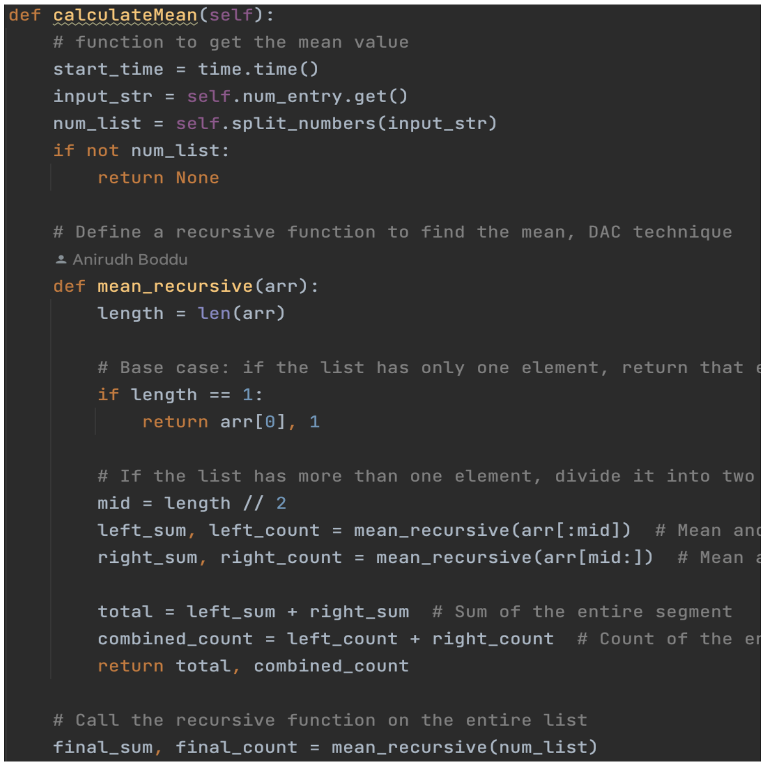
Figure 4: Calculate the median

2.5 The arithmetic mean

This method uses a recursive algorithm to compute the mean value of a dataset. The process of comparison and selection continues until it reaches the base case. After, it returns the overall mean value in the entire list.

The method used the total sum and total count obtained from the recursive calls; it calculates the final mean by dividing the total sum by the total count.

Similar to the previous methods, this function uses a divide-and-conquer technique to recursively break down the dataset and displays the calculated mean and logs information.



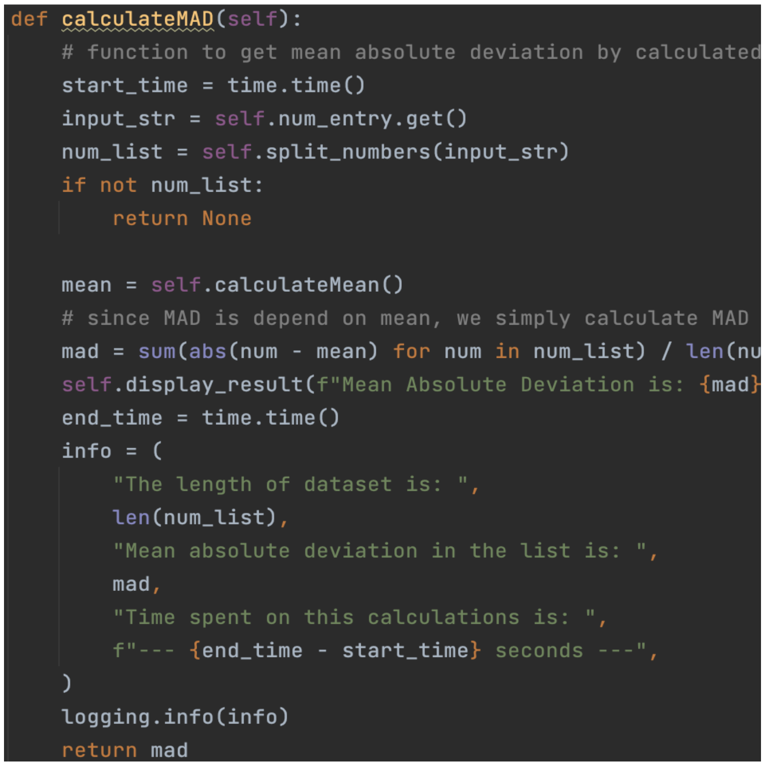
```
def calculateMean(self):
    # function to get the mean value
    start_time = time.time()
    input_str = self.num_entry.get()
    num_list = self.split_numbers(input_str)
    if not num_list:
        return None

    # Define a recursive function to find the mean, DAC technique
    Anirudh Boddu
    def mean_recursive(arr):
        length = len(arr)

        # Base case: if the list has only one element, return that e
        if length == 1:
            return arr[0], 1

        # If the list has more than one element, divide it into two
        mid = length // 2
        left_sum, left_count = mean_recursive(arr[:mid])  # Mean and
        right_sum, right_count = mean_recursive(arr[mid:])  # Mean a

        total = left_sum + right_sum  # Sum of the entire segment
        combined_count = left_count + right_count  # Count of the en
        return total, combined_count

    # Call the recursive function on the entire list
    final_sum, final_count = mean_recursive(num_list)
```

Figure 5: Calculate the arithmetic mean

2.6 The mean absolute deviation

This method uses the mean value that is calculated in calculateMean() to compute the absolute differences between each data point and the mean, providing a measure of the average distance between individual data points and the mean value.

After the calculation is done, it displays the calculated MAD and logs information such as the length of the dataset, the MAD value, and the time taken to perform the calculations.



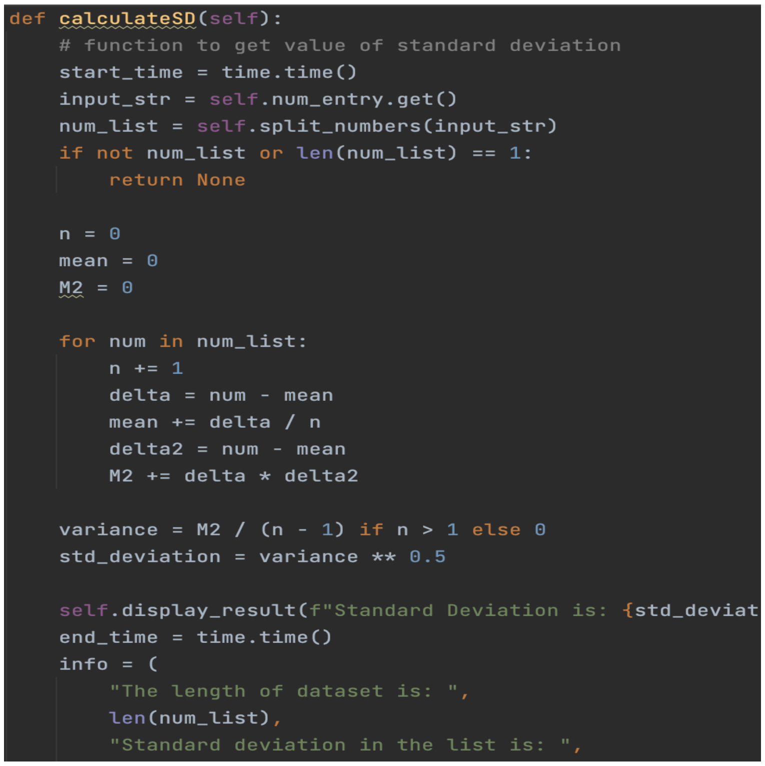
```python
def calculateMAD(self):
    # function to get mean absolute deviation by calculated
    start_time = time.time()
    input_str = self.num_entry.get()
    num_list = self.split_numbers(input_str)
    if not num_list:
        return None

    mean = self.calculateMean()
    # since MAD is depend on mean, we simply calculate MAD
    mad = sum(abs(num - mean) for num in num_list) / len(nu
    self.display_result(f"Mean Absolute Deviation is: {mad}
    end_time = time.time()
    info = (
        "The length of dataset is: ",
        len(num_list),
        "Mean absolute deviation in the list is: ",
        mad,
        "Time spent on this calculations is: ",
        f"--- {end_time - start_time} seconds ---",
    )
    logging.info(info)
    return mad
```

Figure 6: Calculate the mean absolute deviation

2.7 The standard deviation

This method computes the standard deviation of the dataset by iteratively calculating the mean and the squared differences from the mean to obtain the variance and then takes the square root of the variance to find the standard deviation.

Then it displays the calculated standard deviation and logs information.



```
def calculateSD(self):
    # function to get value of standard deviation
    start_time = time.time()
    input_str = self.num_entry.get()
    num_list = self.split_numbers(input_str)
    if not num_list or len(num_list) == 1:
        return None

    n = 0
    mean = 0
    M2 = 0

    for num in num_list:
        n += 1
        delta = num - mean
        mean += delta / n
        delta2 = num - mean
        M2 += delta * delta2

    variance = M2 / (n - 1) if n > 1 else 0
    std_deviation = variance ** 0.5

    self.display_result(f"Standard Deviation is: {std_deviat
    end_time = time.time()
    info = (
        "The length of dataset is: ",
        len(num_list),
        "Standard deviation in the list is: ",
```

Figure 7: Calculate the standard deviation