

STROKE DATA ANALYTICS

IMPLEMENTATION REPORT

ABSTRACT

This project proposes the creation and design of a modular Python environment for stroke dataset analysis. The platform has three main modules: data manipulation (`dataset_module`), query operations (`query_module`), and an interactive user interface (`ui_module`) augmented by a Streamlit-based graphical overlay.

The primary objective is to enable extensive exploration and statistical comparison of stroke-related factors and to offer both tabular output and interactive visualization.

The report provides the context for the problem, system requirements, implementation strategy, execution procedures, and personal commentary on the process of development, emphasizing successes, challenges, and future areas for improvement.

Table of Contents

01	Introduction
02	Problem Analysis
03	Solution Requirements
04	Implementation Of Solution
05	Program Execution
06	Program Structure Flowchart
07	Reflection
08	References
09	Appendix

Introduction

The increasing need for analysis of healthcare data motivated the development of a special Stroke Analysis System capable of handling big datasets without depending on high-level libraries like Pandas or NumPy. The system was programmed to read, process, and analyze patient information in CSV format, extracting useful insights using a defined set of statistical queries.

Divided into three basic Python modules — dataset, query, and ui, the project is made maintainable and modular. The system also includes a (main) Jupyter Notebook to facilitate easier integration and user execution. For easier accessibility, a Streamlit-based graphical user interface was implemented, allowing users to utilize the system through a web browser.

The following subsections describe in detail the problem analysis, solution approach, technical design, implementation problems, and experiences from the entire development process.

Problem Analysis

The primary challenge thrown to this project was to design an effective system to process a dataset of strokes and answer provided medical and demographic queries. The CSV dataset had fields like Age, Gender, Smoking Status, Hypertension, Heart Disease, and Stroke Occurrence, among others.

The system needed to:

- Load and process the dataset correctly, dealing with missing, inconsistent, or mixed-type record entries without using libraries like Pandas.
- Run a sequence of predefined queries, from computation of modes, means, medians, descriptive statistics, cross-condition group comparisons.
- Save analysis results as external CSV files to facilitate documentation and reproducibility.
- Provide a human-interpretable interface through which users can select and execute queries interactively with the added facility for visualization by web-based software.

Major technical challenges were:

- Handling coercion of data types manually (e.g., strings to ints/floats) and detection of bad/missing values.
- Applying manual computation to statistical measures such as standard deviation and percentiles.
- Developing a minimalist and user-friendly user interface without resorting to standard data-handling libraries.
- Ensuring modularity, robust error handling, and immunity to incomplete or faulty data inputs.

In light of these constraints, the solution emphasized modular programming techniques, comprehensive error checking, and unimpeded user experience through both command-line and GUI-based interfaces.

Solution Requirements

1) Functional Requirements

a) Data Loading:

- Import CSV files and load data into lists of dictionaries with the original headers.
- Map data values to appropriate types (integers, floats, or strings) as much as possible.
- Handle errors such as missing files or corrupted lines gracefully to maintain system stability.

b) Query Processing:

- Provide eleven diverse queries for age distributions, frequency of heart disease, correlation of strokes with various factors, and descriptive statistics for particular features.
- Manually filter and select data subsets based on a query using calculations like mean, median, mode, standard deviation, and percentiles.
- Export query results as new CSV files in proper format.

c) User Interface:

- Implement a text-based and graphical (Streamlit) user interface where users can select, execute, and view queries interactively.
- Offer an explicit workflow with the option to execute multiple queries or exit gracefully.
- Display results in both textual and visual form, making it easier to interpret.

d) Integration

- Streamline the dataset handling, querying, and interface modules together via a main.ipynb notebook focused to facilitate easy project running.

Solution Requirements

2) Non-Functional Requirements

a) Modularity:

- Separate responsibilities into independent modules for maintainability and readability.

b) Robustness:

- Use comprehensive error detection and handling to avoid program crashes from file issues or invalid user inputs.

c) Usability:

- Design the system to be user-experience-focused, with interfaces that are easy to navigate and results that are easy to interpret.

d) Adherence to Constraints:

- Avoid the use of high-level data libraries (NumPy, Pandas) for core functionality, with allowances made for GUI visualization libraries like Plotly and Streamlit.

e) Extensibility:

- Allow the system to be extended in the future, e.g., through the addition of new queries or improving visualizations.

f) Documentation:

- Maintain clear and concise code comments, and include a complete technical report of the system design and operation.

Implementation Of Solution

The system design is composed of three independent Python modules and a Jupyter Notebook orchestration file. The architecture of each module aims at a specific segment of the stroke dataset analysis process to ensure modularity, readability, and ease of maintenance.

1) dataset_module.py

a) Purpose:

- This module is tasked with loading and preprocessing the stroke dataset from a CSV file in memory.

b) Key Features:

i) Data Type Conversion:

- There was a helper function (`_convert_to_appropriate_type`) where string inputs were being converted into integers, floats, or left as strings when appropriate. Missing or invalid values such as 'N/A' are mapped to None.

ii) Data Loading:

- The `load_data(filepath)` function reads in the CSV file line by line, manually parsing it into records and headers.
- Each record is stored as a dictionary where feature names map to values.

iii) Design Decisions:

- Parsing was done manually to comply with project requirements that restrict the use of libraries such as `csv`.
- The structure supports the management of records containing malformed columns or leading/trailing whitespace.

iv) Error Handling:

- It properly raises clear exceptions for problems such as lost files or data structure inconsistencies, so that the user can be provided with useful feedback.

Implementation Of Solution

2) query_module.py

a) Purpose:

- This module runs all statistical queries on the data loaded and exports the results.

b) Key Features:

i) Helper Functions:

A number of internal functions were created to support main operations:

- `_filter_data` filters records based on some conditions.
- `_get_numeric_values` extracts numerical features of the dataset.
- Functions to calculate mean, median, mode, standard deviation, and percentiles were manually implemented to avoid using external libraries.

ii) Query Functions:

The module contains individual functions for each of the eleven provided queries. Some of these are:

- Identifying patients with both heart disease and stroke.
- Calculating descriptive statistics (mean, standard deviation, minimum, maximum, and quartiles) for any chosen feature.
- Comparing the average sleeping hours for both stroke and non-stroke patients.

iii) Saving Results:

- Output from every query is saved as a well-formatted CSV file. The system formats numeric results nicely for readability and edge cases (like missing data) nicely.

iv) Error Handling:

- Invalid feature choices and missing data are trapped early, and there are human-readable error messages on screen and in output files

Implementation Of Solution

3) ui_module.py

a) Purpose:

- This module constructs an interactive Streamlit-based graphical user interface for the system.

b) Key Features:

i) Sidebar Navigation:

- Users interact with a sidebar containing all the available queries. Upon selection of a query, results are displayed dynamically.

ii) Result Presentation:

- Results are displayed elegantly with markdown, and charts are generated using Plotly, with histograms and bar charts for corresponding queries.
- Streamlit was chosen because of its simplicity and ability to create professional-level web applications rapidly. Custom color schemes, layouts, and font styles were employed to enhance user experience.

iii) Error Management:

- Loading errors, query failures, and other exceptions are trapped and displayed elegantly in the GUI, avoiding confusion and providing a smooth user experience.

iv) Extension:

- The GUI effectively replaces the command-line interaction, providing an intuitive and visually engaging way of navigating the stroke dataset.

Implementation Of Solution

4) main.ipynb

a) Purpose:

- The notebook serves as the primary entry point to run the entire system.

b) Key Features:

i) Dependency Management:

- It manages the installation of required packages (streamlit, plotly) through pip commands within notebook cells.

ii) Launching the GUI:

- Utilizing Python's subprocess module, the notebook triggers the Streamlit application, which automatically opens the graphical interface.

iii) Design Considerations:

- The choice to use a Jupyter Notebook was guided by ease of demonstration and adherence to instructions, allowing users who are not comfortable with command-line operations to run the project without a hitch.

Program Execution

In order to properly run the Stroke Data Analysis System, the following should be accomplished:

1) Prerequisites

a) Python Installation:

- Ensure that Python 3.12 (or an appropriate version) is installed on the system.

b) Package Installation:

Install required packages using pip:

- `pip install streamlit`
- `pip install plotly`
- `pip install notebook`

c) Dataset Placement:

- Place the stroke dataset file (`data.csv`) in the project directory alongside the Python modules.

2) Project Structure

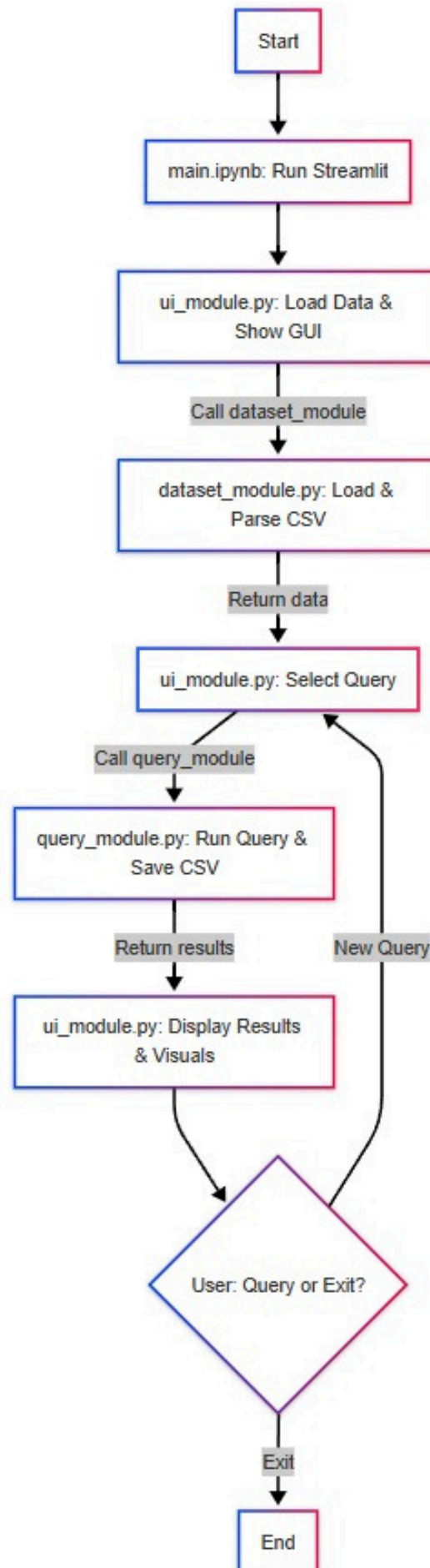
The directory should be organized as follows:

- `data.csv`
- `dataset_module.py`
- `query_module.py`
- `ui_module.py`
- `main.ipynb`

3) Running the System

- Open `main.ipynb` in Jupyter Notebook.
- Modify the `file_path` variable in the notebook if necessary to properly point to `ui_module.py`.
- Execute all cells in the notebook:
- Choose select queries, view results, and visualize data through the sidebar.
- All results will automatically be downloaded as CSV files to the same project directory.

Program Structure Flowchart



Reflection

1) What Went Well

a) Modular System Design:

- Function separation into three modules improved maintainability as well as the effectiveness of testing.

b) Graphical User Interface:

- Integration of Streamlit improved usability significantly by providing easy, graphical access to complex data.

c) Robust Error Handling:

- Effective handling of anticipated errors, such as missing values or incorrect input, kept the system in a stable and user-friendly state.

d) Library Compliance:

- Despite restrictions on the use of high-level libraries for data processing, all the project requirements were fulfilled.

2) Challenges and Difficulties

a) Manual Statistical Calculations:

- Without access to libraries like NumPy, manual standard deviation, median, and percentile calculations took a lot of time and utmost verification.

b) CSV Parsing Difficulty:

- Manual parsing of CSV files, particularly handling edge cases like empty columns or special characters, proved more difficult than initially anticipated.

c) Streamlit Learning Curve:

- Streamlit itself is easy to work with, but creating custom layouts and plots for a polished finish took more investigation and trial and error.

Reflection

3) Lessons Learned

a) Modularity Importance:

- Modular designing of the system made debugging and future extension far easier.

b) Extensive Error Handling:

- Anticipating a wide range of errors during system development made the ultimate system more stable and user-friendly.

c) Learning New GUI Development Skills:

- Developing an interactive Streamlit application expanded my technical repertoire to web-based data visualization.

d) Better Time Management is Crucial:

- GUI construction and calculations manually took longer than expected, which emphasized the need for better task estimation for future projects.

4) Future Work Improvements

a) Automated Testing:

- Adding unit tests would also test statistical computation and ensure long-term maintainability.

b) Enhanced Visualizations:

- The addition of boxplots, pie charts, and user-specified visualizations would potentially provide greater insights.

c) Enhanced Parsing:

- If allowed, utilizing the Python csv library would enhance parsing for better efficiency and fewer errors.

d) Greater User Control:

- Future releases might allow the user to enter dynamic conditions for customized queries.

References

- Streamlit Documentation: <https://docs.streamlit.io/>
- Plotly Documentation: <https://plotly.com/python/>
- Python Official Documentation: <https://docs.python.org/3/>
- Assignment Brief
- w3schools: <https://www.w3schools.com/python/>
- geeksforgeeks: <https://www.geeksforgeeks.org/python-programming-language-tutorial/>

Appendix

1) Pseudocode:

a) dataset_module.load_data

```
FUNCTION load_data(filepath)
  TRY
    OPEN file at given filepath
    READ the first line to extract headers
    SPLIT headers by commas
    INITIALIZE empty list for data records
    FOR each subsequent line in file:
      IF line is empty, CONTINUE
      SPLIT line by commas
      IF number of columns matches header:
        CREATE dictionary for the record
        FOR each (column, value) pair:
          CONVERT value to appropriate type
          ADD to record
        APPEND record to data list
      RETURN data list and header list
    CATCH FileNotFoundError
      RAISE file error
    CATCH other exceptions
      RAISE parsing error
  END FUNCTION
```

Appendix

1) Pseudocode:

b) query_module.query_descriptive_statistics

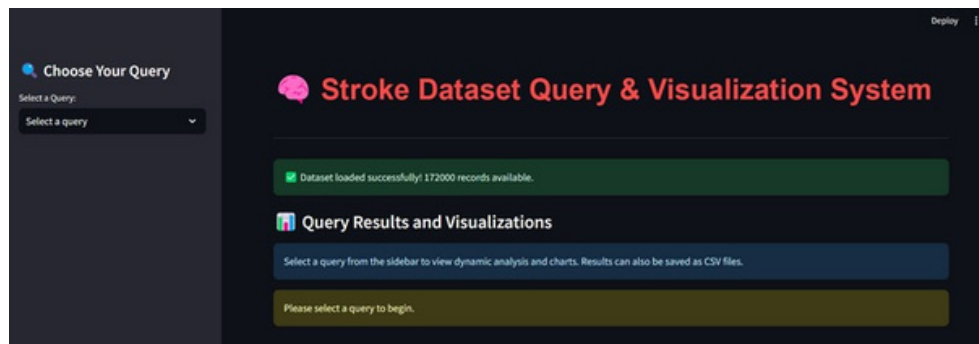
```
FUNCTION query_descriptive_statistics(data, feature_name, header)
  IF feature_name not in header:
    RETURN error
  EXTRACT numeric values for feature
  IF no numeric values found:
    SAVE error to CSV
    RETURN error
  CALCULATE mean, standard deviation, min, max, percentiles (25%, 50%,
75%)
  STORE results in dictionary
  SAVE results to CSV
  RETURN result dictionary
END FUNCTION
```

c) ui_module.run_streamlit_app:

```
FUNCTION run_streamlit_app(filepath)
  SET Streamlit page configuration
  DISPLAY project title and description
  TRY
    LOAD dataset
    IF loading fails:
      DISPLAY error
  DISPLAY sidebar with list of available queries
  WAIT for user query selection
  IF query selected:
    EXECUTE corresponding query
    DISPLAY results and visualization
  CATCH query execution errors:
    DISPLAY appropriate error messages
END FUNCTION
```

Appendix

2) Output Samples:



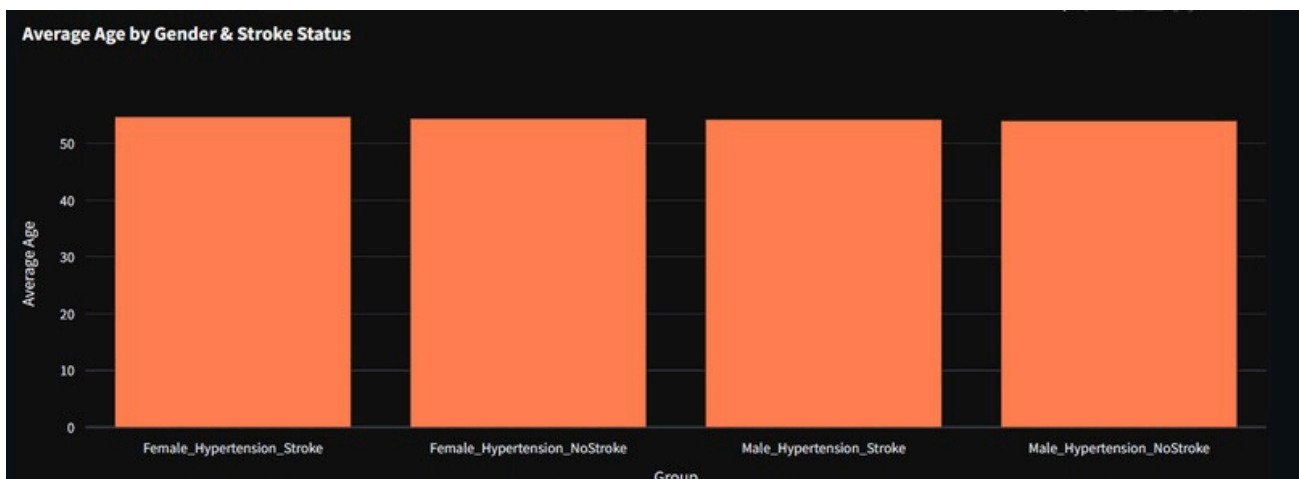
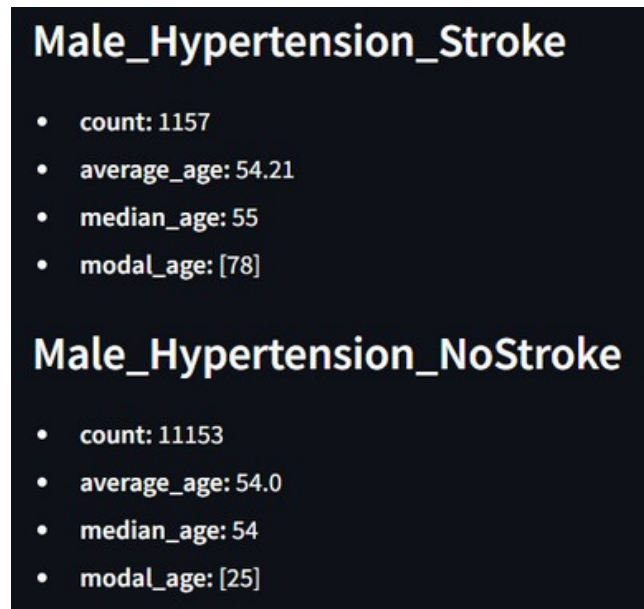
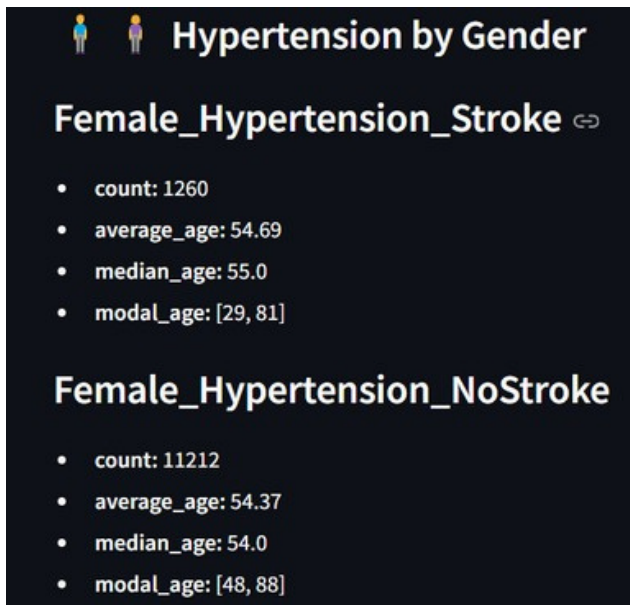
a) Home Page



b) Smoker with Hypertension

Appendix



2) Output Samples:



c) Hypertension by Gender

Appendix

2) Output Samples:

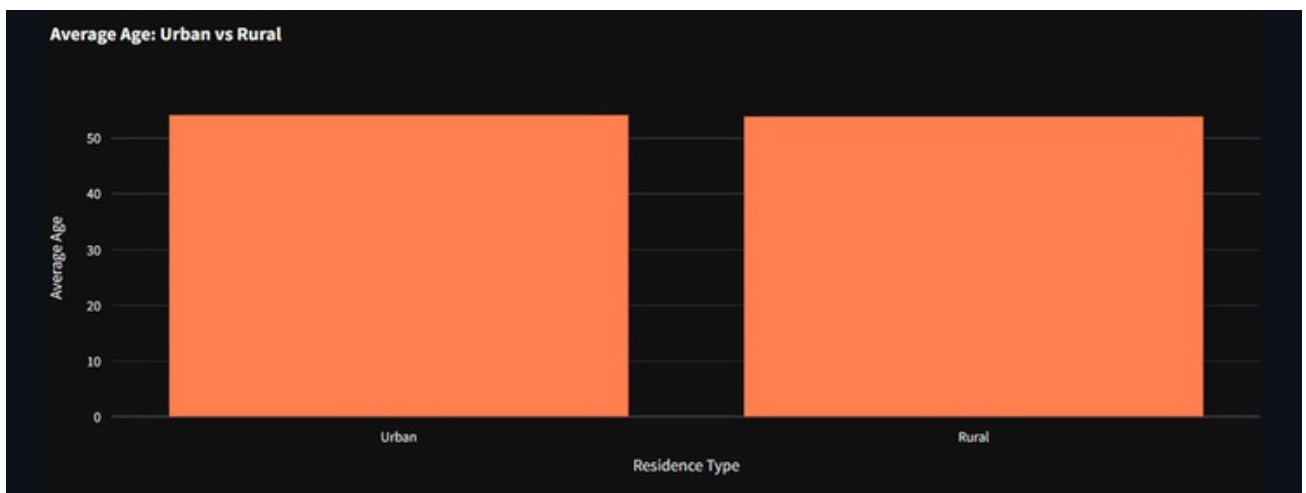
  **Urban vs Rural Stroke**

Urban

- **description:** Urban residents with stroke
- **count:** 8484
- **average_age:** 54.19
- **median_age:** 54.0
- **modal_age:** [29]

Rural

- **description:** Rural residents with stroke
- **count:** 8612
- **average_age:** 53.92
- **median_age:** 54.0
- **modal_age:** [26]



d) Urban vs Rural Stroke