InBody-770 Data Analyser

Varkirti Gopalkrishnan

Table of Contents

[Scope of Work 1](#_Toc142564295)

[Relevant Background and Skills 1](#_Toc142564296)

[Skills/Knowledge Required 2](#_Toc142564297)

[Tools Utilised 2](#_Toc142564298)

[QuantumTX Data Analysis Tool 2](#_Toc142564299)

[Raw Data 2](#_Toc142564300)

[Version 1.0 2](#_Toc142564301)

[Version 2.0 2](#_Toc142564302)

[Functions 3](#_Toc142564303)

[Data Cleaning 3](#_Toc142564304)

[Merging of Data Files 3](#_Toc142564305)

[Creation of Individual User Sheets 3](#_Toc142564306)

[Generating Plots 4](#_Toc142564307)

[Data Cleaning 4](#_Toc142564308)

[Filtration 5](#_Toc142564309)

[Processing 5](#_Toc142564310)

[Collation 6](#_Toc142564311)

[Aggregation 7](#_Toc142564312)

[Things to Take Note Of 7](#_Toc142564313)

# Scope of Work

* Acquire user health metrics from the InBody 770 bio-impedance machine.
* Create a tool to clean, organise, and process the data.
* Create a tool to analyse the data, plot trends, and present the data in a manner that can be easily interpreted by the team, used for further analysis, and depicted in a suitable manner for marketing purposes in the future.

# Relevant Background and Skills

* Experience coding in Python, Visual Basic, Java, and QBasic.
* Experience in the use of various data structures such as lists, tuples, dictionaries, NumPy Arrays, etc.
* Experience using SQL for data storage, organisation, and analysis, along with SQL Querying to acquire said data.
* Experience using various Python libraries, packages, and tools, such as NumPy, SciPy, Matplotlib, Python-MySQL Connector, etc.

# Skills/Knowledge Required

* Extraction and manipulation of data from Microsoft Excel files (.xlsx)
* Transfer of aforementioned files and data to Python for analysis.
* Usage of libraries like OpenPyXL and Pandas to accomplish the above tasks.
* Usage and manipulation of Dataframes (from Pandas), and sets of Dataframes, to filter, clean, and plot data.
* Data Aggregation and Interpolation related skills.
* Use of PyInstaller to compile .py files, along with related dependencies, into executables for MacOS and Windows Systems.

# Tools Utilised

* Visual Studio Code (Code Editor)
* GitHub Copilot (LLM-based coding Assistant)
* Google – to find necessary Libraries and Packages
* Documentation – to understand how to use the Libraries and Packages
* Python 3.11.4

# QuantumTX Data Analysis Tool

## Raw Data

The raw data from the InBody Machine could be exported in several different file formats, but the QuantumTX team chose the ‘.xlsx’ file format (Excel File). This meant that any analysis that I could perform would first require the extraction of the data from the Excel Sheet.

Each raw file exported from the InBody 770 machine contained hundreds of scans since most users would undergo a scan every time they used the BIXEPS machine. Each scan contained almost 300 metrics, making it very difficult to sort out and analyse trends without an automation tool.

## Version 1.0

The first version of the script I created used the OpenPyXL library to manipulate data directly on the Excel sheet. This first version of the tool was only capable of cleaning the data, by deleting redundant “Upper Limit” and “Lower Limit” columns, deleting duplicate scans, and organising individual user data into separate sheets in the Excel document, in case the team would like to analyse the trends of individual users in the future.

This version of the tool accomplished the data cleaning successfully but had several downsides such as being quite slow and inefficient, making the notion of manipulating large quantities of this data directly to and from the Excel file quite unappealing. The time required to even analyse the data for a single user would be too long to be feasible or useful.

## Version 2.0

The second version of the script was made with several improvements. First, instead of manipulating data directly in the selected Excel file, I transitioned to using the Pandas library to extract the data from the Excel file and store it in a data structure called a dataframe (an object in the Pandas Library). Dataframes were perfect for this use case as they are kind of dictionary-like arrays, perfect for storing tabular data efficiently while still being able to retrieve data from specific columns and rows using either indices or name references (like dictionaries).

After extracting the data from the Excel file, the script would then perform all the relevant cleaning, processing, and plotting, and simply write the processed data to a new Excel file as and when was appropriate. This made the program significantly faster while making the code simpler as the nature of dataframes allows for easy filtering and manipulation of tabular data.

### Functions

#### Data Cleaning

The program will request the name of the Excel file containing Raw Data directly from the InBody 770. It will then extract the data to a dataframe, and perform the following:

* Removal of redundant columns.
* Removal of columns with data for which trends would not be required.
* Removal of any duplicate data.

The processed data will then be saved to a new

#### Merging of Data Files

The program will ask the user for a few bits of data:

* The type of files they would like to merge:
  + Raw Files
  + User Data Sheets
* They will then be asked about the number of files they would like to merge.
* Then they will be asked to supply the name of the files they would like to merge.

After acquiring this information, the script will perform the following actions:

* Extract the data from the Excel files and store it in a dataframe (raw data files) or a dictionary of dataframes (user data sheet files) as required.
* Clean the data in every dataframe created.
* Merge the data accordingly:
  + If dealing with raw files, then merge to a single dataframe.
  + If dealing with user data sheets, then merge the data for each user from each dictionary together and store it in a new dataframe that is part of a new dictionary.
* Remove any duplicates from every dataframe.
* Write the data from the merged dataframe (or dataframe dictionary) to a new Excel file.

#### Creation of Individual User Sheets

After requesting the name of requisite raw data file, the program will then do the following:

* Clean the data in the sheet as described above.
* Create a dictionary of dataframes, with each dataframe corresponding to a single user from the raw data file, and the dictionary key associated with each dataframe being the user’s ID as listed in the raw data file.
* Each of the individual user dataframes will then be checked for any duplicate entries.
* A new excel file will be created, with a new sheet created for each user, and subsequently populated with the data for that user from the user dataframe dictionary.

#### Generating Plots

This is by far the most complex part of the program, requiring the organisation, filtration, manipulation, collation, and aggregation of data, followed by the generation of a plot (or plots) for one (or many) users, for one (or multiple metrics).

The inputs requested by the program for plot generation are as follows:

* The program will ask the user if they wish to make a plot using a single user’s data or all the users’ data.
  + If the single-user option is chosen, a user ID will be requested.
  + If the all-users option is chosen, at a later stage in the program more options will be presented.
* It will request a time increment input. This will be the time increment that the plot will use on the x-axis while plotting the data. The options presented will include days, weeks, 2-week chunks, and months.
* The next input required will be regarding the metrics to be plotted. The options presented would include a single metric or all metrics.
* At this point, if the all-users option was chosen earlier on, the program will request a selection between plotting, for a single metric, all the selected users’ data as individual lines on a single plot, or by aggregating all the user’s data into a single line before generating the plot.
* Once the relevant data processing is completed the program will generate the required plot(s) based on the selection characteristics, create a folder to store the plots in within the selected directory, and then save the plots. The program will also store the processed data that was plotted, onto a new Excel file, in case the team wishes to investigate individual points or conduct further analysis.

The raw data supplied will undergo a few procedures before being plotted:

* Data Cleaning
* User Dataframe Dictionary Creation
* Data processing pipeline:
  + Filtration
  + Processing
  + Collation
  + Aggregation (if selected)

Each of these parts of the pipeline is detailed below.

##### Data Cleaning

Beyond the prior cleaning done on the data, a few extra cleaning procedures are performed on the data to be plotted.

First, the ‘Test Date/Time’ column is split into separate ‘Date’ and ‘Time’ columns, storing the respective values in the Python DateTime format. Since the data at this point has already been scanned for duplicates, it is no longer necessary to keep the ‘Test Date/Time’ and ‘Time’ columns, so they are deleted.

Next, the columns such as ‘Gender’ and ‘VFL (Visceral Fat Level)’ are changed to make their values easier to process. The ‘Gender’ column can store only two values, so it is changed to store 0s and 1s, and the ‘VFL (Visceral Fat Level)’ column values are changed to remove the word ‘Level’ and just keep the actual number value.

Lastly, and most crucially, five new columns of data are created: ‘Year’, ’Day’, ’Week’, ’Biweekly Number’, and ’Month’. Using the ‘Date’ column, they are respectively, populated with the year, day of the year, week number, biweekly number (derived from week number), and month number for each entry. These columns are essential to further data processing.

##### Filtration

If the single-user option is chosen, then the dataframes associated with all other users are deleted from the dictionary. This is an example of the removal of unnecessary data to reduce processing overhead and increase efficiency.

Next, the user is asked if they would like to filter by a certain metric. If they choose to filter by a metric, they are allowed to choose a specific value to filter by, or a range of values to filter by.

Additionally, they are presented with the option to filter by start-point or all data. The start-point factor filters the data by a certain metric, but only checks the starting values for each user for that metric, allowing for the categorisation of users for analysis. For example, if filtering by start-point BMI, we can select users who started at a specific BMI (or BMI range) and see trends that may be uniquely associated with them. Conversely, if the user selects to filter by all data, then the same choice of BMI would simply scan all the data for all users and delete all the entries for which the BMI does not meet the required value (or lie within the required range).

This procedure is repeated until the user says they wouldn’t like to filter by anything else, allowing for the progressive narrowing of the data that would be further processed and plotted. The list of metrics that can be filtered by also changes depending on single/all users, and the metric selected for the plot.

Once all the data is filtered the filtered user dataframe dictionary is passed onto the processing section.

##### Processing

In the data processing section, the program first eliminates more unnecessary data columns. The ‘ID’ and ‘Date’ columns are deleted. This is because the user ID can be found using the key of the dictionary that the data is stored in, and the ‘Date’ column is no longer necessary as any particular scan for a particular user can be uniquely identified using the ‘Year’ and ‘Day’ columns.

Next, all of the time increment columns, other than the selected one, are deleted. The data (for each user) is then scanned to find the columns sharing the same increment value (within a particular year), and all the scans with the same increment value are averaged over, yielding a single value for any particular time increment number within a particular year, for a specific user.

After this, each data frame is checked to see whether a user is missing data for a specific time increment number, and if so, creating an entry for that number (and year) and populating the remaining metric columns with empty values. For example, if a user is missing data for week 7 in the year 2022, a new entry will be made with week number 7 and year number 2022, and all other values for that entry will be set to empty. Once this is done, the dataframe for any user left in the dictionary should have an entry for every time increment in the range of their earliest recorded value to their last recorded value, within the given filters. The data in each data frame is then sorted chronologically.

Lastly, the program will ask for the minimum time period that the plot should cover, and then delete the data for any users not meeting this requirement. This is not a chronological time period, but a duration time period, filtering for users that possess data spanning a certain duration of time.

The processed data is then passed to the collation section.

##### Collation

This section deals with missing data and interpolation. First, the program scans all the data left and generates a ‘Missing Data Summary’ report. This report includes the percentage of data that is missing amongst the remaining data. Missing data includes the entries that were made previously, containing nothing but year and time increment values. Additionally, the report will display all the users with missing data, along with the percentage of data they are missing within their own timeline of data. It will then do the same for every time point in the data set, showing the index of each time point of data, and the percentage of users missing data for that particular time point. Each of these lists will be printed in descending order of percentage, showing the users and time points missing the most data at the top of the lists, and the ones missing the least at the bottom.

The goal of this section is to maximise the amount of raw data that is averaged and plotted and minimise the amount of data interpolated. This is to provide both more accurate and honest trends, that could be used to gain a more clear image of the effects that the technology is having, as well as not misguide anyone seeing the trends.

Once the summary is displayed, the user will be asked if they wish to interpolate over all the missing data, delete time points with missing data, or users with missing data. If they select to delete time points or users, a new menu will be displayed offering three options:

1) Delete all time points/users with missing data.

2) Delete a specific time point/user with missing data.

3) Delete time points/users with a certain percentage of missing data.

If option 1 is selected then all the time points/users with missing data will be deleted and in theory, there should be no missing data left to interpolate over. This would be ideal, but ill-advised, as most users are missing some data, and most time points are missing data for some users. I expect it would result in plots with almost no data or simply not plot at all in many cases. If option 2 is selected, then the program will request a specific user ID or a specific time point (from the list shown) and will subsequently delete the data for that time point or user if it is found. Lastly, if option 3 is selected, the program will ask for a threshold percentage of missing data above which it will delete users or time points (based on selection)

It is not advised to delete time points from the dataset, except for analytical purposes, as this leaves gaps in the plot generated.

The summary is regenerated each time some data is deleted, and this process is repeated until the user is satisfied with the amount of missing data present. Then, whatever missing data still remains is interpolated over and subsequently either plotted or passed to the aggregation section if that option was selected.

##### Aggregation

In the aggregation section, first, the dictionary of data is checked to find the lowest number of time points present for a specific user, and the dataframes for all the other users are cut such that all users have the same number of time points. Next, a new dataframe is created, to store the aggregate data over all users for each metric at each time point in the data set. The metrics are cycled over, and for each time point for a given metric, the data for all users is averaged and saved to the dataframe.

Once this is done, a plot is generated for the selected metric (or metrics) containing a single line depicting the average trend of multiple users for that specific metric over time (with the given time increment).

#### Things to Take Note Of

The software I made is relatively robust, given this specific use case, and allows for a wide range of plots that can be generated with different characteristics based on the metrics selected by the user. But there are some trade-offs to take note of:

* There is a trade-off between the time increment used and the nature of the plots generated. As the time increment gets larger, fewer users are likely to be missing data, and so fewer data points will have to be interpolated over. This results in both more honest data, and generally smoother trends as the fluctuates of individual user data are minimised. But the number of time points is also reduced as the span of data for most users is limited, and so the data set over which we can generate sufficient points is greatly reduced. Additionally, there may be more subtle patterns that could be missed when finer points are aggregated into larger time increments.
* There is also a trade-off between the interpolation of data and data integrity. Since the trends being tracked involve body composition, the argument can be made that if two points exist for a specific user then the midpoint between them is a good approximation for their data in the point between, and so interpolation is an appropriate way of handling missing data. I believe this is a fair assessment when dealing with a small gap in data, like a week, but when dealing with a gap of several weeks to months, I don’t know if linear interpolation is a fair method of gauging the trends that body composition follows.