A.INTRODUCTION

Personal Health and Wellness Tracker is a Python-based application that empowers individuals to monitor and visualize their health data on a daily basis. By logging key metrics such as sleep hours, mood, and exercise minutes, users can gain valuable insights into their daily habits and well-being. This simple yet powerful tool provides a convenient way to track personal health, identify patterns, and faster better lifestyle choices.

An ideal tool for individuals who want to stay on top of their health, create positive habits, and ultimately improve their quality of life. By providing a clear, graphical summary of key health indicators, the project makes it easy to stay motivated and informed as you work toward your wellness goals.

Sleep Hours Users can log how many hours they sleep each night, which can provide insights into the quality of their sleep and its impact on energy levels, mood, and productivity. Mood Users to record their mood each day, giving them a better understanding of how their emotions fluctuate and what factors may be influencing their mental well-being. Exercise Minutes This feature enables users to track how much physical activity they do each day, helping them stay on top of their fitness goals.

- # ORGANIZATION PROFILE

KICE INFO SYSTEMS is a professional website designing, Customized Software development, Business process Outsourcing – IT/ITES & Internet marketing Company providing full featured web services including B2B Acquisition & B2C Ecommerce solutions and acting as an offshore development center for overseas Development firms. KICE Info systems is an innovative company, based in India that Provides a series of Web-based and software applications that have helped their Customer to create successful business ventures through online initiatives. KICE Info Systems provide all the services that a company needs to get online from web designing To web hosting and manage leading-edge Web sites and e-business applications. Quality and Client Satisfaction are primarily the telling factors of KICE Info systems Success in the domestic market. Ever since its inception, KICE Info systems has Accrued continuous growth in all its business functions and this has been possible only Due to its commitment, quality training methodologies, the services it offers, knowledge Sharing with industry leaders and professional approach. Services Professional Web Design

- SEO concerts, Internet Marketing
- Pay per Click Campaign
- Link Building, Ecommerce Solution
- Web Application Development
- Multimedia Presentations
- Customized Software development

Business Process Outsourcing- IT/IT

- # SYSTEM SPECIFICATION

- # Hardware Configuration

  - **PROCESSOR** : Intel i3
  - **MEMORY** : 4 GB RAM
  - **STORAGE** : 500 MB
  - **DISPLAY** : 1280x720 resolution
  - **KEYBOARD :** Logitech 108 Keys
  - **MOUSE :** Logitech Optical Mouse
  - **Graphics Card** : Integrated graphics

- # Software Specification

  - **FRONT END** : PYTHON 3.7
  - **BACK END** : csv file
  - **LIBRARIES** : Pandas (for data manipulation and storage)
  - **OPERATING SYSTEM**: Windows
  - **IDLE** : GOOGLE COLAB

LANGUAGE SPECIFICATION

The Python language specification is a formal document that defines the syntax, semantics, and grammar of the Python programming language. It serves as the ultimate reference for Python's behavior and provides a detailed explanation of how Python programs should be written and executed.

A brief overview of key areas of the Python language specification

1. Feature:

1. Free and Open Source

Python language is freely available at the official website and you can download it from the given download link below click on the Download Python keyword. Download Python Since it is open-source, this means that source code is also available to the public. So you can download it, use it as well as share it.

2. Easy to code

Python is a high-level programming language. Python is very easy to learn the language as compared to other languages like C, C#, Java script, Java, etc. It is very easy to code in the Python language and anybody can learn Python basics in a few hours or days. It is also a developer-friendly language.

3. Easy to Read

The learning Python is quite simple. As was already established, Python's syntax is really straightforward. The code block is defined by the indentations rather than by semicolons or brackets.

4. Object-Oriented Language

One of the key features of Python is Object-Oriented programming. Python supports object-oriented language and concepts of classes, object encapsulation, etc.

5. GUI Programming Support

Graphical User interfaces can be made using a module such as PyQt5, PyQt4, wx Python, or Tk in Python. PyQt5 is the most popular option for creating graphical apps with Python.

6. High-Level Language

Python is a high-level language. When we write programs in Python, we do not need to remember the system architecture, nor do we need to manage the memory.

7. Large Community Support

Python has gained popularity over the years. Our questions are constantly answered by the enormous Stack Over flow community. These websites have already provided answers to many questions about Python, so Python users can consult them as needed.

8. Easy to Debug

Excellent information for mistake tracing. You will be able to quickly identify and correct the majority issues once understand how to interpret Python's error traces. Simply by glancing at the code, we can determine what it is designed to perform.

Python's language specification encompasses several key features that define its structure and behavior:

- **Interpreter**

  Python code is executed line by line, without the need for compilation into machine code beforehand. This allows for rapid development and easy debugging.

- **Dynamically Typed**

  Variable types are checked during runtime, not declared explicitly. This offers flexibility but can lead to runtime errors if not handled carefully.

- **High-Level**

  Python abstracts away many low-level details, such as memory management, allowing developers to focus on problem-solving.

- **Object-Oriented**

  Python supports object-oriented programming paradigms, enabling the creation of reusable and modular code through classes and objects.

- **General-Purpose**

  Python is versatile and can be used for a wide range of applications, including web development, data science, scripting, and automation.

- **Readable**

  Python's syntax is designed to be clear and concise, resembling natural language, which enhances code readability and maintainability.

- **Extensive Standard Library**

  Python comes with a rich set of built-in modules and functions, providing ready-made solutions for common tasks.

- **Cross-Platform Compatibility**

  Python code can run on various operating systems, including Windows, mac OS, and Linux, without modification.

- **Memory Management**

  Python employs automatic memory management, freeing developers from manual allocation and deallocation of memory.

- **Exception Handling**

  Python provides mechanisms to handle errors gracefully, preventing program crashes and improving robustness.

- **Support for Multiple Programming Paradigms**

  Besides object-oriented programming, Python also supports procedural and functional programming styles.

- **Dynamic**

  Python allows modification of its structure at runtime.

- **Free and Open Source**
  Python is available free of charge and its source code is open to the public.

2. Data Types and Objects

- Primitive Types: Python includes basic data types like int, float, bool, str, etc.
- Collections: Python has several built-in data structures such as lists, tuples, dictionaries, sets, and arrays.
- Classes and Objects: Python supports object-oriented programming (OOP), allowing users to define classes and instantiate objects.
- Dynamic Typing: Python is dynamically typed, meaning you do not need to specify the type of a variable when you declare it.

3. Control Flow

- Conditionals: Python uses if, elif, and else for branching.
- Loops: It supports both for and while loops for iteration.
- Exceptions: Python uses try, except, finally blocks for exception handling.
- Comprehensions: Python offers powerful shorthand methods for creating collections (e.g., list comprehensions, dictionary comprehensions).

4. Functions

- Defining Functions: Functions are defined using the def keyword.
- Arguments: Python supports positional, keyword, and default arguments, along with variable-length argument lists using *args and **kwargs.
- Lambda Functions: Python supports anonymous functions, or lambda functions, which are defined using the lambda keyword.

5. Modules and Packages

- Importing: Python allows modular programming with import statements to bring in external libraries or custom modules.

Data Visualization with Python

1. Feature:

Data visualization provides a good, organized pictorial representation of the data which makes it easier to understand, observe, analyze. In this tutorial, we will discuss how to visualize data using Python.

Python provides various libraries that come with different features for visualizing data. All these libraries come with different features and can support various types of graphs. In this tutorial, we will be discussing four such libraries

PYTHON LIBRARIES AND TOOLS

List of libraries:

- Matplotlib
- Seaborn
- Pandas
- Plotly
- Numpy
- Mac OS

2. Matplotlib

Matplotlib is a widely-used plotting library for Python, providing a flexible and powerful way to visualize data in various forms such as line plots, scatter plots, histograms, bar charts, 3D plots, and more.

Key Features

1. **2D Plotting**: Matplotlib is primarily known for creating 2D plots, making it easy to visualize data trends, relationships, and distributions.
    - **Line Plots**: Display data trends over time or any other continuous variable.
    - **Scatter Plots**: Show the relationship between two variables.
    - **Bar Charts**: Useful for comparing categorical data.
    - **Histograms**: Great for displaying distributions of numeric data.
    - **Pie Charts**: Represent proportions of a whole.
    - **Box Plots**: Provide insights into data distributions and outliers.
2. **Customization**: Matplotlib provides extensive control over the plot's appearance, including:
    - **Titles** and **Labels** for axes.
    - **Legends** for better understanding.
    - **Gridlines** and **Ticks** customization.
    - Control over colors, line styles, and markers.
3. **Interactive Plots**: Although Matplotlib is traditionally used for static plots, it can be used interactively in environments like Jupyter Notebooks. Tools like `smatplotlib inline` make it easy to embed plots in notebooks.
4. **Integration with Other Libraries**: Matplotlib integrates well with other Python libraries like:

- **NumPy** for numerical operations.
- **Pandas** for data manipulation.

- **Seaborn** (built on top of Matplotlib) for enhanced statistical plotting.

5. **Subplots**: Create multiple plots within a single figure using `subplot()` or `subplots()` for organizing complex visualizations.

6. **3D Plotting**: Using `mpl_toolkits.mplot3d`, Matplotlib can be used for creating 3D visualizations, which is useful for representing three-dimensional data.

7. **Exporting Plots**: Plots can be saved in various file formats like PNG, PDF, SVG, and others for sharing and publication.

Seaborn

Seaborn is a Python data visualization library built on top of Matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics.

Key Features

Built-in Themes & Color Palettes: Enhances the aesthetic appeal of plots.

Works Well with Pandas: Handles Data Frames and Series seamlessly.

Statistical Plotting: Supports regression plots, distribution plots, categorical plots, and more.

Automatic Estimation & Aggregation: Helps in summarizing datasets effectively.

Facilitates Complex Visualizations: Like heat maps, violin plots, and pair plots with minimal code.

Commonly Used Functions:

- **sns.histplot ()** – Plots histograms.

- **sns.boxplot ()** – Displays box plots for distributions.

- **sns.scatterplot ()** – Creates scatter plots.

- **sns.lineplot ()** – Used for line plots.

- **sns.heatmap ()** – Generates heat maps for correlation matrices.

- **sns.pairplot ()** – Plots pairwise relationships in a dataset.

- **sns.barplot ()** – Displays bar plots with statistical aggregation.

3. Pandas Visualization:

Pandas provides built-in visualization capabilities using **Matplotlib** as the backend. It allows quick and easy data visualization directly from Pandas **Data Frames** and **Series**.

Key Features

- **Simple & Quick** – No need for external libraries.
- **Integrated with Pandas** – Works seamlessly with Data Frames.
- **Multiple Plot Types** – Line, bar, histogram, scatter, boxplot, and more.
- **Customization Options** – Supports labels, titles, colors, and styles.

Commonly Used Methods:

- **df.plot ()** – Generic plotting method (default: line plot).
- **df.plot.line ()** – Line plot for trends over time.
- **df.plot.bar () / df.plot.barh ()** – Bar & horizontal bar charts.
- **df.plot.hist ()** – Histogram for distribution analysis.
- **df.plot.scatter ()** – Scatter plot for relationships between variables.
- **df.plot.box ()** – Box plot for statistical summaries.
- **df.plot.pie ()** – Pie chart for categorical distributions.

4. Plotly

Key Features

- Fully interactive plots (zoom, hover, pan)
- 3D visualizations & Geospatial maps.

- High-level `plotly.express` and low-level `plotly.graph_objects`.
- Web-based visualizations (Dash integration).
- Exports as HTML, PNG, or JSON.
- Large dataset support using Web GL**.**

Purpose

Plotly is a library for creating interactive plots that can be embedded in web applications.

Uses:

- Creating interactive charts like line plots, bar charts, bubble charts, pie charts, etc.
- Visualizations with zooming, panning, and hover features.
- Supports 3D plots, geographical maps, and complex visualizations.
- Can be used in web-based dashboards (via Dash).

5. NUMPY:

- NumPy is a Python library used for working with arrays.
- It also has functions for working in domain of linear algebra, fourier transform, and matrices.
- NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.
- NumPy stands for Numerical Python.

Python versions:

Python 3.0 (2008)

Python 3.0, also known as "Python 3000" or "Py3k," was a revolutionary release designed to fix inconsistencies and remove redundant constructs from Python 2.x:

- Enhancing consistency and flexibility.

Python 3.4 (2014)

Version 3.4 introduced several significant enhancements:

- **Asyncio**: A framework for writing asynchronous programs, allowing for concurrent code execution.
- **Pathlib**: An object-oriented file system paths library.

Python 3.5 (2015)

Python 3.5 brought in important features for modern programming:

- **Type Hints**: A syntax for adding type annotations to function arguments and return values.
- **Async and Await**: Syntactic support for asynchronous programming, making async code more readable and maintainable.

Python 3.6 (2016)

Python 3.6 was a landmark release with multiple enhancements:

- **Formatted String Literals (f-strings)**: A concise and readable way to embed expressions inside string literals.
- **Asynchronous Generators**: Enhancements to asynchronous programming.

Python 3.7 (2018)

- **Data Classes**: A decorator for automatically generating special methods like __init__ and __repr__ in classes.
- **Context Variables**: A way to manage context-local state.

Python 3.8 (2019)

- **Walrus Operator (:=)**: An assignment expression that allows assignment and return of a value within an expression.
- **Positional-only Parameters**: A way to specify arguments that can only be passed positionally.

Python 3.9 (2020)

- **Dictionary Merge and Update Operators**: New operators | and |= for merging and updating dictionaries.
- **String Methods**: New methods like str.removeprefix () and str.removesuffix ().

Python 3.10 (2021)

Python 3.10 focused on usability and language consistency:

- **Pattern Matching**: A powerful feature for matching complex data structures.
- **Parenthesized Context Managers**: Support for multiple context managers in a single with statement.

Python 3.11 (2022)

Python 3.11 aimed at improving performance and developer experience:

- **Performance Improvements**: Significant speed improvements across various operations.

- **Error Messages**: More informative and precise error messages.

Python 3.12 (2023)

Python 3.12 brings the evolution of the language.

- Improved Error Messages in Python
- More Flexibility in Python F-String
- Type Parameter Syntax
- Improvement in Modules
- Syntactic Formalization of f-strings

Python 3.13 (2024)

The future release of Python 3.12 is expected to bring more optimizations and features, continuing the evolution of the language

1. Enhanced Interactive Interpreter (REPL)

- Multi-line Editing: Allows for more flexible code editing within the interactive shell.
- Color Support: Prompts and trace backs are now colorized by default, enhancing readability.
- Improved Command Handling: Commands like help, exit, and quit can be used directly without parentheses.
- Additional Shortcuts: Function keys such as F1 for help and F2 for browsing history have been introduced.

2. Experimental Free-Threaded CPython

Introduces an experimental mode that allows running Python without the Global Interpreter Lock (GIL), enabling threads to execute in parallel across multiple CPU cores. This feature is not enabled by default and requires a separate executable, typically named python3.13t or python3.13t.exe.

3. Experimental Just-In-Time (JIT) Compiler

A new experimental JIT compiler has been added to enhance performance by compiling code during execution. This feature is not enabled by default and can be activated by building CPython with the --enable-experimental-jit flag.

4. Improved Error Messages

- Error messages are now more informative and include suggestions for fixing issues. Additionally, tracebacks displayed in the terminal are colorized by default, aiding in debugging.

5. Standard Library Updates

- New Additions: Functions like base64. z85 encode() and base64.z85decode() have been introduced for encoding and decoding Z85 data.

- Removals: Deprecated modules such as cgi, nntplib, and others have been removed as part of the standard library cleanup.

These enhancements aim to improve Python's performance, usability, and maintainability.

Back end: CSV

Uses:

CSV (Comma Separated Values) files are commonly used to store and exchange tabular data. Python provides built-in functionalities through the csv module and libraries like pandas to work with CSV files. Here are some common uses of CSV files in Python:

- **Data Storage and Exchange:**

CSV files offer a simple and human-readable format for storing and sharing data. They are widely supported by various applications, making them ideal for data exchange between different systems.

- **Data Analysis and Manipulation:**

Libraries like pandas allow for efficient reading, processing, and manipulation of CSV data. This is crucial for data analysis tasks, where data is often stored in CSV format.

- **Data Import/Export:**

CSV files facilitate the import and export of data from databases, spreadsheets, and other applications. Python's csv module enables seamless interaction with CSV files for these purposes.

- **Configuration Files:**

CSV files can be used to store configuration settings for applications. Their simple structure makes them easy to parse and manage.

- **Logging and Auditing:**

CSV files can be used to record events, transactions, or other relevant information for logging and auditing purposes.

- **Machine Learning:**

CSV is a popular format for storing datasets used in machine learning. Python libraries like scikit-learn can directly read and process CSV data for model training and evaluation.

Benefits:

In Python, the primary benefit of using CSV files is their simplicity and standardized format, allowing for easy data import and export across different applications and platforms, making them highly compatible for data exchange between various software while being easily readable and editable by humans in basic text editors.

Key benefits of CSV in Python:

- **Readability:**

CSV files are plain text with commas as separators, making them easily understandable by humans and simple to view or edit in a text editor.

- **Platform Independence:**

The CSV format is widely recognized across different operating systems and software, ensuring compatibility when sharing data between applications.

- **Data Manipulation:**

Python's csv module provides efficient functions to read, write, and process data from CSV files, making it easy to manipulate large datasets.

- **Data Analysis:**

CSV files are commonly used in data analysis due to their structured format, allowing for straightforward data loading and processing with libraries like Pandas.

- **Ease of Use:**

The csv module is straightforward to implement, making it accessible for both beginners and experienced Python developers.

- **Storage Efficiency:**

For tabular data, CSV files can be a compact way to store information, especially when compared to more complex data formats.

Important points to consider:

- **Data Validation:**

While CSV is convenient, it may not always include data validation mechanisms, so you might need to implement additional checks when working with large datasets.

- **Complex Data Structures:**

For highly structured or nested data, other formats like JSON might be more suitable.

B. SYSTEM STUDY

- ## Existing System

The existing system, as described, tracks basic wellness metrics such as sleep hours, mood, and exercise minutes. It allows the user to log these daily activities and visualize the data in simple plots, offering basic insights into their trends over time. However, it lacks persistent data storage, as the information is lost when the program ends, and the system is limited to just these few metrics without deeper analysis or additional features.

- ## Drawbacks

  - Complex with numerous features that may not be for users seeking a simple tracker.
  - Leading to concerns over data privacy and security.
  - Not all apps provide detailed graphical representations of user data
  - Subscriptions in some apps can be a barrier for users seeking free alternatives.

- ## Proposed System

The Personal Health and Wellness Tracker addresses the common challenge of maintaining a consistent, organized, and easy-to-understand record of personal health data. Often, individuals struggle to track and manage their daily habits, including sleep, exercise, and mental well-being. By offering an intuitive, accessible tool for logging health data, the tracker enables users to identify patterns, gain insights, and make informed decisions about their lifestyle.

This project is designed to serve as a comprehensive personal health tracker that simplifies the process of logging and analyzing health metrics such as sleep hours, mood, and exercise minutes. By allowing users to enter and visualize their data, it empowers them to become more aware of their daily habits and how these habits influence their overall well-being. Below is an in-depth explanation of how this tracker helps users improve their health by providing actionable insights.

- ## Features
    - Enhanced Visualizations: Interactive charts and graphs for tracking progress.
    - User Authentication**:** Supports multiple profiles for individual tracking.
    - Mobile and Cloud Integration**:** Sync data across devices via mobile app or web interface
    - Reminders and Notifications**:** Prompts for logging data and tracking goals.
    - Saves the plot image as a file named **`wellness_plot.png`** by default.

C. System Design and Development

• Form Design

Form design is the process of creating structured documents or digital interfaces that collect user data efficiently and intuitively. A well-designed form ensures clarity, ease of use, and accessibility, making the data collection process smooth for both users and administrators. The key principles of form design include simplicity, logical structure, and user-friendliness. Forms should be concise, with only essential fields, and organized in a logical flow that guides users naturally from one section to another. Labels should be clear and descriptive, while input fields must be appropriately chosen, such as dropdowns for multiple-choice questions and radio buttons for binary responses. Accessibility is also crucial, requiring mobile-friendly layouts, proper color contrasts, and compliance with accessibility standards to cater to all users.

Visual design plays a significant role in form usability, ensuring readability through consistent fonts, whitespace, and intuitive alignment of elements. Error messages and real-time validation help users correct mistakes as they fill out the form, while confirmation messages provide reassurance after submission. Security is another essential aspect, requiring encryption for sensitive data, CAPTCHAs to prevent spam, and privacy policies to maintain transparency. Depending on the use case, forms can be paper-based, digital, interactive web forms, or mobile-optimized versions. Multi-step forms are useful for complex submissions, breaking the process into manageable sections to enhance user experience. A well-crafted form is not just a data collection tool but a seamless communication channel between users and organizations.

Button Styling:

- Buttons should have a consistent and clear look (e.g., rectangular with borders and a color that stands out).
- Place the "Exit" button at the bottom-right or center, so it's easy to find but not in the way of primary actions (Log Data or Visualize Data).

Success/Error Feedback:

Display success or error messages clearly with pop-up windows that provide instant feedback to the user.

- # Input Design

Input Design is one of the most expensive phases of the operation of computerized system and it is often the major problem of a system. A large number of problems with a system can usually be tracked backs to fault input design and method. Needless to say, therefore, that the input data is the life blood of a system and have to be analyzed and designed with utmost case and consideration. The decisions made during the input design are,

➤ To provide cost effective method of input.

➤ To achieve the highest possible level of accuracy.

➤To ensure that the input is understand by the user.

Input data of a system may not be necessarily be raw data captured in the system from scratch. These can also be the output of another system or sub system. The design of input covers all phases of input from the creation of initial data to actual entering the data to the system for processing. The design of inputs involves identifying the data needed, specifying the characteristics of each data item, capturing and preparing data for computer processing and ensuring correctness of data.

User Input

The project prompts the user to input the following daily health metrics:

- **Sleep Hours**: The number of hours the user slept that day.
- **Mood**: The user's mood on a scale from 1 to 10.
- **Exercise Minutes**: The number of minutes the user spent exercising

- # Output Design

Output Design generally refers to the results and information's that are generated by the system for many end-users, output is the main reason for developing the system and the basis on which they evaluate the usefulness of the application.

The objective of a system finds its shape in terms of the output. The analysis of the objective of a system leads to determination of outputs. Outputs of a system can face various forms. The most common are reports, screen displays, printed forms, graphical drawings etc.,

The output can also be varied in terms of their contents frequency, timing and format. The user of the output from a system are the justification for its existence. If the output are inadequate in any way, the system are itself is adequate. The basic requirements of output are that it should be accurate, timely and appropriate, in terms of content, medium and layout for its intended purpose.

- **Textual Output**: A confirmation message that the data has been logged successfully.
- **Graphical Output**: The project displays three different plots:
    - **Sleep Hours Over Time**: A line graph showing the user's sleep hours on the y-axis over the days on the x-axis.
    - **Mood Over Time**: A line graph showing the user's mood on a scale of 1 to 10.
    - **Exercise Minutes Over Time**: A line graph depicting the number of minutes the user exercised time.

- # Database Design

The most important consideration in designing the database is how information will be used. The main objectives of designing a database are:

Data Integration

In a database, information from several files are coordinated, accessed and operated Upon as through it is in a single file. Logically, the information are centralized ,physically, the data may be located on different devices, connected through data communication facilities.

Data Integrity

Data integrity means storing all data in one place only and how each application to access it. This approach results in more consistent information, one update being sufficient to achieve a new record status for all applications, which use it. This leads to less data redundancy; data items need not be duplicated; a reduction in the direct acces storage requirement.

Data Independence

Data independence is the insulation of application programs from changing aspects of physical data organization. This objective seeks to allow changes in the content and organization of physical data without reprogramming of applications and to allow modifications to application programs without reorganizing the physical data.

The tables needed for each module were designed and the specification of each and every column was given based on the records and details collected during record specification of the system study.

That expands the database design by using csv instead of a simple list. It includes a `date` field, stores data persistently, and improves visualization.

- If implemented csv database will store logged data
- Fields include Date, Sleep Hours, Mood, Exercise Minutes

- **System Development**

The key to control maintenance costs is to design systems that are easy to change, so the link between development and maintenance is very strong. Many of the analysis and design methodologies, tools, and techniques employed during system development can be applied to system maintenance, but there are significant differences between development and maintenance. Maintainability is the ease with which software can be understood, corrected, adopted and enhanced.

- **Description of Modules**

Data Collection

The Data Collection Module serves as the starting point of the system where users input their daily health data. It collects information such as sleep hours, mood ratings on a scale from 1 to 10, and exercise minutes performed each day. The module ensures that the interface for data entry is simple and user-friendly so that users can log their information consistently without facing technical difficulties. It includes basic validation checks to prevent errors such as entering negative values or unrealistic numbers .The module also records the date automatically, ensuring that each entry is time-stamped correctly for future reference. The goal of this module is to make the process of logging data quick, accurate, and accessible to all users regardless of their technical expertise.

Pre-processing

After the raw data is collected, the Data Pre-processing Module prepares it for storage and analysis by organizing it into a structured format. It handles tasks such as converting data into appropriate data types, ensuring all dates follow the same format, and normalizing values if needed. For example, mood scores might need scaling for comparison with exercise or sleep data in some analyses. The module also addresses missing data by using default values or placeholders so that gaps in the dataset do not disrupt analysis or visualization. By cleaning and structuring the data at this stage, it ensures consistency and accuracy before proceeding to storage or visualization, thus maintaining the reliability of the entire system.

Data Storage

Once cleaned, the data must be stored securely for future use, and this is the responsibility of the Data Storage Module. It stores the processed data in a CSV file format, ensuring both portability and simplicity. Each day's entry—comprising the date, sleep hours, mood, and exercise

minutes—is stored in tabular form so that it can be retrieved whenever needed. Storing data in CSV files makes it easy to share across platforms, import into data analysis tools, or migrate to databases later if required. The module ensures persistence, meaning the data remains safe even after the program ends, unlike temporary in-memory storage.

Visualization

The Visualization Module transforms raw data into meaningful insights through graphical representations. It uses libraries like Matplotlib, Seaborn, or Plotly to generate line graphs, bar charts, or scatter plots showing trends in sleep hours, mood fluctuations, and exercise routines over time. Visualization makes it easier for users to observe patterns, identify habits, and assess lifestyle changes. For instance, a user might notice that their mood improves on days when they exercise more or sleep longer. Interactive charts can also be implemented, allowing users to zoom in, hover over data points, and view specific details, thereby enhancing the analysis experience.

Line Graphs

Line graphs in the Personal Health and Wellness Tracker project are used to visualize daily trends in Sleep Hours**,** Mood Ratings, and Exercise Minutes over time. By plotting dates on the x-axis and health metrics on the y-axis, users can easily see patterns, fluctuations, and correlations. Created using Matplotlib or Seaborn**,** these graphs help turn raw data into clear, actionable insights, making it easier for users to track habits and improve their lifestyle.

User Interaction and Output

The User Interaction and Output Module bridges the gap between the system and its users. It provides real-time feedback through confirmation messages when data is logged successfully or error messages when invalid inputs are detected. The module also manages the display of visualizations instantly after data entry, ensuring users can immediately view trends in their health metrics. By maintaining smooth two-way communication, this module enhances usability and ensures that users remain engaged and motivated to track their health regularly.

Database Design

The Database Design Module organizes data systematically to maintain integrity, consistency, and accessibility. Using a CSV-based structure, it defines fields such as Date, Sleep Hours, Mood, and Exercise Minutes, ensuring every piece of information is stored correctly in a predefined format. Proper indexing and structuring help reduce redundancy and improve data retrieval speed. Though CSV files are used here for simplicity, the design approach allows easy migration to advanced databases like MySQL or MongoDB in the future if the application is scaled up, ensuring flexibility and long-term usability.

4 Software Testing and Implementation

The Personal Health and Wellness Tracker was implemented using Python 3.7 with libraries such as Pandas for data handling, Matplotlib for visualization, and CSV for persistent data storage. The implementation was done in stages to ensure smooth integration of all modules, including data logging, validation, storage, and visualization. The application collects daily health metrics such as Sleep Hours**,** Mood Ratings, and Exercise Minutes from the user and stores the data in a structured CSV format for future retrieval and visualization.

Software Testing

System testing is a critical phase in the project, ensuring that the entire application works as intended when all modules are integrated. The main objective is to verify that the system meets the specified requirements, functions correctly under various scenarios, and handles all types of inputs gracefully.The testing was performed after completing unit testing and integration testing for individual components such as data logging, input validation, CSV storage, and data visualization. System testing involved running the complete application with real-time user inputs to verify end-to-end functionality.

Objectives of System Testing

- To validate that the system satisfies all functional and non-functional requirements.
- To ensure smooth interaction between all modules: input, storage, processing, and visualization.
- To confirm that invalid inputs and exceptional cases are handled gracefully without system crashes.
- To check data persistence using CSV files across multiple program executions.
- To evaluate performance with large datasets for accuracy and speed.

Testing Issues

During the testing phase of the project, a few minor issues were encountered and addressed

Invalid Input Handling

Initially, the system did not provide clear error messages for incorrect inputs such as negative values or mood ratings above 10. This was fixed by implementing proper validation with user-friendly alerts.

Duplicate Date Entries

When users entered data for the same date multiple times, duplicate records appeared in the CSV file. The issue was resolved by adding logic to overwrite existing entries for the same date instead of creating duplicates.

Visualization Formatting

Early tests showed overlapping date labels on the x-axis when visualizing large datasets. This was fixed by adding date rotation, spacing, and a rolling average line for better clarity.

Performance with Large Datasets

With multiple months of data, initial tests showed slower plot rendering. Optimization using rolling averages and efficient data handling in Pandas improved performance.

User Interaction Flow

Some users found the command-line interaction slightly confusing during the first testing phase. Instructions were simplified, and clearer prompts were added for better usability

Testing Methodologies

The testing methodologies for the Personal Health and Wellness Tracker project were designed to ensure that the system was reliable, accurate, and user-friendly**.** Unit Testing was performed first to validate individual components such as data logging, input validation, and visualization, ensuring each function worked correctly on its own. Next, Integration Testing was conducted to verify smooth interaction between modules, confirming that data flowed properly from input to storage and visualization. Functional Testing was carried out to ensure all system requirements were met, including error handling, duplicate entry management, and plot saving. System Testing was performed to validate the complete application under real-world conditions, while Performance Testing ensured that the system handled large datasets efficiently.

Unit Testing

Unit testing was performed to ensure that each function in the project worked correctly before integrating the entire system. The `log_data()` function was tested with valid, invalid, and boundary inputs to verify accurate data logging, validation checks, and handling of duplicate entries. The `visualize_data()` function was tested to confirm that line graphs were generated correctly for Sleep Hours, Mood, and Exercise Minutes, even with large datasets, and that plots could be saved successfully. Additionally, the `main()` function was tested to ensure smooth user interaction, accurate prompts, proper error handling for incorrect inputs, and graceful termination

when the exit command was given. These tests confirmed that each module worked independently and reliably, forming a strong foundation for integration and system testing.

Integration Testing

Integration testing was carried out after successful unit testing to ensure that all modules of the project worked together seamlessly. The primary objective was to verify smooth data flow between modules such as data collection, validation, storage, and visualization.

Ensure all modules of the Personal Health and Wellness Tracker worked together smoothly. Data entered by the user was validated, stored in the CSV file, and visualized accurately without errors or data loss.

Functional Testing

Functional testing was conducted to ensure that all features of the project worked as intended according to the project requirements. This included verifying that user inputs were properly validated, data was stored correctly in the CSV file, duplicate entries were managed, and visualization graphs for sleep hours, mood, and exercise minutes were generated accurately.

System Testing

System testing was performed after unit, integration, and functional testing to validate the entire Personal Health and Wellness Tracker as a complete system. It ensured that all modules—from data input and validation to storage and visualization—worked together seamlessly. The testing confirmed that the system handled valid and invalid inputs correctly, displayed accurate visualizations, saved plots as required, and provided a smooth user experience without errors or crashes.

Performance Testing

Performance testing was conducted to ensure the project handled large datasets efficiently without affecting speed or accuracy. Multiple months of data were used to test system responsiveness, data processing speed, and visualization performance. The results confirmed that the system remained fast, stable, and reliable even with high data volumes.

User Acceptance Testing (UAT)

User Acceptance Testing (UAT) was performed to ensure this project met real user requirements and provided a smooth, intuitive experience. A group of sample users tested the system by logging daily data, visualizing health trends, and saving plots. Feedback was collected on usability,

accuracy, speed, and overall experience. Users confirmed that the system was simple to use, displayed clear error messages for invalid inputs, and generated accurate visualizations for Sleep Hours, Mood, and Exercise Minutes

Quality Assurance

Quality assurance consists of the auditing and reporting functions of management. The goal of quality assurance is to provide management with the data necessary to be informed about product quality, thereby gaining insight and confidence that product quality is meeting its goal.

Quality Assurance Goals:

Correctness

Input values are validated (e.g. sleep must be 0–24). Dates are parsed carefully.

Reliability

Handles duplicate entries by overwriting for the same day, ensuring consistent results.

Efficiency

Minimal resources used (CSV for storage, pandas for efficient data ops).

Usability

Simple user prompts, emoji indicators, graceful error handling.

Maintainability

Clean function structure; future changes can be added modularly.

Testability

Input handling and logging logic can be unit tested easily.

Portability

Python + CSV makes it usable on any platform without setup.

Accuracy

Rolling averages improve clarity of trends; float precision supported.

Generic Risk

Input validation prevents most invalid/edge cases.

Security Technologies and Policies

Security technologies and policies are essential components of any software project to ensure data protection, prevent unauthorized access, and maintain user privacy. Although the project is a simple personal health monitoring tool, it integrates fundamental security measures that align with standard best practices. One of the key technologies implemented is **input validation**, which

ensures that user inputs such as sleep hours, mood ratings, and exercise minutes fall within acceptable ranges. This not only prevents data corruption but also guards against malicious input or unintentional logic errors.

The project also demonstrates secure file handling by saving user data in a structured CSV format within the local file system.

System Implementation

The project is a Python-based application that records daily wellness metrics sleep, mood, and exercise—and stores them securely in a local CSV file. It uses **pandas** for efficient data handling and **matplotlib** to generate insightful trend visualizations with rolling averages. The system ensures data accuracy through strict input validation and updates existing records for the same day to avoid duplicates. Designed with a modular structure, it runs offline, is lightweight, and provides a clean foundation for future features like encryption, cloud sync, or a graphical user interface (GUI).

The Stage Consists Of:

- Gathering user inputs for sleep hours, mood, and exercise minutes.
- Ensuring inputs are within valid ranges to maintain data accuracy.
- Saving the validated data in a CSV file using pandas, with updates for existing entries.
- Creating graphs with matplotlib to display trends and rolling averages over time.
- Providing a simple interface for input, visualization, saving, and exiting the system.

Implementation Procedures

The implementation process begins with setting up the necessary environment, importing essential libraries such as pandas for data management, matplotlib for visualization, and datetime for handling dates.

The system first attempts to load existing wellness data from a CSV file; if none exists, it initializes an empty dataset with predefined columns for date, sleep hours, mood, and exercise minutes. Users are prompted to input their daily wellness data through a command-line interface, with clear instructions on acceptable ranges to ensure meaningful data collection.

Each input undergoes rigorous validation to confirm that values fall within logical limits, such as sleep hours between 0 and 24 and mood levels from 1 to 10. If an entry for the current date already exists, it is overwritten to maintain data accuracy and prevent duplication. After updating or adding new entries, the data is saved back to the CSV file for persistence.

Throughout the process, error handling is implemented to manage invalid inputs gracefully, ensuring a user-friendly experience. The modular design of the code, with clear functions dedicated to logging and visualization, facilitates ease of maintenance and allows for future enhancements such as adding encryption or a graphical user interface.

User Training

User Training is a critical phase aimed at equipping users with the knowledge and skills necessary to efficiently test, operate, and maintain the system. Proper training minimizes errors, improves user confidence, and ensures smooth adoption of the system.

1. **User Manual:**

   A comprehensive document that explains the system's features, functionalities, and step-by-step instructions. It serves as a handy reference guide for users to understand how to perform various tasks and troubleshoot common issues.

2. **Help Screens:**

   Context-sensitive help available within the software interface. These interactive help sections provide immediate assistance by offering detailed descriptions, tips, and solutions relevant to the current task or screen, enhancing user self-sufficiency.

3. **Training Demonstrations:**

   Live, instructor-led sessions where users observe and practice the system's operations in a controlled environment. These demonstrations often include simulations, role-playing, and hands-on activities, which help users gain practical experience and build confidence.

System Maintenance

System Maintenance is an ongoing process that ensures a system continues to function effectively after its initial deployment. It involves updating, correcting, and improving the system to meet evolving user needs and changing technical environments. Proper maintenance helps extend the system’s lifespan, enhances performance, and reduces the risk of unexpected failures.

Without regular maintenance, software can become outdated, less secure, and inefficient, leading to increased costs and user dissatisfaction. Maintenance is a critical phase in the software lifecycle, requiring careful planning, skilled resources, and clear documentation to manage changes smoothly and minimize disruptions.

- Adaptive maintenance updates the system to remain compatible with changing environments.
- Performance monitoring tracks system efficiency to identify and resolve issues early.
- Automation streamlines routine maintenance tasks to reduce errors and save time.
- Documentation updates keep manuals and guides aligned with system changes.
- Backup and recovery plans prevent data loss and ensure business continuity.
- Fixing bugs and errors discovered after deployment to ensure the system works as intended.
- Enhancing and adding new features to improve system functionality and user experience.
- Making changes proactively to avoid future problems, such as optimizing code and performing regular system audits.
- Analyzing system usage trends to plan for future resource needs.
- Performing routine tests to verify system functionality after updates or changes
- Distributing workload evenly across servers to improve performance and reliability.

Corrective Maintenance

This involves identifying and fixing defects, errors, or faults discovered after the system has been deployed. The primary goal is to correct issues that cause system malfunction or degrade performance, ensuring the system operates as intended.

Perfective Maintenance

Perfective maintenance focuses on improving the system's functionality and performance by adding new features, refining existing functions, or enhancing user experience. It responds to changing user requirements and aims to keep the system relevant and efficient over time.

Preventive Maintenance

Preventive maintenance is proactive and aims to anticipate and resolve potential problems before they occur. This type of maintenance involves activities such as code refactoring, performance tuning, and system audits to avoid future failures and extend the system's lifespan.

CONCLUSION

The project successfully achieved its goal of providing a simple yet effective tool for monitoring daily health parameters such as sleep hours, mood ratings, and exercise minutes. By integrating modules for data collection, validation, storage, visualization, and user interaction, the system offers a seamless experience for tracking health trends over time.

It is evident that the wellness data tracking system provides valuable insights into users' health habits, helping individuals monitor and improve their sleep, mood, and exercise routines.
The system's ease of use encourages regular data entry, which is critical for accurate trend analysis.

- Visual representations such as graphs help users quickly identify improvements or areas needing attention.
- The integration of user training and maintenance ensures the system remains reliable and effective over time.
- Proper system maintenance, including corrective, perfective, and preventive measures, extends the system's lifespan and adaptability.
- User engagement through manuals, help screens, and training demonstrations significantly enhances system adoption and success.

The system's user-friendly design, supported by comprehensive user training including manuals, help screens, and demonstrations, ensures smooth adoption and efficient usage. Furthermore, systematic maintenance—covering corrective, perfective, and preventive tasks—guarantees the system remains reliable and adaptable to future needs.

The Following Measures Are Suggested For Improving User Engagement And System Effectiveness

- Providing detailed and easy-to-understand user manuals for quick reference.
- Incorporating interactive help screens within the software for on-demand assistance.
- Conducting regular training demonstrations to enhance user familiarity.
- Ensuring timely maintenance to fix issues and add new features

By adopting these measures, the wellness tracker can effectively support users in leading healthier lives while maintaining a robust and user-centered system.

BIBLIOGRAPHY

Referenced Books:

- **McKinney, W. (2017)** Python for Data Analysis: Data Wrangling with Pandas, NumPy, and I Python. O'Reilly Media.
- **Agarwal, R. (2020)** Hands-On Data Visualization with Python: Create compelling graphics and interactive dashboards with Matplotlib and Seaborn. *Packet Publishing.*
- **Walker, S. (2019)** Sleep: The New Science of Sleep and Dreams. *Penguin.*
- **Ratey, J. J. (2008*)*** Spark: The Revolutionary New Science of Exercise and the Brain. Little, Brown Spark.
- **Neil A. Campbell and Jane B. Reece(2006) :**Campbell Biology (11th Edition)
- **Richard Dawkins (2013):** The Biology of Belief
- **Arthur M. Lesk (2022):** Introduction to Bioinformatics
- **VanderPlas, J. (2016)** :Python Data Science Handbook: Essential Tools for Working with Data. O'Reilly Media
- **Pandas Development Team (2024**): pandas (Version 2.x) Documentation – Used for data manipulation and analysis in the tracker
- **Python Software Foundation (2024)**: *Python 3.x Documentation* – The core programming language used in the project

Referenced Websites:

- https://pandas.pydata.org/docs/
- https://matplotlib.org/stable/contents.html
- https://www.cdc.gov/sleep/index.html
- https://www.health.harvard.edu/exercise-and-fitness
- https://www.who.int/news-room/fact-sheets/detail/physical-activity
- https://www.nhlbi.nih.gov/health/sleep
- https://www.apa.org/topics/mood
- https://seaborn.pydata.org

APPENDICES

D.Data Flow Diagram

User Input:

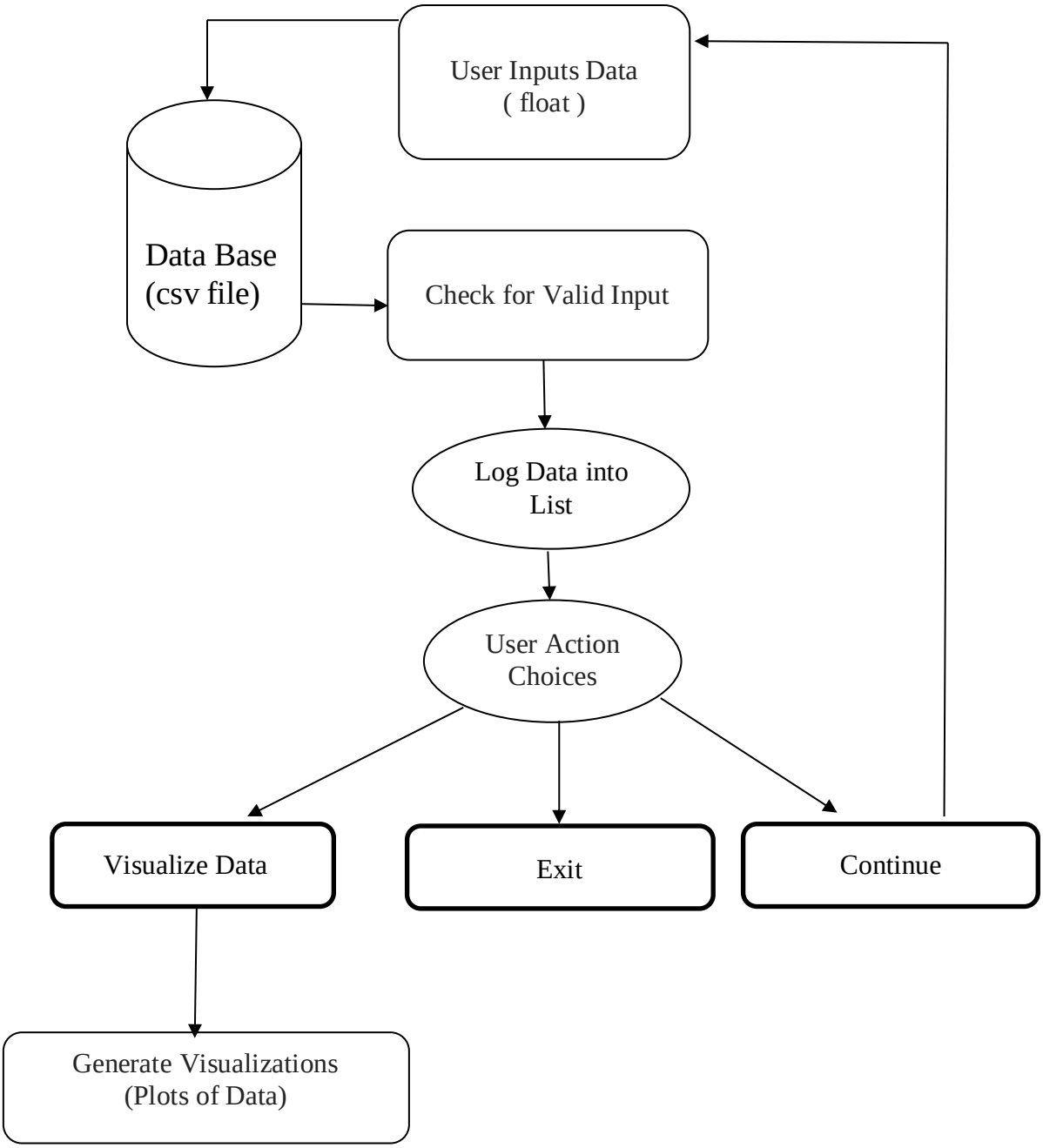
- **Inputs**:
    - Sleep hours (`sleep_hours`).
    - Mood (`mood`).
    - Exercise minutes (`exercise_minutes`).

2. System Process:

- **Log Data**:
    - The user input is logged into the system.
    - The data is appended to the `data` list, structured as dictionaries.

3. System Output:

- **Graphical Output**:
    - **Plot 1**: "Sleep Hours Over Time"
    - **Plot 2**: "Mood Over Time"
    - **Plot 3**: "Exercise Minutes Over Time"
    - These plots are shown on the user interface using **Matplotlib**.
    - If no data is available, an error message "No data to visualize" is shown.


User Inputs Data
(float)
Data Base
(csv file)
Check for Valid Input
Log Data into
List
User Action
Choices
Visualize Data
Exit
Continue
Generate Visualizations
(Plots of Data)

E.Table Structure

Field Name	Data Type	Size	Description
Date	Date	10	The date of the entry.
Sleep Hours	Float	4	Number of hours the user slept.
Mood	Integer	2	Mood rating on a scale of 1 to 10.
Exercise Minutes	Integer	6	Duration of exercise in minutes.

F.Sample Coding

This project tracks daily wellness metrics including sleep hours, mood level, and exercise duration. These three parameters are critical indicators of a person's overall health and well-being. By recording this information every day

The data is logged daily and securely stored in a CSV file format, enabling easy access and management. Using this historical data, the project provides visualization tools to display trends over time.

```
import pandas as pd
from datetime import datetime
from pathlib import Path
import matplotlib.pyplot as plt
DATA_FILE = Path("wellness_data.csv")
if DATA_FILE.exists():
    data = pd.read_csv(DATA_FILE, parse_dates=["Date"])
else:
data = pd.DataFrame(columns=["Date", "Sleep Hours", "Mood", "Exercise Minutes"])
data["Date"] = pd.to_datetime(data["Date"], errors='coerce')
def log_data(sleep_hours: float, mood: float, exercise_minutes: float):
    global data
    today = pd.to_datetime(datetime.today().date())
    if not (0 <= sleep_hours <= 24):
        raise ValueError("Sleep hours must be between 0 and 24.")
    if not (1 <= mood <= 10):
        raise ValueError("Mood must be between 1 and 10.")
    if not (0 <= exercise_minutes <= 1440):
        raise ValueError("Exercise minutes must be between 0 and 1440.")
    new_entry = {
        "Date": today,
        "Sleep Hours": sleep_hours,
        "Mood": mood,
        "Exercise Minutes": exercise_minutes
```

```
    }
    if today.date() in data["Date"].dt.date.values:
        print("Today's entry already exists. Overwriting...")
        data = data[data["Date"].dt.date != today.date()]
    data = pd.concat([data, pd.DataFrame([new_entry])], ignore_index=True)
    data = data.sort_values("Date")
    data.to_csv(DATA_FILE, index=False)
    print(" Data logged and saved successfully!")
def visualize_data(data: pd.DataFrame, save_plot: bool = False):
    if data.empty:
        print("⚠ No data to visualize.")
        return
    df = data.sort_values("Date")
    df["Date"] = pd.to_datetime(df["Date"])
    df.set_index("Date", inplace=True)
    df = df.sort_index()
    df_rolling = df.rolling(window=3, min_periods=1).mean()
    plt.figure(figsize=(12, 10))
    plt.subplot(3, 1, 1)
    plt.plot(df.index, df["Sleep Hours"], 'o-', label="Sleep Hours", color="blue")
    plt.plot(df_rolling.index, df_rolling["Sleep Hours"], '--', color="cyan", label="3-Day Avg")
    plt.title("Sleep Hours Over Time")
    plt.ylabel("Hours")
    plt.legend()
    plt.grid(True)
    plt.subplot(3, 1, 2)
    plt.plot(df.index, df["Mood"], 'o-', label="Mood", color="green")
    plt.plot(df_rolling.index, df_rolling["Mood"], '--', color="lime", label="3-Day Avg")
    plt.title("Mood Over Time")
    plt.ylabel("Mood Level")
    plt.legend()
```

```
    plt.grid(True)
    plt.subplot(3, 1, 3)
    plt.plot(df.index, df["Exercise Minutes"], 'o-', label="Exercise", color="red")
    plt.plot(df_rolling.index, df_rolling["Exercise Minutes"], '--', color="orange", label="3-Day
Avg")
    plt.title("Exercise Minutes Over Time")
    plt.xlabel("Date")
    plt.ylabel("Minutes")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.xticks(rotation=45)
    if save_plot:
        plt.savefig("wellness_plot.png")
        print("Plot saved as 'wellness_plot.png'.")
    plt.show()
def main():
    print(" Welcome to the Wellness Tracker!")
    while True:
        try:
sleep_hours = float(input("Enter sleep hours for today (0-24): "))
            mood = float(input("Enter your mood (1-10): "))
exercise_minutes = float(input("Enter exercise minutes (0-1440): "))
            log_data(sleep_hours, mood, exercise_minutes)
        except ValueError as ve:
            print(f"{ve}")
            continue
        while True:
            action = input(
"\nType 'visualize' to view graphs, 'save' to save plot, 'continue' to log another day, or 'exit' to
quit:"
            ).strip().lower()
            if action == "visualize":
                visualize_data(data)
```

```
        elif action == "save":
            visualize_data(data, save_plot=True)
        elif action == "continue":
            break
        elif action == "exit":
print("Exiting Wellness Tracker. Stay consistent and healthy!")
            return
        else:
            print("Invalid input. Please try again.")
if __name__ == "__main__":
    main()
```

Basic information of Data like Shape, data type, null values, Unique Characters

```
import pandas as pd
from pathlib import Path
# File path for CSV
DATA_FILE = Path("wellness_data.csv")
# Load the dataset if exists, else create empty DataFrame
if DATA_FILE.exists():
    data = pd.read_csv(DATA_FILE, parse_dates=["Date"])
else:
    data = pd.DataFrame(columns=["Date", "Sleep Hours", "Mood", "Exercise Minutes"])
# Basic Information
# 1. Shape of the dataset (rows, columns)
print("Shape of the dataset:", data.shape)
# 2. Data types of each column
print("\nData types:")
print(data.dtypes)
# 3. Check for null/missing values in each column
print("\nNull values in each column:")
print(data.isnull().sum())
# 4. Number of unique values in each column
print("\nUnique values per column:")
print(data.nunique())
```

WellnessLogger

```
import pandas as pd
from datetime import datetime
from pathlib import Path
# File path for CSV
DATA_FILE = Path("wellness_data.csv")
# Load existing file or create empty DataFrame
if DATA_FILE.exists():
    data = pd.read_csv(DATA_FILE, parse_dates=["Date"])
else:
    data = pd.DataFrame(columns=["Date", "Sleep Hours", "Mood", "Exercise Minutes"])
# Ensure Date column is datetime
data["Date"] = pd.to_datetime(data["Date"], errors='coerce')
def log_mood(mood: float):
    """Logs the mood."""
    today = pd.to_datetime(datetime.today().date())
    # Validation
    if not (1 <= mood <= 10):
        raise ValueError("Mood must be between 1 and 10.")
    # Log the data
    new_entry = {
        "Date": today,
        "Sleep Hours": None,  # No sleep value here
        "Mood": mood,
        "Exercise Minutes": None  # No exercise value here
    }
    # If today's entry exists, overwrite
    if today.date() in data["Date"].dt.date.values:
        data = data[data["Date"].dt.date != today.date()
    data = pd.concat([data, pd.DataFrame([new_entry])], ignore_index=True)
    data.to_csv(DATA_FILE, index=False)
    print("✅ Mood logged successfully!")
```

Data Analysis

Line chart

```
import matplotlib.pyplot as plt
import pandas as pd
def line_charts(data: pd.DataFrame, save_plot: bool = False):
    if data.empty:
        print("⚠ No data to visualize.")
        return
    df = data.sort_values("Date")
    df["Date"] = pd.to_datetime(df["Date"])
    df.set_index("Date", inplace=True)
    # Rolling average for smoothing
    df_rolling = df.rolling(window=3, min_periods=1).mean()
    plt.figure(figsize=(12, 10))
    # Sleep chart
    plt.subplot(3, 1, 1)
    plt.plot(df.index, df["Sleep Hours"], 'o-', color="blue", label="Sleep Hours")
    plt.plot(df_rolling.index, df_rolling["Sleep Hours"], '--', color="cyan", label="3-Day Avg")
    plt.title("Sleep Hours Over Time")
    plt.ylabel("Hours")
    plt.legend()
    plt.grid(True)
    # Mood chart
    plt.subplot(3, 1, 2)
    plt.plot(df.index, df["Mood"], 'o-', color="green", label="Mood")
    plt.plot(df_rolling.index, df_rolling["Mood"], '--', color="lime", label="3-Day Avg")
    plt.title("Mood Over Time")
    plt.ylabel("Mood Level")
    plt.legend()
    plt.grid(True)
    # Exercise chart
```

```
    plt.subplot(3, 1, 3)
    plt.plot(df.index, df["Exercise Minutes"], 'o-', color="red", label="Exercise")
     plt.plot(df_rolling.index, df_rolling["Exercise Minutes"], '--', color="orange", label="3-Day
                                                    Avg")
    plt.title("Exercise Minutes Over Time")
    plt.xlabel("Date")
    plt.ylabel("Minutes")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.xticks(rotation=45)
    if save_plot:
        plt.savefig("wellness_line_chart.png")
        print("💾 Line chart saved as 'wellness_line_chart.png'.")
    plt.show()
```

Classification

Decision Tree

```
if not (0 <= sleep_hours <= 24):
    raise ValueError("Sleep hours must be between 0 and 24.")
if not (1 <= mood <= 10):
    raise ValueError("Mood must be between 1 and 10.")
if not (0 <= exercise_minutes <= 1440):
    raise ValueError("Exercise minutes must be between 0 and 1440.")
```

G.Sample Input

Enter sleep hours for today (0-24): 7.5

Enter your mood (1-10): 8

Enter exercise minutes (0-1440): 45

✅ Data logged and saved successfully!

Type 'visualize' to view graphs, 'save' to save plot, 'continue' to log another day, or 'exit' to quit: visualize

Type 'visualize' to view graphs, 'save' to save plot, 'continue' to log another day, or 'exit' to quit: save

Type 'visualize' to view graphs, 'save' to save plot, 'continue' to log another day, or 'exit' to quit: continue

H.Sample Output

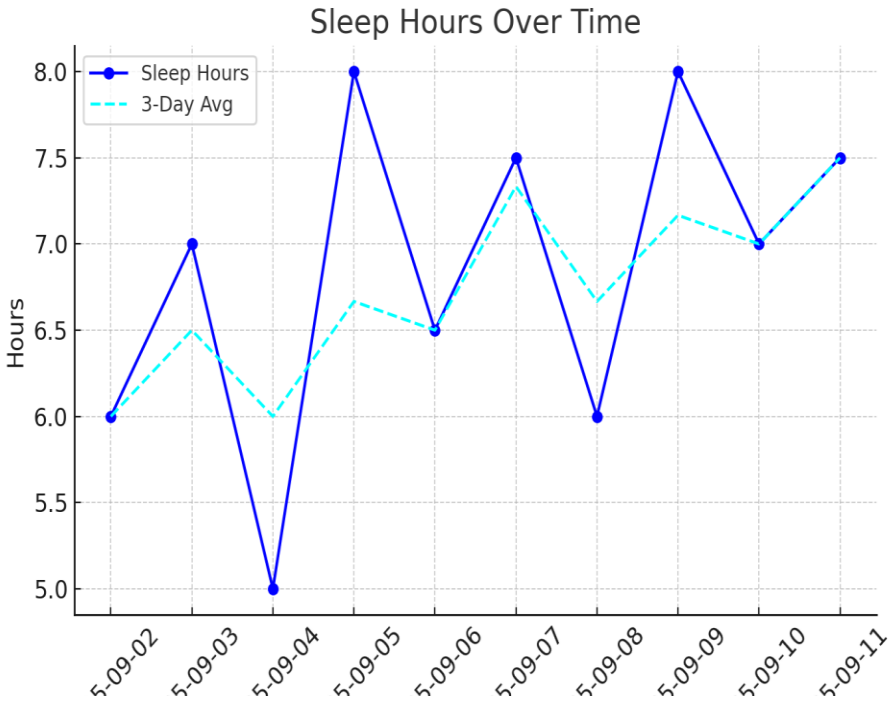
Welcome to the Wellness Tracker!

Enter sleep hours for today (0-24): 7.5

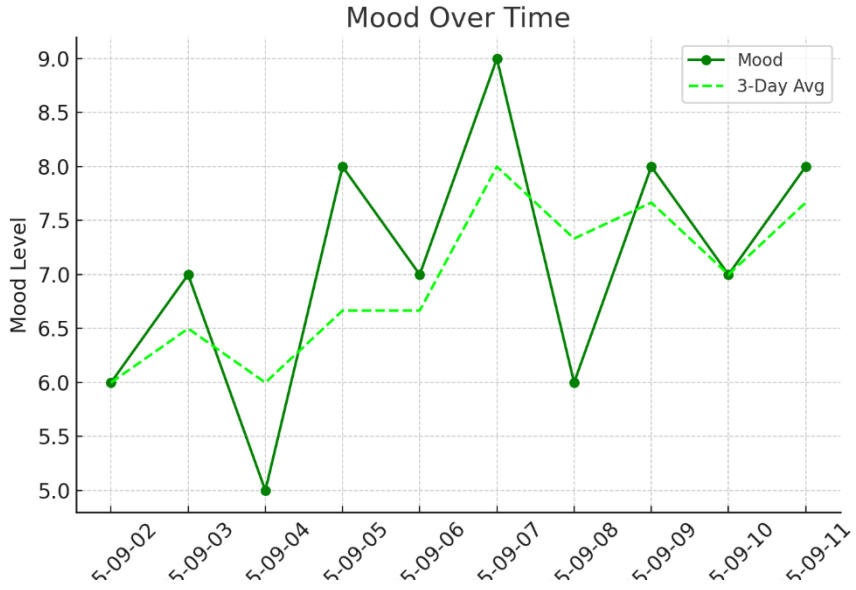
Enter your mood (1-10): 8

Enter exercise minutes (0-1440): 45

Data logged and saved successfully!




Exercise Minutes Over Time
Exercise
3-Day Avg
Minutes
60
55
50
45
40
35
30
25
20
5-09-02
5-09-03
5-09-04
5-09-05
5-09-06
5-09-07
5-09-08
5-09-09
5-09-10
5-09-11


Mood Over Time
Mood
3-Day Avg
Mood Level
9.0
8.5
8.0
7.5
7.0
6.5
6.0
5.5
5.0
5-09-02
5-09-03
5-09-04
5-09-05
5-09-06
5-09-07
5-09-08
5-09-09
5-09-10
5-09-11