CSIT998 - Professional Capstone Project

November 2, 2025

Health Trends Analytics for Older People: Automated Data-Driven Visualization

(Project Instructor: Prof. Ping Yu)

Group 11: Analytic Top Gangs

YING KIT JERRY, LI, 8469180 (ykjl975@uowmail.edu.au)

CHENKUN, YE, 8376141 (cy981@uowmail.edu.au)

YI OUYANG, 8773397 (yo343@uowmail.edu.au)

SHASHA, YOU, 8834428 (sy255@uowmail.edu.au)

HIU FUNG, WONG, 8558796 (hfw890@uowmail.edu.au)

HO, NG, 8527507 (hn951@uowmail.edu.au)

Abstract

Project Overview
This project tackles several primary challenges: the under-quality and less-integrated nature of dementia records, the limited availability of effective analysis tools for tracking health trends in aged care facilities, and the lack of intervention guidelines and actionable information to inform nursing practices.

Problem Statement
This project tackles several primary challenges: the under-quality and less-integrated nature of dementia records, the limited availability of effective analysis tools for tracking health trends in aged care facilities, and the lack of intervention guidelines and actionable information to inform nursing practices.

Project Methods
Through developing a Power BI-based health trend visualisation platform via a disciplined process of intense data cleaning and preprocessing, extensive exploratory data analysis, comprehensive feature engineering using TruncatedSVD dimensionality reduction, K-Means clustering for symptom pattern identification, and training of ensembles of classification models that were optimized for the identification of high-risk patients, in a modular React-FastAPI architecture.

Project Deliverables and Contributions
The designed system identified two distinguished patient care clusters (Mobility-Dominant and Pain-Dominant profiles). It achieved 75.9% F1 score with 73.2% precision and 82.9% ACU when predicting high nutritional risk with a conservative weighted SVM model, and it presented an interactive Power BI dashboard with eight integrated analytical modules to monitor multi-level health. Through this project, it transforms fragmented aged care data into clinical information that enables proactive risk detection, evidence-based intervention recommendations.

Contents

1 Introduction

1.1 Background Information

According to the WHO report from 2019, global aging is causing more and more cases of dementia. This condition affects thinking, memory, and daily activities. WHO estimates indicate that about 57 million people will have dementia in the upcoming years, with roughly 10 million new cases each year. One of the best examples is in the US, research by Saliba et al. (2023) revealed that over 3 million nursing home residents were diagnosed with chronic disease, such as related dementias, between 2017 and 2019. This figure is expected to increase as the population ages, as dementia rates increase. Dementia has a large associated economic burden and causes many deaths among older adults and their caregivers. These trends render dementia a significant systems design and public health concern beyond medical treatment.

With an aged-care society, especially in dementia services, information can often become fragmented across different interactions, tests, medications, and patient-reported symptoms from healthcare or community systems. Converting fragments into dynamic, usable information embedded in regular operations and clinical triage is a challenge. Ludlow et al. (2021) looked over that resident important health information can exist in separate domains and poorly integrated. This problem causes aged-care systems to "get rich data but fail to create useful information", which impacts the extent to which decisions can be evidence-based as well as coordinated.

Considering the above-mentioned phenomena, creating a more tangible and user-friendly dashboard of these separate data sets and a common display for clinicians and administrators is a significant milestone. The dashboard indicates current risks, workload, and other quality indicators, as well as limits opportunities for human error and issues with reconciliation. A literature review of hospital dashboards suggests that they improve responsive decision-making when used with current data and workflows. However, focusing on usability, integration, and data freshness is key to avoiding unexpected problems. Meanwhile, machine learning methods often integrate with tabular health data. Evaluating these methods is effective for risk assessment. Unsupervised clustering helps group symptoms into valid phenotypes for triage and follow-up. Together, dashboards and machine learning can reduce variation, identify high-risk patients earlier, and support cost-aware care planning (Coiera et al., 2025; Javeed et al., 2023).

1.2 Problem Statement

Data Quality and Missing Value
Low data quality and missing information in dementia EHRs negatively affect machine-learning analysis and visualization. Since EHR dementia phenotypes differ significantly across sites, such as different code sets, reference standards, and validation methods, the definition of "a dementia case" varies from system to system. This creates label noise, which reduces the accuracy of apparent prevalence, undermines feature importance, and misleads classifiers regarding the correct target. On dashboards, it produces distorted trends and case-mix comparisons Walling et al. (2023).

Limited Effective Tools of Health Trends in Aged Care
While acute care hospitals have widely adopted advanced data analytics platforms and clinical dashboards to support evidence-based practice, many hospital rehabilitation and day healthcare centres often lack integrated, near-real-time views of patient status and service activity in their regular routines. Even where dashboards exist, their effectiveness relies on the quality of the data, its timeliness, and how well they in-

tegrate into workflows. A 2025 systematic review Coiera et al. (2025) of over 70 healthcare and medical centers' dashboard evaluations found benefits, such as effective care and shorter lengths of stay, only when dashboards were supplied with timely, integrated data and aligned with clinical workflows; otherwise, the results were inconsistent or neutral. These delays and inconsistencies hinder trend analysis and reduce trust in the displays.

Data-driven Nursing Intervention Recommendations
Current systems often produce incomplete, data-oriented nursing prompts. They depend on fragmented, variably coded inputs and assess different outcomes. These systems barely include cost or workflow context. Therefore, some recommendations can become outdated, difficult to implement, and not clear. This can lead to alert fatigue and a lack of trust in the recommendation system. Additionally, a lack of data and biased EHR data may lead to miscalibration of risk flags for certain groups. Insufficient explainability of the recommendation presented as compared to a nursing routine, therefore some prompts might be overlooked or over-applied, resulting in both under- and over-triage as well as relative errors in medications or monitoring. Further, this leads to mixed escalation pathways in patient risk management. Lastly, dashboards usually provide descriptive reporting instead of offering actionable step-level direction or recommendations when appropriate. Dashboards can create a larger divide between data collection and care decisions and squander precious nursing time. Improving provenance, calibration, and human-centred design is needed for safe and effective recommendations for intervention (Lampe et al., 2024).

1.3 Project Objectives

The main objective of our system is to generate a health trend visualisation platform powered by automated Power-BI model with Machine learning. It aims to identify significant gaps in aged care data analysis. This platform will address the challenges of tracking health trends, identifying high-risk patients, recommending interventions, and transforming data into actionable insights for dementia care in residential aged care facilities. Additionally, users can upload their files in a specific format, which may improve database features and boost the performance of machine learning models. Therefore, there are several Highlights that should be included in our finalisation.

- **Data Integration and Preparation** (a unified data foundation that is able to collect all scattered health information into a structured format for follow-up selection and analysis)
- **Comprehensive EDA and Pattern Discovery** (understand the characteristics, distributions, and relationships between different attributes, e.g. gender, age, BMI, m-risk and others)
- **Symptom Clustering and Pattern Recognition** (Identify different symptom groups, age distribution, bmi change and behaviour patterns to support personalized care methods.)
- **Machine Learning Model Development for Risk Classification** (build and validate machine learning models that can identify high-risk patients for health outcomes based on characterised and clustered symptom patterns and past data)
- **Interactive Dashboard Development with Power BI** (Design and create interactive dashboards to display population-level health trends and individual patient insights for healthcare staff)
- **Intervention Recommendation Mechanism** (Develop an evidence-based system that suggests suitable nursing interventions)

1.4 Scope and Limitation

The scope of this project is to develop an automated health trend visualisation platform specifically for dementia care within residential aged care facilities. The project encompasses a complete data pipeline, beginning with the integration and preparation of scattered health data into a unified, structured format. Key activities include conducting a comprehensive exploratory data analysis (EDA) to discover patterns, using clustering techniques to identify symptom groups, and building machine learning models to classify high-risk residents. A primary deliverable is an interactive Power BI dashboard designed to present population-level health trends and individual resident insights to healthcare staff. The system will also feature a mechanism for recommending evidence-based nursing interventions and will allow users to upload their own data files to insert new resident records for data visualisation on the Power BI platform.

However, there are several limitations, as the system is developed using a specific historical dataset, its predictions might not be as accurate for different resident groups. The analysis is based on past data, meaning the dashboard can not be used for real-time decision making. Furthermore, the final deliverable does not include critical features needed for real-world use, such as industry-standard security or direct integration with live hospital records.

2 Literature Review

2.1 Data Dashboard and Visualisation

According to Dowding et al. (2015), an effective clinical dashboard provides healthcare staff with relevant and dynamic information to inform their daily decisions, thereby improving the quality of patient care. It enables easy access to multiple sources of data in a visual, concise, and usable format. The main function of clinical and quality dashboards is to provide summary data on performance measured against metrics (often related to quality of care or productivity) and use data visualisation techniques to provide feedback to healthcare leaders or individual clinicians. Meanwhile, Dowding et al. (2015) explains that the metrics for evaluating a successful dashboard visualisation should consider the colour coding with a tabular indicator list. For example, using a 'traffic light' (red- yellow- green) approach where green indicates no action needed and red indicates action is necessary, and the format of the bar or pie chart can be used to display as a whole visualisation, since such visibility is associated with better outcomes.

Regarding the Power BI integration, Rochin et al. (2021) highlights that the high-fidelity Power BI dashboard provides clear visualisations and thorough analysis by following established dashboard design guidelines and interaction standards. It organizes the main screen into seven distinct sections: resident personal information, incidents, behaviours, quarterly assessments, physical and cognitive performance, group limits, and prospective indicators panels. Users can drill down to observe data at various levels of detail. This approach makes sure the dashboard connects with decision points, such as resident-group assignment. Usability and acceptance testing, using a TAM-based questionnaire, indicated that the dashboard is very likely to be useful and fairly likely to be usable. This supports the effectiveness of this layout for day-centre dementia care. In our capstone project, there are some patterns of dashboard design that can be mirrored, such as the resident overview, high-risk symptoms analysis, and cohort limits etc, that can be introduced inside.

2.2 Machine Learning Model in Dementia Analysis and Predicted Caring

Machine Learning is a tool for early diagnosis, differential diagnosis, and prediction of onset/progression. Some authors analysed 60-75 studies that employed various ML algorithms in the field of dementia prediction and nursing (Javeed et al., 2023). Furthermore, the proposed model achieved a high accuracy (82.00% or higher). The researchers' suggested Support Vector Machine (SVM) and Random Forest (RF) Machine Learning techniques outperformed the rest of the ML algorithms in terms of performance. However, Javeed et al. (2023) also mentions problems such as poor data quality, inadequate selection of ML models, the bias-variance tradeoff, and training overly complex models. Therefore, challenges in model generalizability, interpretability, and clinical integration still have to be considered.

Although some machine learning models published in various journals are not entirely related to Dementia resident care and its high-risk symptom prediction, several models can be referred to, providing a baseline for leveraging these model functions and their accuracy in our project.

- Support Vector Machine (SVM): Identifies the hyperplane that maximizes the margin and, through kernels, is able to raise nonlinear data to higher dimensions. The kernel drives its performance, the regularizing parameter C, and γ (RBF), which, in combination, will affect the margin width and locality. Advantages include being effective in high-dimensional and small-sample situations, as well as solid margins. Disadvantages include sensitivity to class imbalance and hyperparameters, the absence of native probability outputs, and a lack of interpretability (Battineni et al., 2019; Javeed et al., 2023).

- XGBoost (XGB): Creates trees in sequential order to improve on the errors of previous trees, whereas XGBoost is a fast, regularised version developed for structured/tabular data. It is very useful for tuning accuracy, works well with mixed feature types, and can provide per-feature measures of contribution. However, it would be more sensitive to hyperparameter tuning than other models (e.g. Random Forest); replacing clear bias with overfitting potential if one is not validating model fit variability in data; and a little heavier to globally explain (Kavitha et al., 2022).

- Logistic Regression (LR): Describes the log-odds of a categorical result by constructing a linear combination of features, which allows for calibrated class probabilities to inform a simple and interpretable decision boundary. The advantages of LR include the transparency of feature effects and the ability to calibrate risk thresholds. On the other hand, LR assumes some linear separability (after applying transforms) and can result in underfitting relationships that are more complex and nonlinear, particularly in risk associated with symptoms in dementia trajectories, compared to tree ensembles (Velazquez et al., 2021).

- Random Forest (RF): Ensembles many decorrelated decision trees on bootstrapped samples, which reins in overfitting while modelling rich, non-linear interactions. RF naturally provides feature-importance profiles that can be interpreted at both the model and resident levels—clinically useful for discussing the reason a resident is flagged. RF was hypothesized as a high-accuracy ensemble for decision support and handled imbalance well after oversampling. Its Strengths are strong tabular performance, interaction capture, and explainability via importance. On the other hand, larger models can be less compact; global interpretability is not as fine-grained as that of other models, such as LR (Velazquez et al., 2021; Kavitha et al., 2022).

2.3 Gaps of Existing Literature

While there are many research papers focusing on machine learning to forecast upcoming health trend such as in dementia, and analyse healthcare status, some research gaps are still worth being concerned about.

- Aged Care Implementation in Reality: Many research papers rely on research or synthesis datasets, rather than actual data from aged care facilities. This approach can be more complex and challenging, as it increases the bias and uncertainty of the machine model.
- Behavioural Symptom Focus: Many studies on dementia and machine learning emphasise cognitive measurements and imaging. However, they pay less attention to behavioural and psychological symptoms, which are crucial in aged care.
- Integration of ML and Visualization. Few studies present comprehensive systems that integrate machine learning predictions with user-friendly dashboards for non-technical nursing staff.
- Intervention Recommendation: While researchers have extensively investigated risk prediction, they do not have much investigation on exploring machine learning-based recommendations for specific caring interventions.

3 Methodology

3.1 Methodology Overview

This project employs a defined data science lifecycle approach, transforming raw health data into actionable insights for aged care providers. The first step is data collection and preparation, which includes selecting the dataset for analysis, data quality assurance (cleaning the data), and exploratory data analysis (EDA) to help identify primary patterns in the data we have available. Next, feature engineering will occur to create new features from the raw data for input into the machine learning platform for predictive and risk modelling. The primary focus of the data analysis stage is on clustering to identify natural groupings of symptoms, as well as creating predictive models for risk classification. The last step of the methodology is the implementation of the system, including creating an interactive dashboard for the visualization and reporting of the data, developing and implementing interactions with the front end of the system, and deploying the system in a containerized environment for testing.

3.2 Data Source Selection and Exploration

The database used was from the University of Wollongong, which is important because it's important to start with. A historical dataset from residential aged care facilities was selected for this analysis. The dataset was chosen for its significant relevance to examining health trends in older individuals, in particular people with dementia. The data consists of long-term resident records, containing a range of relevant information, including age, gender, BMI, weight and a complete list of symptoms and risk factors. The detailed information is well-suited for observing meaningful patterns for the required Power BI dashboard visualisation. After selecting the data source, an initial exploration of the data occurred before analysis began. Primary exploration is a key initial step in developing a decent level of understanding of the nature, contents, and quality of the dataset. Initial exploration involved both checking basic statistics and spotting potential problems, particularly with missing data. Understanding the data before it can easily inform the data cleaning and preprocessing segment so that we know we are building a sound dashboard.

3.3 Data Cleaning and Preprocessing

This stage was crucial in the cleansing of the raw data, which tends to be accompanied by noise, inconsistencies, and gaps that may lead to erroneous analytical results. The process used to resolve these problems step by step guaranteed the data to be accurate, complete, and appropriate for high-level analysis and modelling, thereby increasing the validity of findings and predictions obtained from the rest of the subsequent stages in the process.

Manage Missing Values
A complete scan was conducted to identify missing observations on all columns, prioritizing key numeric fields. Data type dictated imputation tactics; e.g. missing values were filled in with logical inference, such as estimation of related measures from present correlated variables to prevent data loss. Where imputation has the possibility of injecting high bias (e.g. very high missingness over 20-30% in a single particular record), whole rows or groups were deleted after considering their impact on the balance and representativeness of the entire data. This stage not only preserved the statistical power of the set but also eliminated artifacts.

Standardise Data Formats
In order to have integration without any problems and analysis, all the different data formats were made the same. The parsing and standardizing of the temporal fields were done to a consistent datetime object format (e.g. YYYY-MM-DD), which made it easier to do chronological computations such as duration calculations. The categorical variables were cleaned by mapping different forms (e.g. abbreviations or misspellings) to the standard labels, and the unstructured textual data like lists of conditions, was tokenized and converted into binary flags for consistency. This standardization helped in reducing the errors during the aggregation and also made the analytical tools compatible.

Apply Filters for Proper Records
Custom filters were put in place to keep only the best and the most relevant records. Data coverage residents with low data points were excluded through threshold-based filtering, for instance, those who had fewer observations than a pre-defined minimum (e.g. equivalent to a few months of monitoring) so that the trend analysis was supported by sufficient longitudinal depth. Furthermore, heuristic rules were employed to find and discard erroneous records, such as those with contradictory patterns or values that lie outside the expected domain limits. Consequently, the dataset was directed to trustworthy subgroups and the reason for the exclusions was recorded to ensure transparency.

Handle Outlier Values and Duplications
Outliers were identified by flagging extreme values in numerical attributes using a combination of statistical techniques (such as Z-scores and interquartile range) and visual inspections (such as box plots). Context-specific treatment differed: implausible outliers (such as values that violated physical constraints) were either eliminated or winsorized (capped at percentiles), while plausible outliers were kept if domain-justified. To avoid inflated counts and guarantee data uniqueness, duplicates were methodically found by exact matching on key identifiers and timestamps, with resolutions giving priority to the most recent or complete entry.

Normalise the Values
Normalisation helped with comparability and model stability by converting variables to a common scale. In addition to aggregation techniques (e.g. computing means, medians, or percentages for time-series met-

rics), min-max scaling and Z-score normalisation were used when necessary for numerical data. One-hot encoding or presence/absence flagging was used to normalise binary and categorical data, establishing a consistent 0–1 range. In order to handle skewed distributions and get the data ready for algorithms that are sensitive to scale differences, this step was essential.

Symptom Group in Care Domains
To facilitate consistent analysis and nurse-friendly displays, the symptoms were clustered into high-level care areas, making use of the Australian SNOMED-CT ontology as a guide to overcome near-synonyms prior to pooling. The pooled items were coded against ten nurse-friendly care domains (e.g. Nutrition / Hydration, Mobility / Falls, Skin Wound). At this stage, the SNOMED concept IDs were not stored in the database; the coding was merely an in-memory pointer to the appropriate grouping and visualisation in the dashboard.

3.4 Comprehensive Exploratory Data Analysis

For better comprehension of the dataset's architecture, revelation of hidden structures, and guidance of hypothesis formation, EDA was a key first step. This phase identified prominent trends, imbalances, and interdependencies using descriptive statistics, transformations, and visualizations. This helped to facilitate better data preparation and feature selection for better modelling.

Exploring Data Characteristics
Prior to granular summaries, the overall data shape (rows, columns) and the types of data were profiled well. Distributions like the average, minimum, and maximum observations for an entity were revealed by summarizing record frequencies by identifiers. Percentiles (such as 25th, 50th, 75th) gave information on variability. Univariate statistics (means, standard deviations, and modes) were calculated for every variable, and class imbalances were quantified (such as ratios in danger categories). To identify potential biases or dominant patterns at an early stage, multivariate explorations identified correlations and co-occurrences, e.g. condition prevalences.

Performing Data Transformation
In order to make the data more useful and uncover underlying relationships, transformations were performed. Although new derived variables, e.g. composite scores of multiple inputs, were constructed, aggregation condensed time-series data into summaries at the patient level (e.g. locating maxima/minima or averaging metrics). Encoding changes over time, temporal data was differenced, and encoding techniques mapped categorical data to numerical representations (e.g. binary vectors for multi-label conditions). These operations conditioned it for visualization and model-building without distorting the data's inherent meaning.

Visualizing the Relationship of Data
Relationships within the data were shown intuitively with a set of visualizations. Although bivariate tools like scatter plots and heatmaps showed correlation (e.g. between measures and groups), univariate plots like density charts and histograms showed distributions (e.g. spreads by age). Multivariate visuals, like pie and grouped box plots, showed interactions (e.g. co-occurrences of conditions by groups). In addition to enabling pattern detection and outlier discovery, these plots enabled separating narratives of risk factor clusters or population trends, to facilitate data-driven decision making.

3.5 Feature Engineering

Feature engineering, with an emphasis on developing informative features that detect complex patterns, played a vital role in shaping raw data into modelling-ready format. To improve model performance, minimize noise, and address high dimensionality efficiently, this included creating, developing, and optimizing features.

Feature Creation
To capture essential insights, domain-specific characteristics were created, which included, for instance, drawing up timelines with different types of data (e.g. average values for different time periods, percentage changes like loss metrics) and creating composite indicators (e.g. risk scores from weighted combinations of base variables). Moreover, the introduction of binary flags to indicate condition presence allowed the representation of complex attributes. This activity not only produced the dataset but also added variables that directly related to the analytical objectives, like forecasting risk levels.

Feature Transformation
Transformations preserved and enhanced the features to make the data easier to interpret and more distributed. Among the few techniques used to modify the distribution of numeric variables, skews were one such case where logarithmic scaling or box-cox transformations were applied to bring them closer to normality. Categorical features were encoded into binary or ordinal encodings, and the time features were reorganised (e.g. by building elapsed time or season indicators). These processes eliminated more multicollinearity, as well as increased feature strength, so that they could be used directly with multiple modelling algorithms.

Feature Extraction
Advanced extraction procedures compressed high-dimensional data into compact representations. Truncated Singular Value Decomposition (SVD) was primarily used on sparse binary matrices to uncover underlying factors, thus dimensionality reduction with preservation of variance (e.g. 80-90% with fewer factors). Furthermore, Principal Component Analysis (PCA) was also explored and used as an alternative for dense numeric data, finding orthogonal components contributing the most to variance explanation. A two-faceted approach allowed one to deal with different data sparsities and structures in an agile manner and to pick out informative patterns, including co-occurrences of conditions, among others.

Feature Selection
The shared selection process winnowed the features based on their relevance and redundancy. Univariate tests (e.g. variance thresholds) eliminated variables with little information and multivariate tools like correlation analysis and recursive feature elimination brought out the ones with great predictive potential. SVD/PCA loadings were important in choosing the most important components in such a manner that the top set remained brief but informative, thereby lowering the risk of overfitting and reducing the need for computation.

Feature Scaling
In order not to allow scale differences to affect the performance of models, all features were scaled in the same way. Standardization (e.g. through StandardScaler) adjusted the data so that its mean was zero and its variance was one, which is desirable for algorithms like SVM or clustering. Min-max scaling was applied wherever tightly controlled ranges were needed, for example, in the case of neural networks. This normalisation made sure that all the features played a fair role, especially in methods based on distance.

3.6 Data Clustering

Clustering was used to identify distinct resident groups based on transformed features, uncovering underlying phenotypes for risk assessment.

Clustering Process
Multiple clustering algorithms (e.g. K-Means, Agglomerative Clustering, DBSCAN, and Gaussian Mixture Models) were tested on SVD-reduced data. The optimal number of clusters was determined using metrics like Silhouette Score, Calinski-Harabasz Index, and Davies-Bouldin Index. The best algorithm and cluster count were selected, with clusters interpreted based on dominant features and clinical phenotypes.

3.7 Model Selection Strategy

To choose classification models for high-risk resident prediction, a focused strategy was used, with an emphasis on managing class imbalance and maximizing recall for the minority class. Class weighting was used to assess models like Support Vector Machines (SVM), Random Forest (RF), Gradient Boosting (GB), and Logistic Regression (LR). Different intensities of oversampling techniques (such as synthetic sample generation) were tested. AUC, precision, recall, and F1-score were used to rank performance, and the model with the best balance of sensitivity and specificity was selected.

3.8 Model Training and Validation

To maintain class distributions, models were trained on stratified train-test splits. Generalization was evaluated using cross-validation (e.g. Stratified K-Fold). For high-risk detection, threshold optimisation improved recall. Through fixed random seeds, reproducibility was ensured by saving the top-performing model along with related scalers and metadata.

3.9 Model Evaluation and Visualisation

Metrics specific to class imbalance, such as AUC for discriminative power, precision, recall, and F1-score for the high-risk class, were used to assess the models. Cross-validation was used to ensure that the estimates were reliable. Confusion matrices revealed error patterns, feature importance plots improved interpretability, and ROC and precision-recall curves demonstrated performance trade-offs.

3.10 System Design and Development

Frontend
The frontend is the part of a website or web app that people actually use and see. It is everything the user interacts with directly, such as layout, text, images, buttons, and other elements. Frontend development utilise programming languages such as JavaScript, Hypertext Markup Language (HTML), and Cascading Style Sheets (CSS) to create websites. Every language has a distinct function. HTML sets up the site's structure and content, CSS makes design elements better, and JavaScript adds advanced interactive features.(Simmons, 2022). For this project, A modern JavaScript framework such as React or Vue.js would be used to make an interactive application that works on all devices. This interface would also allow users to upload new data files, browse resident lists, and interact with the Power BI dashboard.

Backend and REST API
According to Simmons (2022), the backend of the application is the server-side that executes the logic, data processing, and communication. A REST API will be constructed using a python framework named FastAPI. REST, or Representational State Transfer, is a style of software architecture for designing web services to simplify and allow communication between the front end and the backend of the application. REST is a style that follows several guidelines and principles including a clean separation between the client (frontend), and the server (backend) while being stateless, which means that every request is considered a new request and is not dependent on previous requests.(Lokesh, 2019).

Modular Architecture
The system will be developed using modular architecture, which breaks a complex system into smaller, independent units or modules.(GeeksforGeeks, 2024). The modular approach has key advantages over the traditional monolithic architecture that relies on a single large codebase for all features. These include better scalability as team members can add or remove features without rebuilding the entire codebase. The modular system also offers better flexibility to customise and simpler maintenance procedures because debugging can be confined to separate self-contained modules. This architecture also leads to faster development cycles because it is feasible for different teams to be working on various modules simultaneously.(Clarke, 2023).

Integration
Various components are connected to form a unified system. The React-based frontend interacts with the Python backend through asynchronous HTTP requests to the REST API. This separation between the user interface (client) and the data storage and processing logic (server) represents a fundamental advantage of the modular and client-server architecture. Additionally, the interactive Power BI dashboard is integrated directly into the frontend application, enabling users to access comprehensive data visualisations within a single consolidated interface.

3.11 Deployment and Testing Strategy

Containerisation
The backend API and frontend web application are each packaged into lightweight, portable Docker containers. A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a standalone, executable package that includes everything needed to run an application, such as code, runtime, and system libraries. This approach ensures that the software will always run the same, regardless of the underlying infrastructure.(Docker, 2025).

Orchestration and Hosting
Docker Compose is used to define and manage multi-container applications, allowing the entire stack to be launched with a single command. The containerised application is designed to be deployed on a cloud platform such as Amazon Web Services (AWS), Google Cloud Platform (GCP), or Microsoft Azure, which enables scalable hosting and simplifies management.

Unit and Smoke Testing
Software testing is a process of verifying and validating that a software application is free of bugs, meets technical requirements, and satisfies user needs efficiently and effectively. Various testing techniques are employed to ensure the system is robust and reliable. Unit Testing is employed to initiate the test process,

in which individual units or modules of software are tested to find whether they are functioning as expected. This aligns well with the modular architecture, as each module has the ability to be tested independently. Smoke Testing is also performed on a new build to quickly validate whether its most critical functionalities are available, ensuring that it's sufficiently stable to perform more complex testing.(GeeksforGeeks, 2025).

System and Acceptance Testing
System Testing is then performed to verify the functionality and performance of the complete, fully integrated software application. The final step is Acceptance Testing, which is performed to determine if the software is properly functioning within the user environment and whether the users' requirements are met before its delivery.(GeeksforGeeks, 2025).

4 System Design and Architecture

4.1 System Architecture Overview

The structure is built upon a contemporary three-tier framework that clearly delineates the different levels for presentation, application logic and data management. The frontend is the place where React and TypeScript come in handy to build an extremely interactive and user-friendly interface that makes it possible for the users to upload data, check the patients and connect to Power BI dashboard. In addition, regarding the backend side, FastAPI gives RESTful services for API, data processing, machine learning intervention, and Power BI authentication by approching the service principal patterns. Structured data storage is done using a MySQL database in the data layer while an end-to-end pipeline is made for ETL processing of raw CSV inputs into analysis-ready datasets. Additionally, the deployment is done via Docker containerization which allows for the same application to run on different environments and Nginx reverse proxy is used to allocate incoming requests to the frontend and backend services. This type of architecture allows to scale the individual components separately, run ML tasks concurrently and add the visualisation, forecasting, and intervention advice mechanisms into a single healthcare analytics platform.

4.2 Data Pipeline Design

Data ingestion process
Data ingestion begins with importing CSV files, e.g. 'UpdatedDataFile.csv', using pandas.read_csv with arguments for UTF-8, comma delimiters, and missing data handling (e.g. considering empty strings or 'NULL' as NaN). The initial validation involves inspecting the DataFrame shape, column metadata, and count of unique records per PersonID for checking data integrity. The ingestion process relies on data import into pandas DataFrames without using external databases or real-time streams.

ETL (Extract, Transform, Load) workflow
The ETL process reads raw data from CSV files and groups it by PersonID for analysis on the resident level (see Machine Learning/Clustering and ML/1_data_analysis.ipynb and UpdatedDataFile.csv). Transformation includes cleaning by removing residents with fewer than 30 records or outlier patterns, imputation of missing values for BMI/Weight, calculation of height from BMI/Weight, derivation of Adjusted_BMI, and calculation of MUST scores (BMI score + weight loss score). M-Risk Factors' column risk factors are binary encoded into columns (e.g. one-hot encoding for states like "Pain", "Dry Skin"), and data is grouped by PersonID to calculate averages for weight and BMI, max weight loss percentage, MUST risk levels, and binary flags on risk factors. Date columns ('Start date', 'End date') are converted into datetime

in order to compute durations. The new data is then imported into new CSV files, namely 'UpdatedDataFile_aggregated.csv' and 'UpdatedDataFile_svd_transformed.csv', for further use.

4.3 Backend Design

Proxy-Based Architecture and Service Separation
The backend is a proxy-based microservice in which the frontend Nginx server acts as a reverse proxy to send requests to the FastAPI backend service through Docker's internal network. This enables the frontend and backend service to be separated with one entry point for all API traffic. The system employs a three-level routing pattern with different API prefixes: /api/ for general application operations, /cluster/ for machine learning clustering operations, and /risk/ for risk assessment services. It allows for separate functional domains to scale and deploy individually without displaying a heterogeneous 13 API surface to the frontend, with each proxy configuration involving common headers to preserve client information throughout the request lifetime.

Contract-First API Design
Frontend-backend integration uses a contract-first approach with TypeScript interfaces and centralized API client functions. The API client employs strong contracts with typed interfaces. Relative path routing is applied in the design whereby API BASE defaults to an empty string to allow Nginx to proxy requests transparently without hardcoding URLs. The client also provides generic apiGet and apiPost functions that use a single error handling approach, mapping all HTTP errors into JavaScript Error objects with composed error details.

Domain-Driven Endpoint Design
The backend maps frontend functional requirements into equivalent API endpoints using a domain-driven design approach. For data importing, the system employs a two-phase commit architecture in which the data is pre-validated and staged against a temporary token prior to actual commitment subject to user approval to prevent data corruption by error. For machine learning processing, the architecture utilizes concurrent request execution such that concurrent requests are executed concurrently, following a fault-tolerant parallel execution pattern in which one model can fail without impairing the other. The backend's integration with Microsoft Power BI follows an OAuth2 service principal pattern where the backend acts as a trusted intermediary, handling token acquisition and secure storage while shielding the frontend from complex authentication flows. The design implements an asynchronous polling pattern where the frontend triggers refresh operations through simplified endpoints and polls for completion status at regular intervals, accommodating Power BI's operations without blocking the user interface.

4.4 Visualisation Design

Dashboard Design Goals and Users
The Aged Care Health Trends Dashboard was built as an attempt to educate nurses, allied health professionals, and Australian residential aged-care workers. It is designed to respond quickly to situations that can arise in residential settings recognition of the population level health status of residents and to facilitate streamlined triage to individual case examination.

The system's architecture purposely reflects the workflow of clinical reasoning: the users first recall a general cohort description, then an examination of the study's subgroup profiles, drill down into the individual resident's longitudinal record. This sequence—of Population Overview $\rightarrow$ Group Profiles $\rightarrow$ Individual

Resident Review—encourages data-based insight, anticipatory risk identification, and more balanced workforce distribution within healthcare teams.

In keeping with best practices in data visualisation, the dashboard design aims at producing sharp, clear, accurate, and efficient visuals by focusing on clarity, accuracy, simplicity, and efficiency. Repeating visual encoding, standardised summaries of statistics, and natural interactions give clinical users the capability to read information at a glance and take action decisively.

Visual Design and Interaction Practice
The key data visualisation concepts were applied in designing the dashboard in order to enhance clarity, accuracy, and usability.

Clear chart type: The dashboard consisted of Donut charts of the shares of the categories (i.e., symptom These categories MUST have a distribution of scores; bar charts for comparison and frequency (Symptom Frequency, Age Distribution group); line plots of trends over time; monthly time–symptom density heatmaps; and resident timelines of individual longitudinal records.

Strategic colour use: Fixed colour semantics were used (Female = yellow, Male = green), and nutritional risk was carried out under a traffic-light system (light green = low, yellow = moderate, red = high). A consistent legends were preserved over all pages to prevent misinterpretation, with selection states offering higher contrast while maintaining usability.

Minimising clutter: Appropriate white space was applied as well as a consistent grid structure. The pages all centred around one or two seminal questions to pose, to sidestep redundant chart components and maintain the main view uncluttered and readable.

Showing key information: Showing essential information: Bottom KPI and statistic cards show the display of median, mean, CI, and IQR values in order to show central tendency and distribution spread clearly and quickly.

Providing context and interactivity: Global date and gender slicers are aligned across all pages. The Search Symptom function and related interactions (category $\rightarrow$ frequency $\rightarrow$ heatmap) create an ”overview-to-drill down” visualisation workflow. Clear metric delineations (in-care, admission, event, resident count) and uniform units allow drill-through to the Resident Profile in support of data-driven decisions.

Information Architecture Overview
The information architecture converts the preceding analytic narrative into a regular structural model. Dashboard is comprised of eight related modules displayed in a left-hand navigation bar: Resident Overview, Gender Distribution, Weight Distribution, BMI Profile, Nutritional Status, Nutrition Risk, Symptom Distribution, and Resident Profile. There is one logical dimension of resident health presented by each module while sharing the same grid of layout and slicer context.

Resident Overview: Produces an aggregate resident summary report with gender mix, BMI ranges, weight distribution, nutritional stability, and resident month admits.

Gender Breakdown: Displays male and female age distributions side by side through paired histograms with statistical cards (mean, median, 95%CI, and IQR).

Weight Distribution: Show mean weight by age group (line chart) and number of residents by weight group (stacked bar chart). Statistical cards below show the median, mean, 95%CI and IQR for clear comparison.

BMI Profile: Shows average BMI distribution, BMI trends by age and categorical composition (underweight $\rightarrow$ obese), linking physiological balance with demographic factors.

Nutrition Status: Displays residents' Mini Nutritional Assessment (MNA) scores and age-group distribution. Statistical cards report the median, mean, and IQR for transparent benchmarking.

Nutrition Risk: Visualises Malnutrition Universal Screening Tool (MUST) risk levels using traffic-light colours (light green = low, yellow = moderate, red = high). A resident-level table links risk status with resident ID, admission date, discharge date, and length of stay, bridging population analytics to personal-level details.

Symptom Distribution: Provides a multi-layered interactive view of residents' clinical and functional symptoms. A doughnut chart summarises symptom categories (e.g. Neuro-cognitive, Mobility/Falls, skin and Wound, Nutrition/ Hydration) and displays both the number of affected residents and the percentage contribution of each category to the total symptom records. The Symptom Frequency bar chart lists individual symptoms, displaying the number of recorded cases and the count of affected residents for each, enabling quick identification of prevalent or high-impact conditions. The monthly heat map presents the number of residents affected by each symptom over months, supporting temporal pattern analysis and detection of seasonal fluctuations. Clicking on any category of symptoms on the doughnut chart automatically filters both the frequency chart and the heat map, creating a fully linked analytic flow from category-level overview to symptom-level temporal trends. A Search Symptom box allows keyword-based lookup, improving navigation and usability for clinicians and data analysts

Resident Profile: The final module provides longitudinal tracking for a single resident, showing admission/discharge information, weight trend, symptom category composition, and a monthly heat map of symptom events. This micro-view supports aged care workers in monitoring residents' condition changes and informing daily care decisions.

4.5 Machine Learning Pipeline

Feature engineering pipeline

The very first step of the feature engineering pipeline is to take the collected data and to add different features like Gender, Age and over 460 risk factor columns that are represented as yes/no (see Machine Learning/Clustering and ML/2_data_engineering.ipynb and UpdatedDataFile_aggregated.csv). The binary risk factor matrix is then subjected to TruncatedSVD for the purpose of extracting 80 components which are responsible for the capture of about 87% of the variance, and among the contributors like "Pain" and "Dry Skin", the leading ones are highlighted (see Machine Learning/Clustering and ML/3_data_clustering.ipynb and UpdatedDataFile_aggregated_full.csv). Eventually, the whole feature set is made up of these SVD components plus Gender and Age, and is aimed at a binary MUSTScore (≥ 2 for high-risk). The class imbalance (which is a ratio of 1.8:1 low: high risk) is mitigated by the means of a well-planned oversampling, where artificial high-risk samples are produced with controlled noise in order to form balanced datasets.

Model training workflow
The training process consists of dividing the dataset by means of train_test_split to obtain a 30% test size with stratification (random_state=42) applied to both the original and the balanced datasets (see Machine Learning/Clustering and ML/4_model_selection.ipynb). The models that have been trained are weighted Logistic Regression, random Forest optimized for recall, and Gradient Boosting, and SVM weighted, all of which have been modified for the class imbalance by using class weighting. Furthermore, hyperparameter tuning is performed.

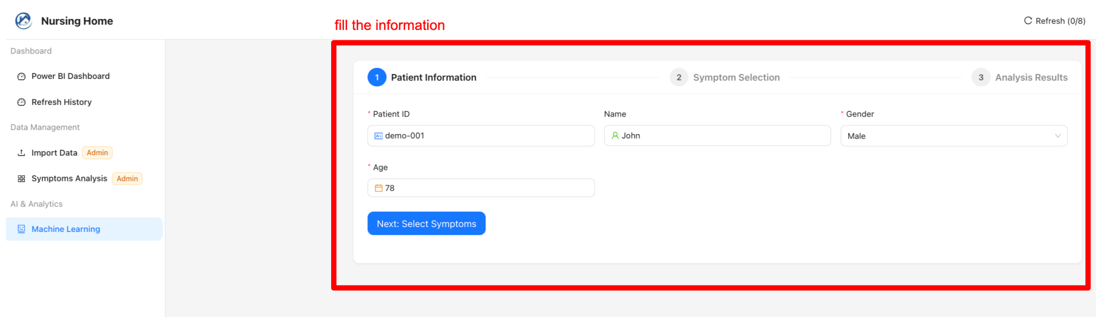
Model evaluation and selection process
Models are assessed according to the metrics of Precision, Recall, F1, and AUC for the high-risk class with the use of StratifiedKFold cross-validation. They are subjected to performance tests conducted on the original imbalanced dataset as well as on the balanced dataset, and recall levels (e.g. 0.70-0.85) are targeted for threshold optimisation. The ranking of models is determined by F1 and AUC. Validation is performed on a hold-out test set, where high recall for effective high-risk detection is the priority.

Deployment architecture for ML models
The models are serialized using the pickle module, forming all-inclusive packages that contain not only the model objects but also scalers (for instance, StandardScaler for Logistic Regression and Support Vector Machine), feature names, and metadata like the one for the resident predictor model as 'patient_predictor_model.pkl'. The inference arrangement consists of loading the package, applying the scaling if necessary, and making predictions through the predict_proba method with optimized thresholds. The package is also comprised of SVD transformers and clustering models (such as K-Means) for the phenotype analysis process. The design allows for stateless batch inference, which can be easily integrated into APIs but does not specify any need for real-time serving. The use of random seeds secures reproducibility, and the model's life cycle is recommended to include prospective validation and periodic retraining to monitor the model's performance.

4.6 Intervention Recommendation System

The intervention recommendation system is integrated in the Machine Learning module and operates according to an optimized three-step workflow aligned with clinical evaluation procedures.

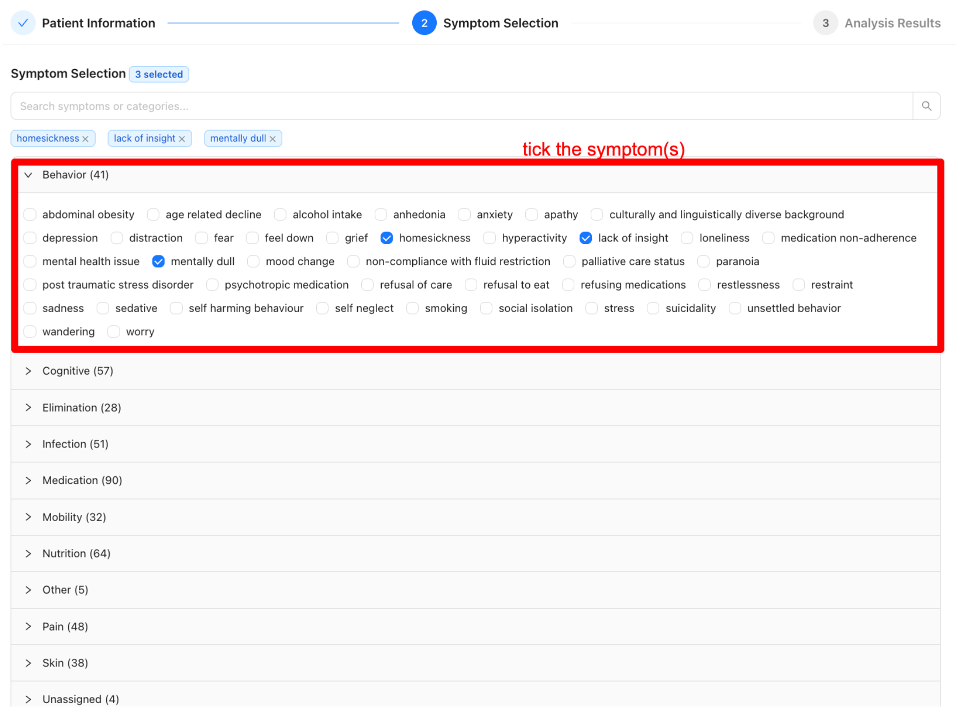
Resident Data Collection
Clinical staff provide key demographic data such as patient ID, name, gender, and age through a structured form interface. Demographic variables enable age- and gender-scaled risk evaluations based on population-specific nutritional risk patterns.



* Patient ID	Name	* Gender
demo-001	John	Male
* Age		
78		

Symptom Selection
Healthcare Staff can select suitable clinical symptoms from an extensive library of grouped symptoms or-

ganized into 11 categories (including: Behaviour, Cognitive, Elimination, Infection, Medication, Mobility, Nutrition, Pain, Skin, and Other categories). The symptom library has over 460 distinct indicators to enable comprehensive phenotyping at the clinical level. Meanwhile, new symptoms can be added to the library for an update. Indicated symptoms are mapped within the system to binary feature vectors consumed by the machine learning models.



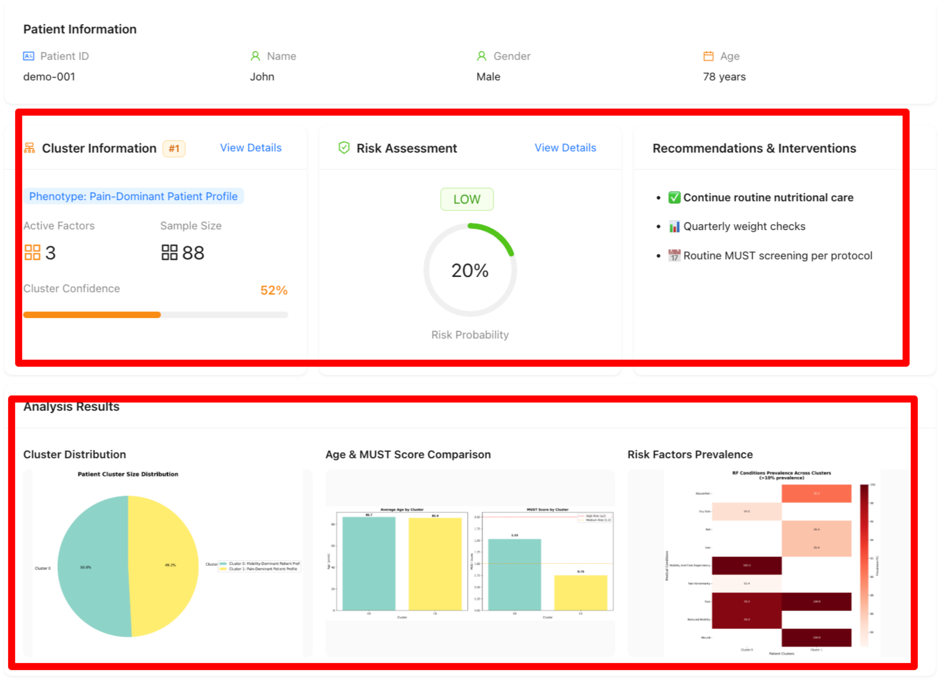
Behavior (41)

abdominal obesity · age related decline · alcohol intake · anhedonia · anxiety · apathy · culturally and linguistically diverse background
depression · distraction · fear · feel down · grief · [✓ homesickness] · hyperactivity · [✓ lack of insight] · loneliness · medication non-adherence
mental health issue · [✓ mentally dull] · mood change · non-compliance with fluid restriction · palliative care status · paranoia
post traumatic stress disorder · psychotropic medication · refusal of care · refusal to eat · refusing medications · restlessness · restraint
sadness · sedative · self harming behaviour · self neglect · smoking · social isolation · stress · suicidality · unsettled behavior
wandering · worry

Cognitive (57)
Elimination (28)
Infection (51)
Medication (90)
Mobility (32)
Nutrition (64)
Other (5)
Pain (48)
Skin (38)
Unassigned (4)

Automated Analysis and Recommendation Generation

After submission, the system does parallel clustering and risk prediction analysis. Results are displayed on an integrated three-panel presentation of:
(1) Cluster phenotype labelling with confidence indicators,
(2) Categorical risk level (Low/Medium/High) with numerical probability,
(3) Ranked clinical intervention recommendations. Visual aids consist of cluster distribution plots, bar plots of comparison between age and risk scores, and statistics for risk factor prevalence.


Patient Information

Patient ID	Name	Gender	Age
demo-001	John	Male	78 years

Cluster Information #1 View Details

Phenotype: Pain-Dominant Patient Profile

Active Factors	Sample Size
3	88

Cluster Confidence 52%

Risk Assessment View Details

LOW

20%

Risk Probability

Recommendations & Interventions

- ✅ Continue routine nutritional care
- 📊 Quarterly weight checks
- 📅 Routine MUST screening per protocol

Analysis Results

Recommendation Logical Framework

The recommendation engine employs a different-level rule system by risk severity:
Low Risk (≤ 40): Maintenance and ongoing monitoring is prioritized (e.g. standard nutritional protocols, quarterly assessment)
Medium Risk (40 - 60): Prioritizes preventative interventions and frequency of monitoring (e.g. review of diet, bi-weekly weighing, follow-up scheduling)
High Risk (> 60): Triggers urgent measures requiring immediate clinical attention (e.g. dietitian referral, aggressive supplementation, daily monitoring, root cause analysis)

Symptom-specific augmentation rules supplement base recommendations. For example, co-occurrence of pain symptoms initiates evaluation of the impact on appetite and meal adjustment, while mobility impairment initiates evaluation of the need for assistance with feeding. This conditional logic ensures that recommendations take into account the specific clinical circumstance rather than general risk scores alone.

5 System Implementation

5.1 Development Environment and Tools

The development environment relies on Docker for containerization, and the backend uses FastAPI as the Python web framework and python-dotenv to load environment variables from .env files during local development. Configuration validation is handled using typed Python dataclasses that ensure all the required credentials are present during startup.

The frontend is built with React and TypeScript, which allows for type safety throughout the application. The UI layer makes use of Ant Design components for styling and UI elements like buttons, tables, inputs, and tooltips. The frontend uses the dayjs library for date and time manipulation, particularly in the refresh history table where durations are calculated. The API traffic is handled by a custom client module that wraps the fetch native API with typed interfaces to provide type-safe request and response handling for all Power BI operations. Nginx serves as the reverse proxy and web server, proxying API requests to the backend service and implementing a 200MB client upload limit with 7-day static asset caching.

5.2 Data Handle Implementation

The dataset was pre-processed to ensure quality and consistency. Missing values were managed by inspecting for nulls using Pandas isnull().sum(), with no critical missing data found in columns like BMI and Weight. Implicit missing data (e.g. empty strings) was handled during CSV loading with na_values. Invalid derived metrics, such as non-positive heights calculated from BMI and Weight, were excluded. Date columns (‘Start date’, ‘End date’) were standardized to Pandas datetime objects using pd.to_datetime(). Risk factors in ‘M-Risk Factors’ were split and one-hot encoded into 460 binary columns (e.g. ‘RF_Pain’). Numerical columns (Age, BMI, Weight) were verified as appropriate data types. Filters excluded residents with fewer than 30 records, reducing the dataset from 20,197 to 19,564 records across 180 unique persons, and removed four anomalous PersonIDs. Outliers were addressed by recalculating BMI using derived height, with no duplicates found via duplicated(). Numerical features were prepared for later scaling, and the pre-processed data was saved as ‘UpdatedDataFile_preprocessed.csv’. Exploratory data analysis revealed 19,564 records, 180 residents, a mean of 101 records per resident, an equal gender split, an age range of 62–112 years (mean 86), and prevalent risk factors like Pain (99.4%) and Dry Skin (83.2%). Data was aggregated per resident using groupby, calculating duration, average Weight, adjusted BMI, and max weight loss percentage. MUST scores were derived based on BMI and weight loss thresholds, and the transformed data were saved as ‘UpdatedDataFile_aggregated.csv’. Feature engineering created features like Duration, AverageWeight, AvgAdjustedBMI, MaxWeightLossPercent, and MUST scores. TruncatedSVD reduced 460 binary risk factor features to 80 components, capturing 87.2% variance. Selected features included SVD components, Age, Gender, MUSTScore, and MaxWeightLossPercent. K-Means clustering on the SVD-transformed data identified two clusters: Complex Care (48.6%, high-risk conditions, older age) and General Elderly Care (51.4%, milder risks), with a silhouette score of 0.142. The model was saved as ’patient_classifier_model.pkl’.

5.3 Machine Learning Model Implementation

Model selection focused on high-risk patient detection (MUSTScore ≥ 2) using 82 features (Gender, Age, 80 SVD components). To address class imbalance (116 low-risk, 63 high-risk), three oversampling strategies were used: aggressive (doubling high-risk samples), moderate (no oversampling), and conservative (adding 14 synthetic samples). Four models were trained: Weighted Logistic Regression, Recall-Optimized Random Forest, Custom Gradient Boosting, and Weighted SVM, all with class weights to prioritize high-risk recall. An ensemble combined model probabilities weighted by recall. Training used a 70/30 stratified train-test split with 5-fold StratifiedKFold cross-validation. Hyperparameters were tuned for recall, and synthetic samples were generated with Gaussian noise. Threshold optimisation targeted recalls of 0.70–0.85. The best model, conservative_Weighted_SVM, was saved as ‘patient_predictor_model.pkl’ with StandardScaler and metadata.

5.4 Model Evaluation Implementation

The evaluation approach prioritized recall for the high-risk class (MUSTScore ≥ 2) to maximize identification of at-risk residents, with secondary metrics including precision, F1-score, and AUC-ROC. Models were assessed on both original imbalanced and balanced datasets using a 70/30 stratified train-test split and 5-fold StratifiedKFold cross-validation. Threshold optimisation was performed using precision-recall curves to target recall levels of 0.70, 0.75, 0.80, and 0.85 by adjusting decision thresholds. Visualizations, including precision-recall curves, ROC curves, confusion matrices, and classification reports, were generated using Matplotlib and Seaborn to analyse performance trade-offs. The ensemble model aggregated probabilities from base models, weighted by their recall performance, with an optimized threshold. The top-performing model was selected based on high-risk recall while maintaining acceptable precision, ensuring clinical suitability for screening.

5.5 Power BI Dashboard Implementation

System Architecture and Authentication
The solution employs a service principal-based authentication model where the backend is a trusted intermediary both for the application and for Power BI services. This architecture choice eliminates explicit user authentication with Power BI but retains secure access managed through the application's own authentication layer. The backend obtains OAuth2 access tokens from Azure Active Directory through client credentials flow with a default scope to request all permissions held by the service principal. The authentication process is encapsulated within the acquire_access_token() function that builds requests to Azure AD token endpoint based on tenant ID, client ID, and client secret held as environment variables. Each Power BI API call triggers a fresh token procurement rather than token caching, prioritizing correctness and simplicity over potential performance optimizations. This approach ensures tokens are always up to date when used with API calls, but at the expense of increased latency per operation.

Dashboard Embedding and Visualisation
The principal element is the internal Power BI dashboard, constructed utilizing an iframe to display a published Power BI report. The Power BI Embed component shows a full-screen iframe pointing to a Power BI view URL with an embedded report key. The embedded dashboard is exposed to all authenticated users by accessing the /dashboard route, which will be used as the default landing page upon login. The route configuration wraps the component within the SidebarLayout to enable consistent navigation and user interface controls across the application.

Dataset Refresh Operations
The five environment variables are required for the setup to be properly run: PBI_TENANT_ID for the Azure AD tenant ID, PBI_CLIENT_ID and PBI_CLIENT_SECRET for the service principal credentials, PBI_WORKSPACE_ID for the Power BI workspace containing the dataset, and PBI_DATASET_ID as a default dataset name. These are imported through a configuration module that verifies their presence and raises informative errors if any of the required values are missing.

Power BI Configuration and Deployement
The implementation requires five environment variables to function correctly: PBI_TENANT_ID for the Azure AD tenant identifier, PBI_CLIENT_ID and PBI_CLIENT_SECRET for service principal credentials, PBI_WORKSPACE_ID for the Power BI workspace containing the dataset, and PBI_DATASET_ID as a default dataset identifier. These variables are loaded through a configuration module that validates their presence and raises descriptive errors if any required values are missing.

It is deployed with Docker Compose, where backend and frontend services are executed in separate containers. Configuration for Power BI is offered as environment variables to the backend container, and the React app is exposed through the frontend container using nginx. Backends are prepared with health checks before frontend receives traffic, and Watchtower automatically updates containers when there are new images available.

Error Handling and User Feedback
The solution has robust error handling at various levels. Backend endpoints respond to network failures, token acquisition failures, and Power BI API failures by raising HTTP exceptions with appropriate status codes and informative error messages. Frontend components catch these exceptions and display messages based on state variables controlling banner appearance and content. When successful refresh operations occur, the interface displays a success message with the green checkmark icon and automatically conceal the message after three seconds. Failed operations show persistent error messages in orange, and timeout conditions inform users that status checking took longer than permitted. Such a multi-state feedback mechanism ensures users are always aware of what the status of their operations is, without needing to check themselves.

5.6 Backend API Integration and Implementation

Nginx Proxy Configuration and Routing Implementation
The backend API integration commences with the Nginx reverse proxy configuration in frontend/nginx.conf that proxies frontend requests to the backend service. The Nginx server listens on port 80 and serves static files from /usr/share/nginx/html with a client body size of 200MB to support large file uploads. Three independent location blocks are utilized to undertake API routing: the /api/ prefix leverages location ~/api/ to route generic application endpoints, /cluster/ for ML operations, and /risk/ for risk assessment requests.

All proxy configurations utilize proxy_pass http://backend:8000 to forward the request to the backend service using Docker's internal network, with standard proxy headers to preserve client information. The /healthz path is also proxied to support frontend health checks on the backend service. The frontend includes a centralised API client in frontend/src/api/client.ts where the API_BASE constant defaults to an empty string, and relative path routing is supported which works nicely with the Nginx proxy. The handleResponse function includes full error handling through response status checking, attempting JSON error body parsing, and constructing Error objects with status codes and precise error messages. The generic apiGet and apiPost functions wrap the native fetch API, automatically setting the correct headers and serializing request bodies to JSON.

Data Import and ML Workflow Implementation
The data import workflow follows a two-phase commit pattern with preview and commit endpoints. The frontend ImportFiles component handles the file uploads by building a FormData object and submitting it to /api/import/preview, complete with robust error handling, checking for 413 status codes (file too big), attempts to parse JSON error responses, and includes backup error messages. After a successful preview, the component saves the returned token and displays column mapping information, and then handleCommit function sends the token and mode to /api/import/commit with the same success notification and error messages.

The machine learning workflow process synchronizes simultaneous API calls to clustering and risk prediction services, while the runAllPredict function uses Promise.allSettled to execute both predictions simultaneously, such that partial success can occur and one model can fail without delaying the other. Outputs are stored in separate state variables, error handling displaying messages for failed predictions but still displaying successful outputs.

5.7 Software Testing

5.7.1 Frontend Test Suite

The frontend test suite of the Power BI Dashboard web application ensures that the user interface remains reliable, interactive, and consistent throughout development and deployment. The suite contain one major categories: Functional tests — verify that every feature and interaction performs its intended purpose. All tests are implemented using Jest and React Testing Library, supported by TypeScript and JSDOM for browser simulation.

Test Architecture
The application's source structure follows a separation of components under frontend/src, with corresponding test files grouped in frontend/src/tests. Jest's configuration file (jest.config.cjs) defines transformers and environment setup. A global setup script (setupTests.ts) initialises required browser polyfills such as TextEncoder and mock matchMedia.

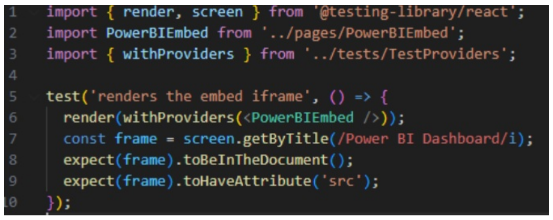
Authentication and Context (Unit test)
The AuthContext.test.tsx (Appendix, Fig 1) suite verifies that the authentication context maintains and updates the user state properly. The login() and logout() functions are ensured to modify username, role, and isAuthenticated across the app.
The Login.test.tsx (Appendix, Fig 2) checks that users can log in through role-specific buttons. The user is redirected to the home page upon success log in.

Protected Routing and Role Enforcement (Unit test)
The ProtectedRoute.test.tsx (Appendix, Fig 3) validates access control mechanisms. It ensures that:

- Unauthenticated users are redirected to /login.
- Authenticated users with correct role can access restricted routes.
- User with incorrect role is redirected to safe path.

These tests guarantee that the sensitive areas (administrative dashboards) remain protected and accessible only to authorized users.

Power BI Integration (Smoke test)
In PowerBIEmbed.test.tsx, the test simply checks that the PowerBIEmbed component actually shows the Power BI iframe on the screen. This tells us that the embed link is being loaded properly and that the dashboard can appear without the app crashing or failing to render.



```
import { render, screen } from '@testing-library/react';
import PowerBIEmbed from '../pages/PowerBIEmbed';
import { withProviders } from '../tests/TestProviders';

test('renders the embed iframe', () => {
  render(withProviders(<PowerBIEmbed />));
  const frame = screen.getByTitle(/Power BI Dashboard/i);
  expect(frame).toBeInTheDocument();
  expect(frame).toHaveAttribute('src');
});
```

5.7.2 Backend Test Suite

The backend test suite for the project ensures the reliability, correctness, and performance of every API route before deployment. It consists of two major parts:

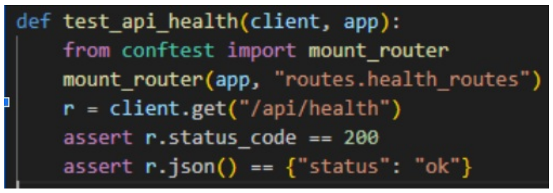
- Functional tests – verify that each route performs its intended operation correctly.
- Non-functional tests – examine the quality attributes such as speed, error handling etc.

All tests are implemented using Pytest together with FastAPI's TestClient, which allows sending simulated HTTP requests to the application without running a live server.

Test Architecture
All backend routes are stored under backend/routes, and each has a matching test file in backend/tests. The conftest.py file provides the shared setup and logic used by every test.

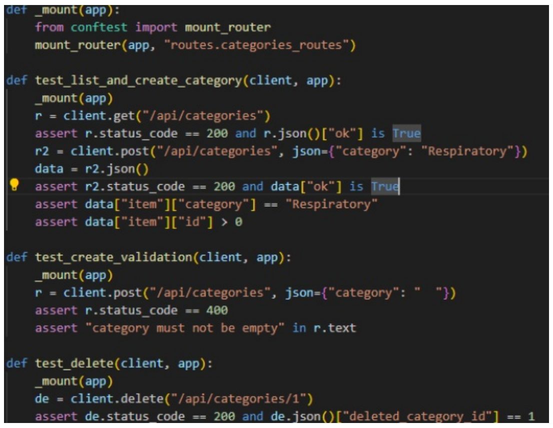
Functional Test

API Health and Availability (Smoke test)
The `test_health.py` suite confirms that the /api/health endpoint responds successfully and returns expected status indicators. This acts as a backend smoke test, ensuring that the FastAPI service is running, routes are registered properly, and no fatal initialisation errors occur.



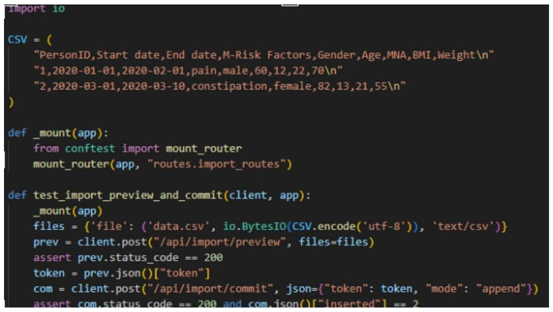
```
def test_api_health(client, app):
    from conftest import mount_router
    mount_router(app, "routes.health_routes")
    r = client.get("/api/health")
    assert r.status_code == 200
    assert r.json() == {"status": "ok"}
```

Category Management (Unit Test)
The `test_categories.py` file validates CRUD operations related to the /api/category endpoints. The categories can be retrieved, created, and managed through standard API interactions that can be validated by the tests. By processing both GET and POST flows, the category data integrity and database interaction logic across requests can be ensured.



```
def _mount(app):
    from conftest import mount_router
    mount_router(app, "routes.categories_routes")

def test_list_and_create_category(client, app):
    _mount(app)
    r = client.get("/api/categories")
    assert r.status_code == 200 and r.json()["ok"] is True
    r2 = client.post("/api/categories", json={"category": "Respiratory"})
    data = r2.json()
    assert r2.status_code == 200 and data["ok"] is True
    assert data["item"]["category"] == "Respiratory"
    assert data["item"]["id"] > 0

def test_create_validation(client, app):
    _mount(app)
    r = client.post("/api/categories", json={"category": "  "})
    assert r.status_code == 400
    assert "category must not be empty" in r.text

def test_delete(client, app):
    _mount(app)
    de = client.delete("/api/categories/1")
    assert de.status_code == 200 and de.json()["deleted_category_id"] == 1
```

Import Workflow Test(Unit Test)
The `test_import.py` suite validates the CSV import flow end-to-end. File upload using FastAPI's /api/import/preview route is simulated , the generated token is generated, and then /api/import/commit is triggered and data is inserted.



```python
import io

CSV = (
    "PersonID,Start date,End date,M-Risk Factors,Gender,Age,MNA,BMI,Weight\n"
    "1,2020-01-01,2020-02-01,pain,male,60,12,22,70\n"
    "2,2020-03-01,2020-03-10,constipation,female,82,13,21,55\n"
)

def _mount(app):
    from conftest import mount_router
    mount_router(app, "routes.import_routes")

def test_import_preview_and_commit(client, app):
    _mount(app)
    files = {'file': ('data.csv', io.BytesIO(CSV.encode('utf-8')), 'text/csv')}
    prev = client.post("/api/import/preview", files=files)
    assert prev.status_code == 200
    token = prev.json()["token"]
    com = client.post("/api/import/commit", json={"token": token, "mode": "append"})
    assert com.status_code == 200 and com.json()["inserted"] == 2
```

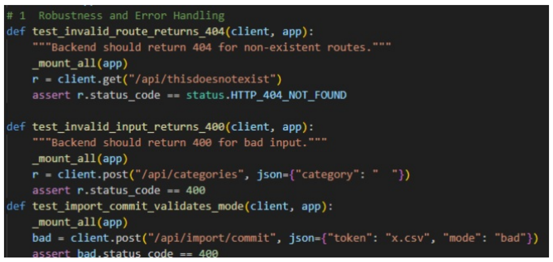
Risk Prediction API (Unit Test)
The `test_risk_predict.py`(Appendix, Fig 4) suite verifies the /risk/predict endpoint. The first test is used to ensure that valid resident data are processed correctly and return a structured list with expected fields like age, gender, condition count, and score. The batch data processing and correct prediction response is confirmed by the test. The second test forces a prediction failure by monkey-patching the predictor. The API returns a clean 500 error with a controlled message is tested. These tests validate both normal operation and safe error handling.

Non-Functional Tests

Robustness and Error Handling (Negative tests)
This test confirms:

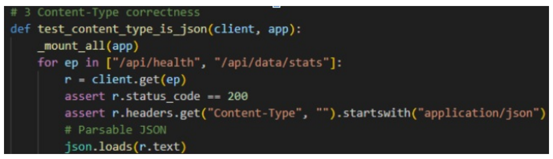
- Invalid endpoints (/api/thisdoesnotexist) return 404.
- Invalid inputs (e.g. blank category names) return 400.

The test is used to ensure the API fails correctly and communicates problems clearly.



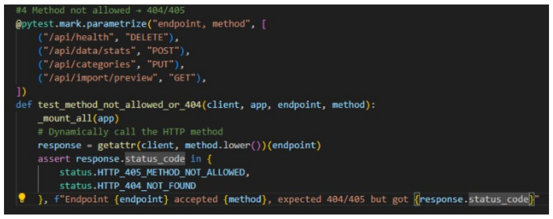
```python
# 1  Robustness and Error Handling
def test_invalid_route_returns_404(client, app):
    """Backend should return 404 for non-existent routes."""
    _mount_all(app)
    r = client.get("/api/thisdoesnotexist")
    assert r.status_code == status.HTTP_404_NOT_FOUND

def test_invalid_input_returns_400(client, app):
    """Backend should return 400 for bad input."""
    _mount_all(app)
    r = client.post("/api/categories", json={"category": "  "})
    assert r.status_code == 400
def test_import_commit_validates_mode(client, app):
    _mount_all(app)
    bad = client.post("/api/import/commit", json={"token": "x.csv", "mode": "bad"})
    assert bad.status_code == 400
```

Content-Type Contract Verification
This test ensures that the API responds using valid JSON.



```python
# 3 Content-Type correctness
def test_content_type_is_json(client, app):
    _mount_all(app)
    for ep in ["/api/health", "/api/data/stats"]:
        r = client.get(ep)
        assert r.status_code == 200
        assert r.headers.get("Content-Type", "").startswith("application/json")
        # Parsable JSON
        json.loads(r.text)
```

API Hardening
The test is to ensure the API does not accidentally accept unsupported or unintended HTTP methods. It tries invalid HTTP method combinations such as: DELETE /api/health → health is GET-only, POST /api/data/stats → likely GET-only
It confirms that in each case, the API returns 405 METHOD NOT ALLOWED or 404 NOT FOUND



```
#4 Method not allowed → 404/405
@pytest.mark.parametrize("endpoint, method", [
    ("/api/health", "DELETE"),
    ("/api/data/stats", "POST"),
    ("/api/categories", "PUT"),
    ("/api/import/preview", "GET"),
])
def test_method_not_allowed_or_404(client, app, endpoint, method):
    _mount_all(app)
    # Dynamically call the HTTP method
    response = getattr(client, method.lower())(endpoint)
    assert response.status_code in {
        status.HTTP_405_METHOD_NOT_ALLOWED,
        status.HTTP_404_NOT_FOUND
    }, f"Endpoint {endpoint} accepted {method}, expected 404/405 but got {response.status_code}"
```

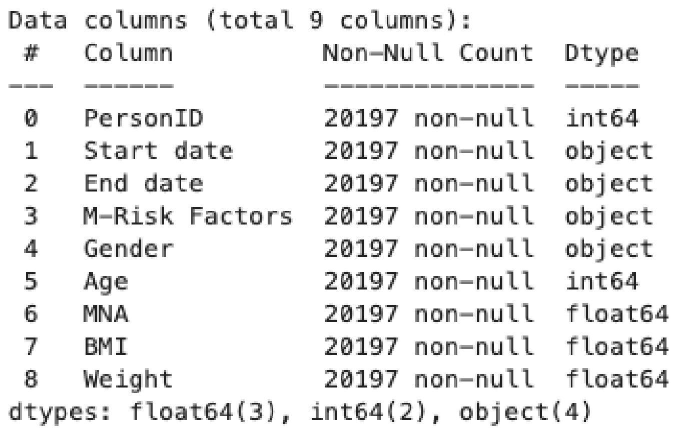
5.8 Intervention Recommendation System

The intervention recommendation system is integrated into the machine learning analysis workflow, When users complete the resident information form (including patient ID, name, gender, and age) and select relevant symptoms from the categorised library, the runAllPredict() function executes two parallel API calls: one to /cluster/usage/predict for cluster classification and another to /risk/predict for risk assessment. The risk prediction endpoint receives an array containing the resident's demographic data (name, age, gender) along with a conditions_dict object that maps all risk factor columns to binary values (0 or 1) based on the selected symptoms. The backend risk prediction model processes this data through an SVD-based dimensionality reduction pipeline combined with a trained risk estimator, generating not only the risk level classification (Low/Medium/High) and probability score but also an array of tailored clinical recommendations specific to the patient's risk profile and active conditions. These recommendations are returned as part of the risk prediction response object and stored in the riskResult[0].recommendations array in the component state.

The frontend displays these recommendations in a dedicated "Recommendations & Interventions" card positioned alongside the cluster information and risk assessment cards in a three-column layout, where each recommendation is rendered as a list item with the first recommendation emphasised in bold. The system provides a fallback UI state when no recommendations are available, ensuring a consistent user experience even when the model cannot generate specific guidance. This architecture ensures that clinical staff receive actionable, evidence-based intervention suggestions immediately upon completing the risk analysis, with recommendations dynamically generated based on the unique combination of the patient's demographic characteristics, selected symptoms, and calculated risk level rather than relying on static, generic advice templates.

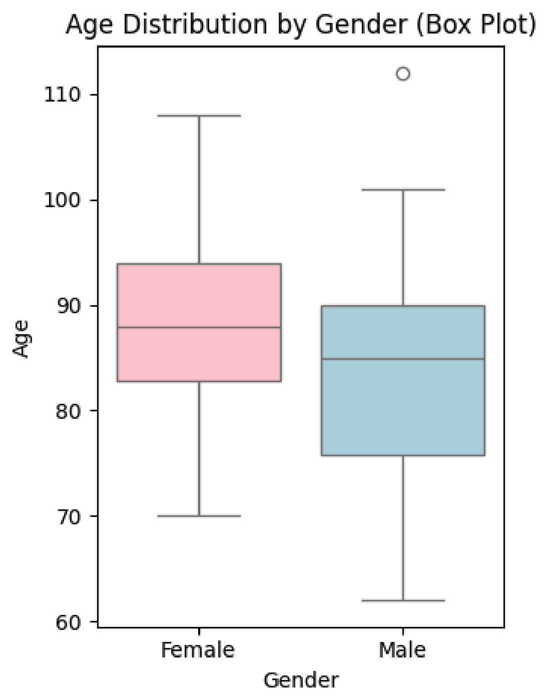
6 Results and Evaluation

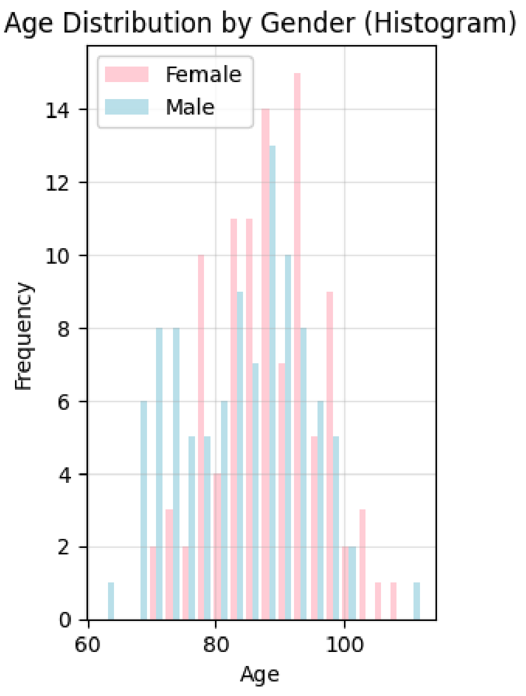
6.1 Dataset Overview



```
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   PersonID        20197 non-null  int64
 1   Start date      20197 non-null  object
 2   End date        20197 non-null  object
 3   M-Risk Factors  20197 non-null  object
 4   Gender          20197 non-null  object
 5   Age             20197 non-null  int64
 6   MNA             20197 non-null  float64
 7   BMI             20197 non-null  float64
 8   Weight          20197 non-null  float64
dtypes: float64(3), int64(2), object(4)
```

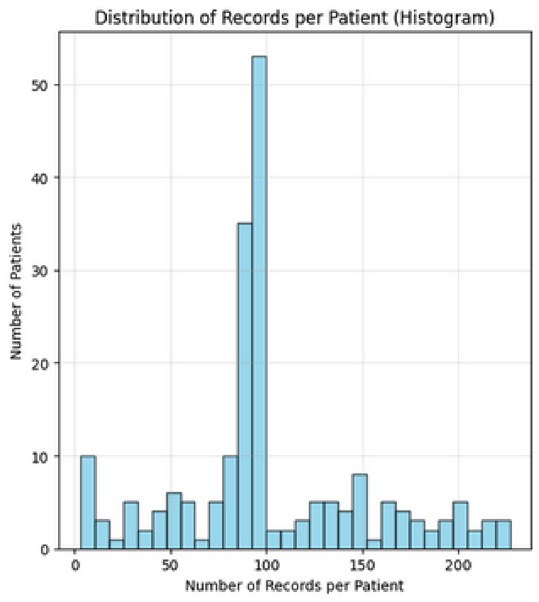
This table provides an overview of the data columns in a dataset that has 9 columns and 20,197 non-null values. The columns include PersonID (integer, 20,197 non-null), Start date (object, 20,197 non-null), End date (object, 20,197 non-null), M-Risk Factors (object, 20,197 non-null), Gender (object, 20,197 non-null), Age (integer, 20,197 non-null), MNA (float64, 20,197 non-null), BMI (float64, 20,197 non-null), and Weight (float64, 20,197 non-null). Types of data are a mixture of float64 (3 columns), int64 (2 columns), and object (4 columns), which indicates the blend of numerical and category/text data.

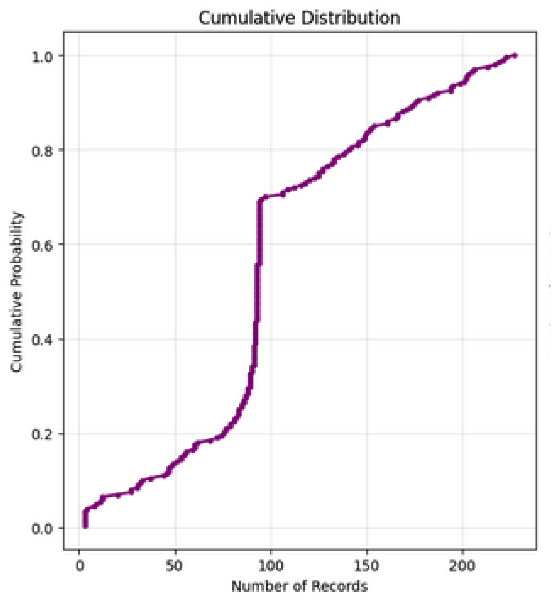




Age Distribution by Gender (Histogram): The histogram shows age distribution by sex, with age between 60 and 110 years on the x-axis and frequency on the y-axis. The cyan bars show females and the pink bars show males, both showing overlapping peaks at 80-90 years, with females more prevalent towards the upper end (about 90) and males having a wider spread. The graph highlights a widely similar age composition between the sexes with overall skewing towards older ages in the female sex.

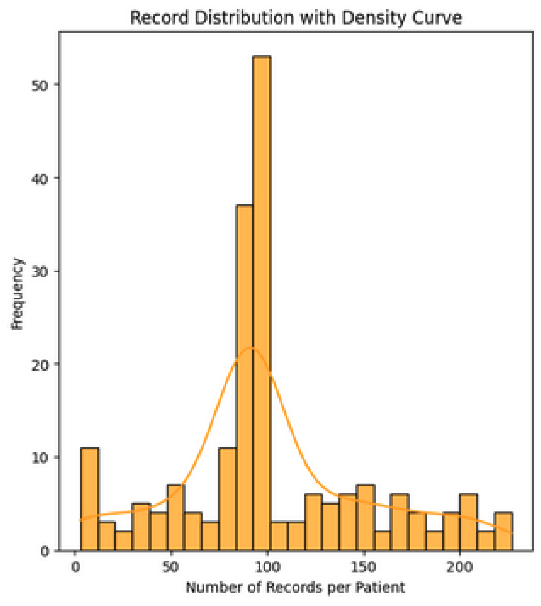
Age Distribution by Gender (Box Plot): This box plot displays age distribution by gender, where the x-axis classifies “Female” and ”Male” and y-axis represents ages ranging from 60 to 110. The female pink box and male cyan box both have medians of about 85-90 years, with interquartile ranges (IQR) ranging about 80-95 years. Whiskers achieve the lowest and highest ages (around 70-100), one outlier in males at around 110, displaying a comparatively wider range of ages for males compared to females.

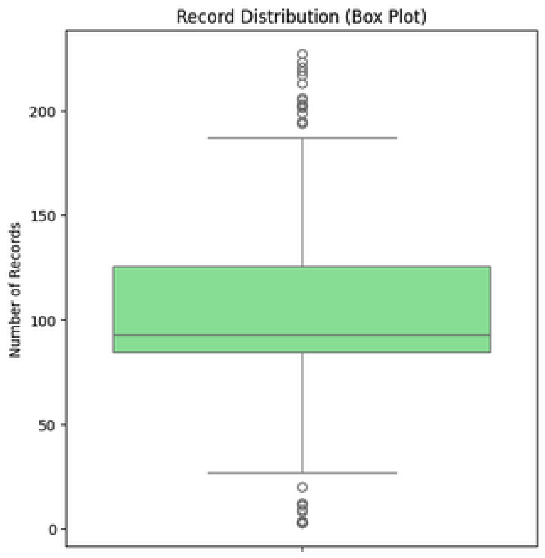




Distribution of Records per Patient (Histogram): This histogram indicates the count of patients on the x-axis versus the number of records they hold. The majority of the residents fall within 100 records, with a tall peak indicating the high concentration, and fewer residents holding records between below 50 or above 150. The light blue bars provide a clear indication of the distribution, including the central tendency and variability in record counts.

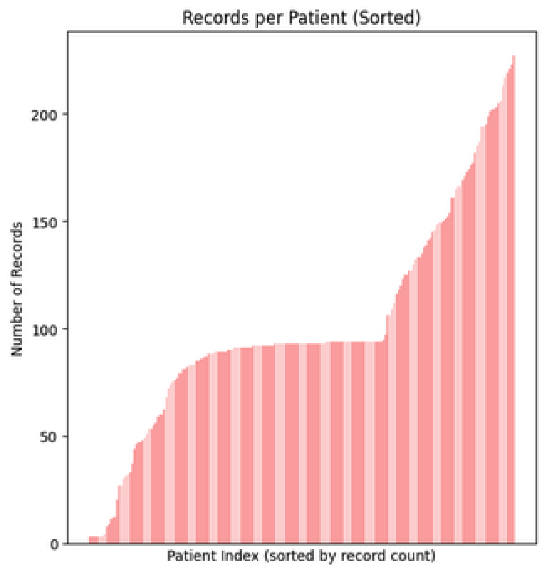
Cumulative Distribution: This CDF plot graphs the cumulative probability of records for each resident on the y-axis (0-1) versus records on the x-axis. The purple line rises steadily, most notably increasing rapidly between 100-150 records, reaching nearly 1, indicating that most residents have fewer than 150 records, and giving a general idea of the overall cumulative trend of the data.

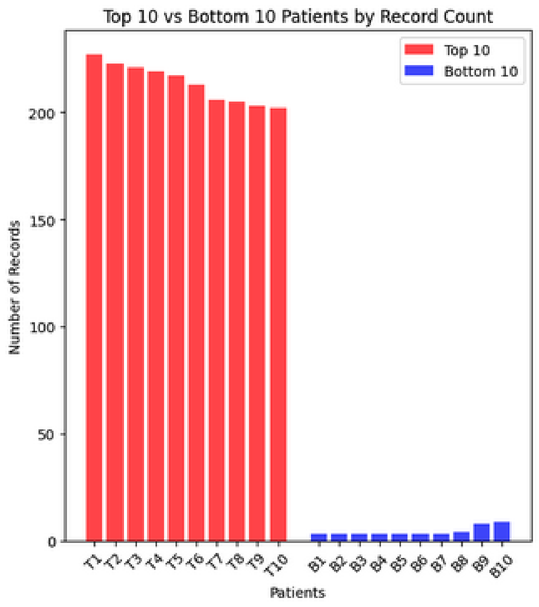




Record Distribution with Density Curve: The histogram, with a density curve overlaid, shows frequency of records per patient with the greatest frequency at 100 records, shown in orange bars. The yellow curve smooths the distribution, peaking similarly and declining, to provide a continuous approximation of the form of the data and to illustrate the central clumping with light tails on both sides.

Record Distribution (Box Plot): The box plot shows the distribution of records per patient, with the box indicating the interquartile range (IQR) positioned at 100-150 records, where the middle 50% of patients are located. The median is close to the upper quartile, with whiskers reaching the minimum and maximum values, and some outliers under 50 records indicated by circles. The green fill emphasizes high-density central clustering, which is useful for emphasizing typical and unusual record counts.



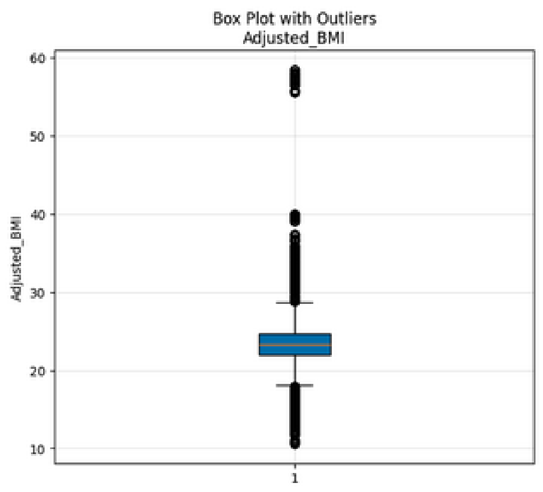


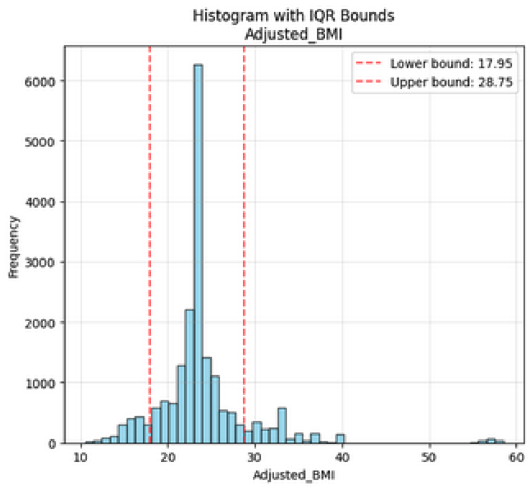
Records per Patient (Sorted): This is a sorted bar graph showing records per patient with x-axis as patients ordered by number of records and y-axis as record numbers to 200. The pink stripey bars spike high, with most of the patients having 100-150 records and some having over 150, visually highlighting the strongly skewed distribution where hardly any patients have a much higher record count.

Top 10 vs Bottom 10 Patients by Record Count: Here, the record counts for the top 10 and bottom 10 patients are being compared, with the most appropriate x-axis as patient indices (top 1-10, bottom 91-100) and the y-axis as records up to around 200. Red bars dominate for the top 10 with high counts (around 150-200), while blue bars for the bottom 10 reflect zero counts (virtually 0-50), referencing the immense disparity in data obtained by patients.

6.2 EDA Result

Outliers Detection Method Comparisons

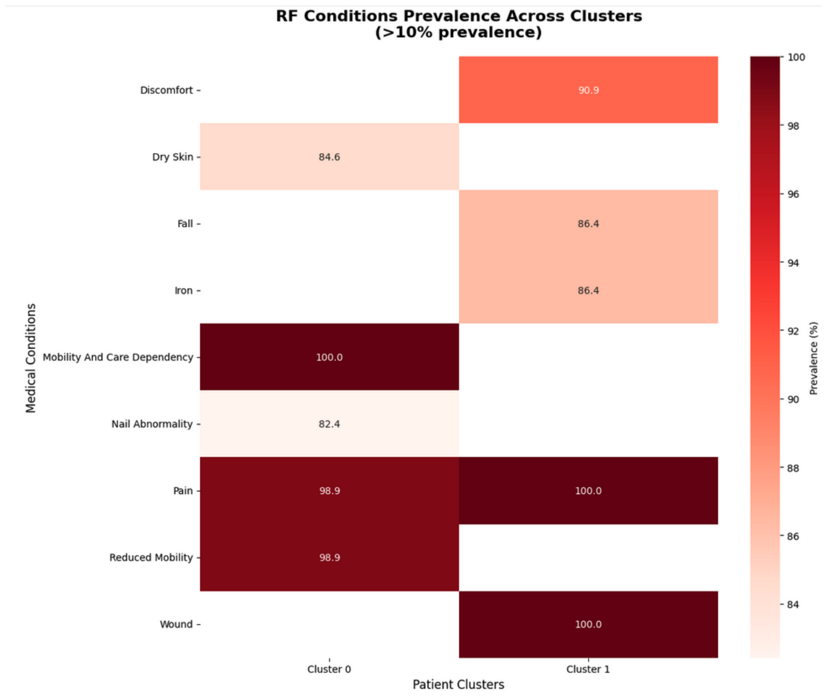




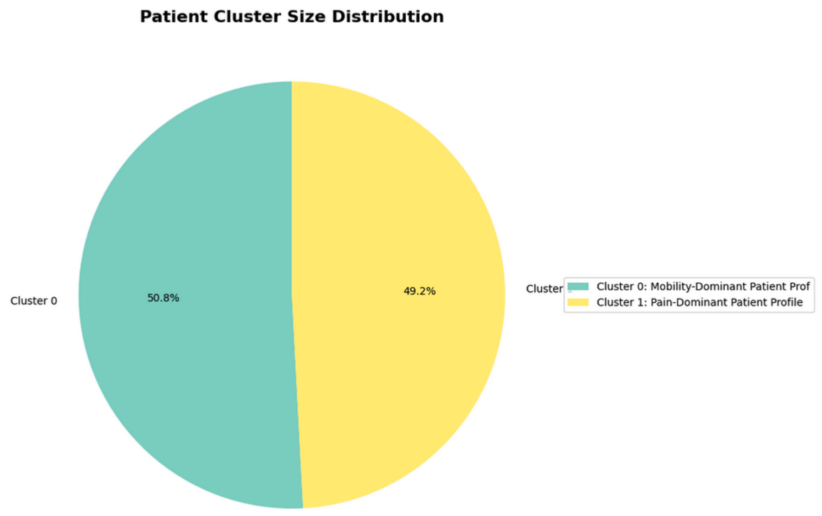
Box Plot with Outliers: This box plot illustrates the spread of Adjusted BMI scores, with the scale on the y-axis being between 10 and 60. The interquartile range (IQR) is represented by the middle box with a center of 20-30, median at around 25, and whiskers extending to the minimum and maximum value within 1.5x IQR. There are several outliers represented as circles above 40, which are near the much higher than average Adjusted BMI scores, showing potential anomalies that might be investigated further.

Histogram with IQR Bounds: This histogram indicates the frequency of Adjusted BMI on the y-axis (max 6000) against BMI values from 10 to 60 on the x-axis, where there are cyan spikes at 20-30. Red vertical lines are the IQR bounds (lower: 17.95, upper: 28.75) and indicate the middle 50% data. Points beyond the bounds, especially above 28.75, are indicated in lower frequency as outliers and can be utilized to identify extreme BMI cases.

6.3 Symptom Clustering Results

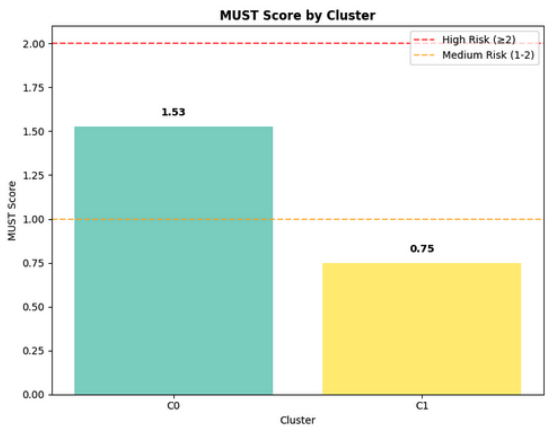


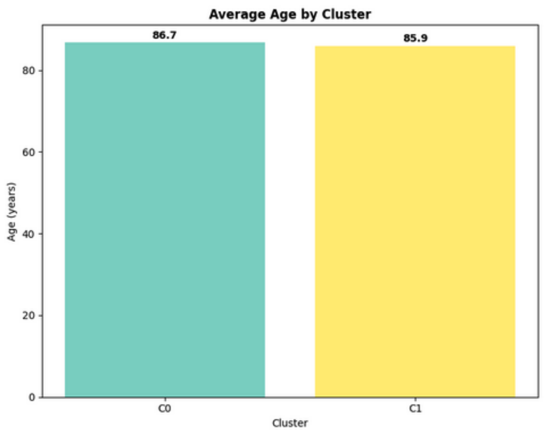
RF Conditions Prevalence Across Clusters: This heatmap contrasts the prevalence of risk factor (RF) conditions between two patient clusters (Cluster 0 and Cluster 1), where conditions with $>$ 10% prevalence are included. The y-axis shows conditions like Discomfort, Dry Skin, Fall, Iron, Mobility and Care Dependency, Nail Abnormality, Pain, Reduced Mobility, and Wound, and the x-axis shows the clusters. Prevalence rates (e.g. 100.0 for Mobility and Care Dependency in Cluster 0, 90.9 for Discomfort in Cluster 1) are graded in color on a scale from pale pink (84%) to dark red (100%) to show that Cluster 0 is characterized by mobility conditions and Cluster 1 has a high prevalence for every listed condition.



Patient Cluster Size Distribution: The distribution of patients in two clusters is depicted in this pie chart, where the pie is divided into a cyan slice (50.8%) representing Cluster 0 (Mobility-Dominant Patient Profile) and a yellow slice (49.2%) for Cluster 1 (Pain-Dominant Patient Profile). The proximity of percentages is an indication of even distribution, with the legend describing the dominant condition profiles, indicating a clear image of the split in the resident population based on health attributes.

Demographics Comparisons

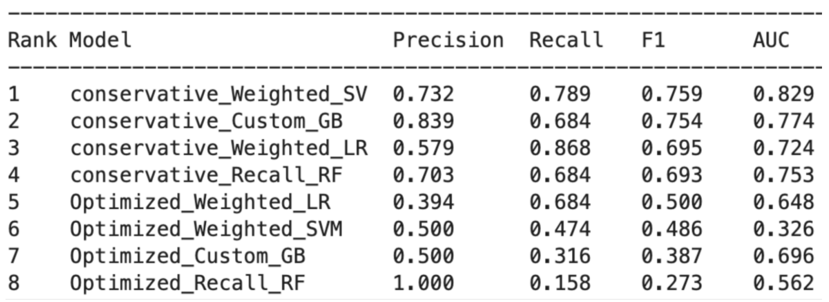




MUST Score by Cluster: This bar graph shows a comparison of mean MUST scores between two clusters on the x-axis marked as Cluster 0 (C0) and Cluster 1 (C1), with the y-axis from 0 to 2.5. The cyan bar for C0 shows a mean value of 1.53, while the yellow bar for C1 shows 0.75, and dashed orange and red lines show high-risk ($\geq$ 1.75) and medium-risk (1-1.75) values. This suggests that patients with C0 are at medium to higher risk, while C1 patients are generally less at risk.

Average Age by Cluster: In this bar graph, the average age of two groups of patients is compared, with the x-axis labeling Cluster 0 (C0) and Cluster 1 (C1) and the y-axis ranging from 20 to 90 years. The cyan bar for C0 represents an average age of 86.7 years, and the yellow bar for C1 represents 85.9 years, a slight age difference. Both groups have similar old age populations, but C0 patients are a little older on average.

6.4 Model Performance



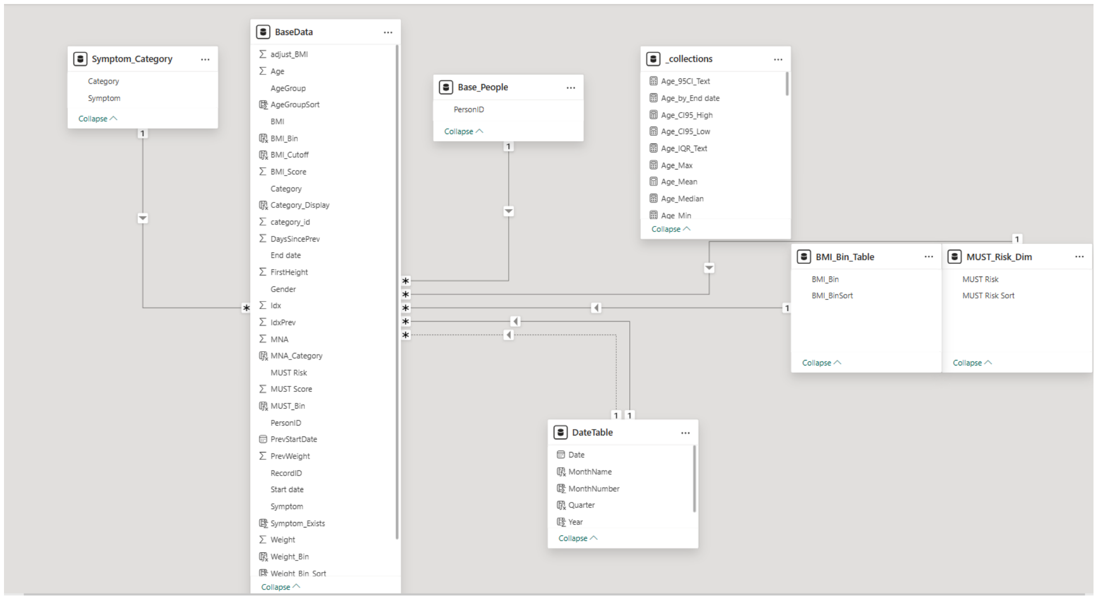
```
--------------------------------------------------------------------------
Rank Model                         Precision  Recall    F1        AUC
--------------------------------------------------------------------------
1    conservative_Weighted_SV      0.732      0.789     0.759     0.829
2    conservative_Custom_GB        0.839      0.684     0.754     0.774
3    conservative_Weighted_LR      0.579      0.868     0.695     0.724
4    conservative_Recall_RF        0.703      0.684     0.693     0.753
5    Optimized_Weighted_LR         0.394      0.684     0.500     0.648
6    Optimized_Weighted_SVM        0.500      0.474     0.486     0.326
7    Optimized_Custom_GB           0.500      0.316     0.387     0.696
8    Optimized_Recall_RF           1.000      0.158     0.273     0.562
```

This table contains a comparison between eight machine learning models based on their performance for high-risk class outcome prediction, including metrics like Precision, Recall, F1 Score, and AUC. Rank 1 model, "conservative_weighted_SVM," has Precision of 0.732, Recall of 0.789, F1 Score of 0.759, and AUC of 0.829 indicating high overall performance. Ranked at position 2 is "conservative_custom_GB" with extremely high Precision of 0.839 and Recall of 0.684, and ranking at position 3 is "conservative_weighted_LR" with Recall of 0.868 but precision of 0.579. Lower-performing models, such as "optimized_weighted_SVM" (Rank 6) and "optimized_recall_RF" (Rank 8), display mixed performance with the latter exhibiting flawless Precision (1.000) but low Recall of just 0.158, suggesting compromise on sensitivity. The evidence points towards "conservative_weighted_SVM" being the best-balanced performer in all aspects.

6.5 Dashboard Data Model Structure

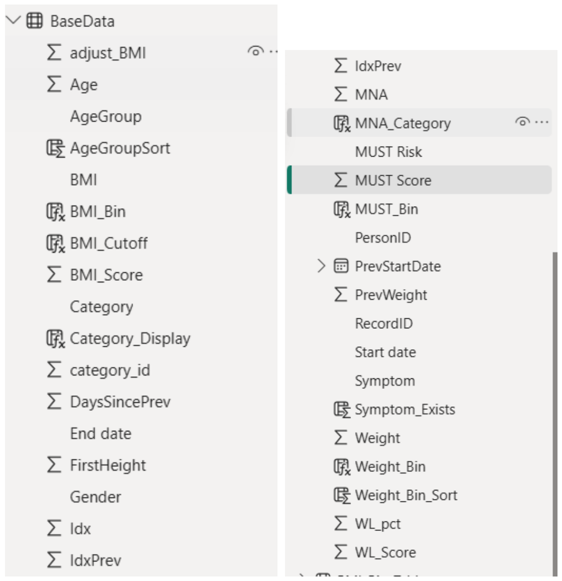
The raw data was saved in the MySQL database and was transformed in Power Query before entering Power BI. The process produced a solitary primary fact table, Base Dat,a and five other dimension tables (Symptom_ Category, Base_People, DateTable, BMI_Bin_Table, and MUST_Risk Dim).

The raw data was stored in a MySQL database and processed in Power Query before loaded into Power BI. The process created one main fact table Base_Data and supporting other five dimension tables (Symptom_Category, Base_People, DateTable, BMI_Bin_Table and MUST_Risk_Dim).



(a) Fact Table – BaseData Table

The BaseData table contains the minimum information of every resident, i.e., their PersonID, start and end dates. It also contains long-term healthcare data in the form of weight, BMI, MNA score, and symptomatic status. The columns PersonID, Start date, End date, Gender, Age, Weight, BMI, MNA, and Symptom are the raw dataset's source columns. The dashboard was simplified to analyse and visualise by deriving several new columns, i.e., AgeGroup, Symptom Category, MUST Score, MUST Bin, MNA Category, and Weight Bin.



(b) Dimension Tables

Table 1: Dimension Tables and Relationships in the Data Model

Table	Description	Relationship
Base_People	Resident demographics	1-to-many → BaseData
Symptom_Category	Symptom classification	1-to-many → BaseData
Date Table	Calendar for time trend analysis	1-to-many → BaseData
MUST_Risk_Dim	Defines MUST risk order for chart sorting	1-to-many → BaseData
BMI_Bin_Table	Defines BMI range order for chart sorting	1-to-many → BaseData

The model utilises onedirection relations by employing integer keys so that the filters can run faster and avoid confusion among tables.

(c) Model Optimisation
The dashboard performance was enhanced by the elimination of unnecessary columns and the alteration of the data types to numeric where feasible. Query folding was carried out in MySQL rather than Power BI memory in an attempt to avoid heavy operations, such as joins and sorting. DAX measures are intended for the dashboard's leading indicators rather than columns that are directly calculated in the BaseData table, allowing the refresh time to remain short.

(d) DAX Measures
To support analysis and interactive visualisation, a separate semantic layer was developed in Power BI using a collection of DAX measures stored in the table _collections. This layer acts as a core analytical engine for the dashboard and contains all key calculations used across different pages, such as average, medians, mean value, confidence intervals, IQR and category percentages. By keeping all DAX measures in one place, the model ensures consistency across visuals and reduces the need for repeated calculations, improving performance and maintainability. The measures are grouped according to their analytical purpose and page usage:

Table 2: Measure Groups and Their Usage Across Dashboard Pages

Measure Group	Example Measures	Used in Dashboard Pages
Demographic Measures	Age_Mean, Age_Median, Age_95CI_Text, Age_IQR_Text, Female_Percentage_Text, Male_Percentage_Text	Resident Overview; Gender Breakdown
Weight Related Measures	Weight_Mean, Weight_Median, Weight_95CI_Text, Weight_IQR_Text, BMI_Mean, BMI_Median, BMI_95CI_Text, BMI_IQR_Text, BMI_Bin	Weight Distribution; BMI Profile
Nutritional Measures	MNA_Mean, MNA_Median, MNA_95CI_Text, MNA_IQR_Text	Resident Overview; Nutritional Status
Symptom Analysis	Symptom_Count, Symptom_Percentage, HighRisk_Symptom_Count	Symptom Distribution; Resident Profile
Resident-Level Measures	Latest_BMI, Latest_MNA, Latest_MUST_Risk, Latest_Weight, Length_of_Stay, Resident_n	Resident Profile; Resident Overview

6.6 Dashboard Overview and User Views

The Power BI dashboard includes three analytical levels. It allows different roles of user such as managers, nurses, and care workers, to explore the data from an overall view down to each resident's personal record. Global filters and bookmarks connect all pages for smooth navigation.

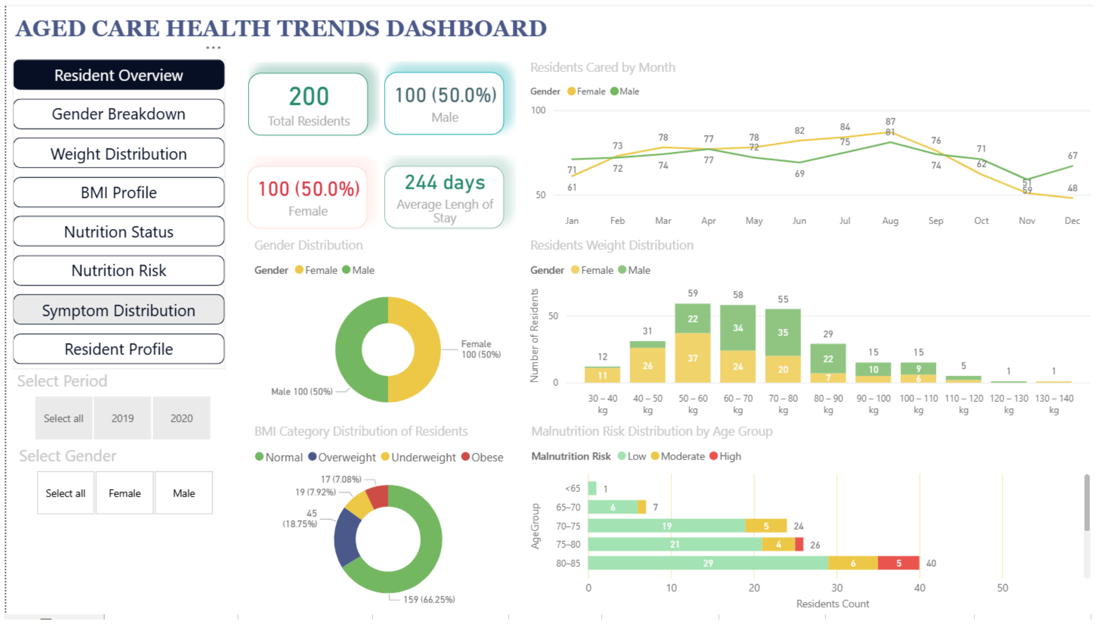
Level 1: Core Overview and Demographic Analysis
Level one provide an overview of residents' demographics and health overview data through four pages: Resident Overview, Gender Breakdown, Weight Distribution and BMI Profile.

Resident Overview
This is the home page of the dashboard. It shows essential KPIs, including Total Residents, Gender Ratio, and Average Length of Stay for residents in an aged care facility.

A line chart shows Residents Cared by Month, highlighting gender and seasonal trends. A histogram displays the Weight Distribution across male and female residents. Also, a stacked bar chart presents the MUST score risk across all the age groups. A doughnut chart displays gender and BMI distribution to give a clear insight into residents' demographic information.

Tooltip interactions
Hovering on any point shows the number of residents and the percentage change that month. Hovering on a weight bar displays the number of the selected group, the ratio by gender, and the mean age for the selected group.

Summary
This page gives the user a quick snapshot of overall operations and population health. The interactive tooltips allow users to explore exact values without leaving the page. The design of the Resident Overview page is shown in the figure below.



Gender Breakdown

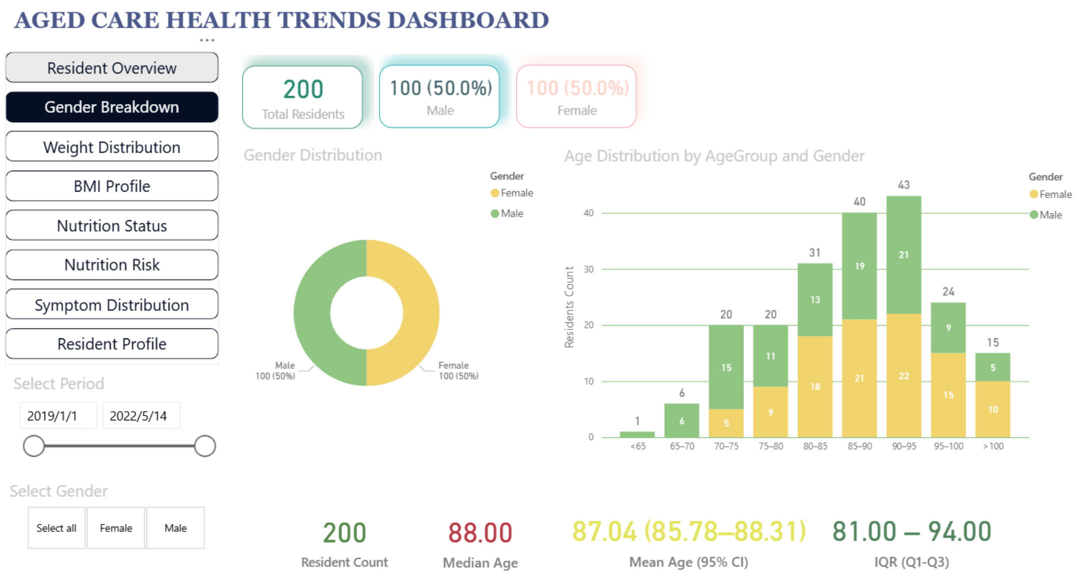
This page explores how male and female residents differ in age. The donut chart shows the distribution and proportion of residents. The stacked bar chart Age Distribution by Age Group and Gender compares both genders, while the KPI cards display the Median Age and Mean Age. The 95% CI and IQR (Q1–Q3) provide a clear summary of age variation among residents, showing both the reliability of the mean estimate and the spread of ages within the middle 50% of the population.

Tooltip interactions

Mouse over displays the number of residents by sex and age group.

Summary

The Gender Breakdown page facilitates the spotting of population-based distinctions. The structure of the Gender Breakdown page appears in the following figure.



Weight Distribution

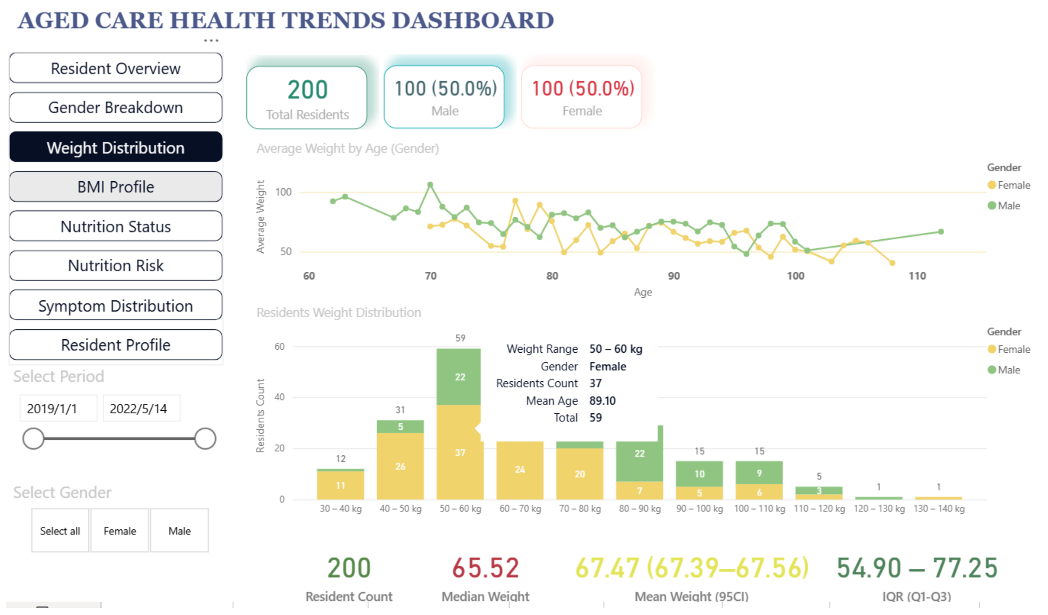
It's regarding weight disparities by sex and age. The line chart Average Weight by Age shows the general pattern, and the histogram indicates the distribution of the residents' weights. At the bottom of the page, KPI cards show the Median Weight, Mean Weight, and the IQR (Q1–Q3), giving an account of the distribution of the weight among the resident population.

Tooltip interactions

Mouse over displays the mean weight by gender at that age group and how many residents are weighing in that range.

Summary

This page helps detect abnormal weight ranges and compare male–female patterns clearly. The design of the Weight Distribution page is shown in the following figure..



BMI Profile

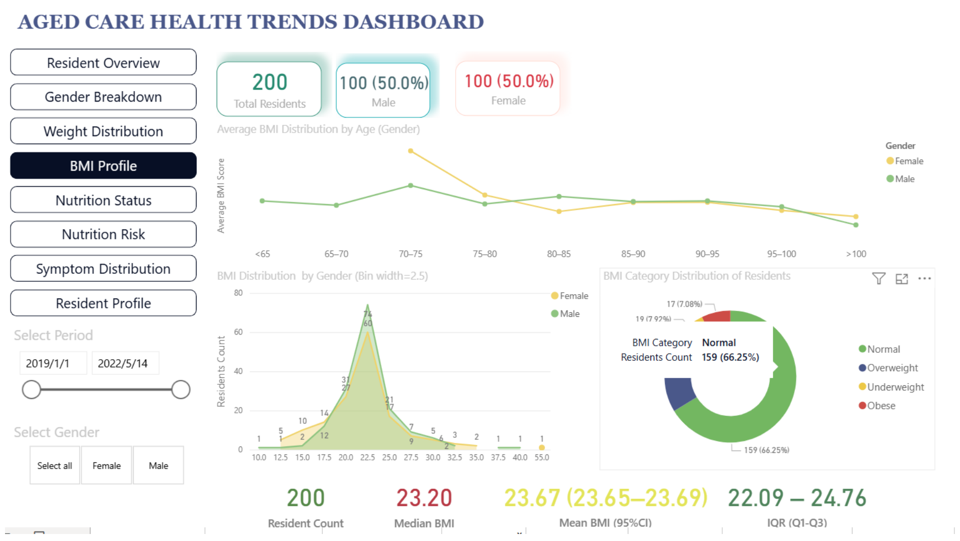
The following page provides the classification of BMI among various age categories and sexes. The doughnut chart displays the numbers under the Underweight, Normal, Overweight, and Obese categories. The line chart displays the mean BMI by age period and sex, allowing employees to compare the differences in BMI by age period. Additionally, an area chart indicates how the residents are distributed across different intervals of their BMIs (with a bin step of 2.5). KPI cards show Median BMI, Mean BMI and BMI's IQR range.

Tooltip interactions

Mousing over the doughnut chart shows detailed proportions for each BMI category, such as Normal: 66.25% (n=159). Hovering over the line or area chart displays the corresponding age group, gender, and resident count within each BMI range.

Summary

This visual helps aged care staff understand residents' BMI patterns and monitor balance across different demographic group. The design of the Weight Distribution page see the figure below.



Level 2: Health Risk and Symptom Trends

Level 2 focuses on analysing residents' nutrition risks and common symptoms across three pages: Nutrition Status, Nutrition Risk, and Symptom Distribution.

Nutritional Status

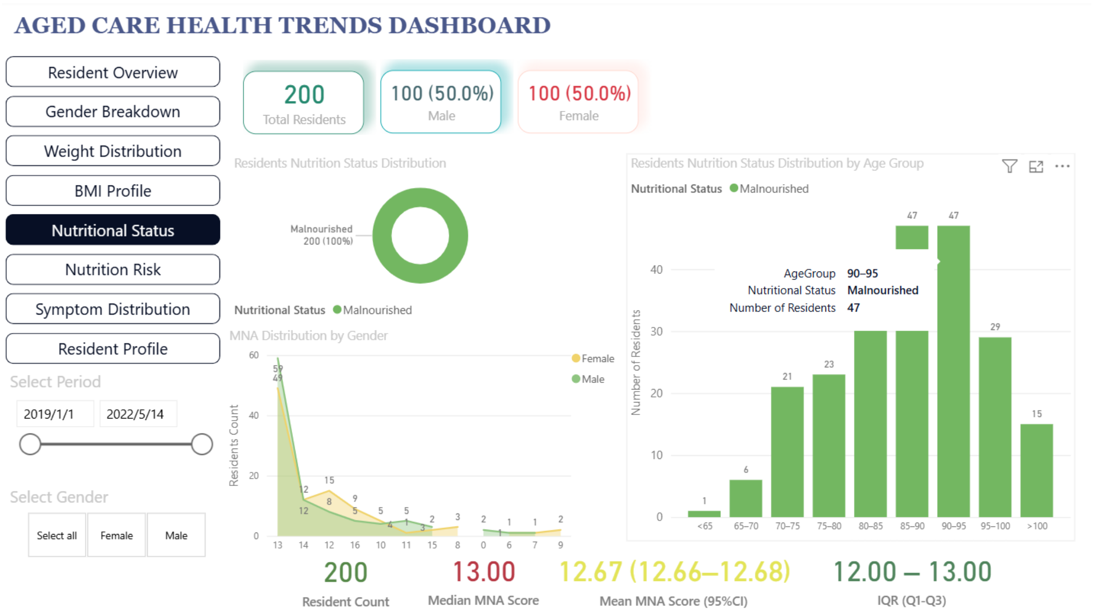
On this page, the nutritional status is presented through an MNA-based nutritional assessment. The pie chart shows the proportion of residents in each MNA category, while the bar chart compares malnutrition levels across different age groups. In addition, an area chart illustrates how residents are distributed across various MNA score ranges. The KPI cards show the Median MNA, the mean MNA, and the IQR (Q1–Q3), providing a quick summary of the general distribution of nutrition.

Tooltip interactions

Hovering on a bar shows the number of residents and the average MNA score for that group. Hovering over the line or area chart displays the corresponding age group, gender, and resident count within each MNA.

Summary

This page provides a clear view of the prevalence of malnutrition in age and sex. The details of the Nutritional Status page see the below figure.



Nutrition Risk

The Nutrition Risk page is designed to visualize the Malnutrition Universal Screening Tool (MUST) results for all residents in a clear and structured way. The layout follows a three-part format: top KPI summary, middle visual comparison, and bottom detailed data table.

As other pages, at the page top, three KPI cards give an at-a-glance overview of major figures: Total Resident, Male, and Female. These KPI cards ensure consistency for all dashboard pages.

The center element offers two overriding visual aspects:

- The leftmost doughnut chart, titled MUST Score Distribution by Gender, presents the distribution of the proportion of residents by risk levels — Low, Moderate, and High. The colour sequence has a gradient of green-yellow-red, whereby the severity of the risk levels is clearly distinguished.
- On the far right, a stacked bar chart disaggregates these risk categories by various age ranges (e.g. <65, 65–70, 70–75, etc.). The individual bar in each stacked bar shows the resident population in each risk level, so it's easy to compare by eye how malnutrition risk rises with age.

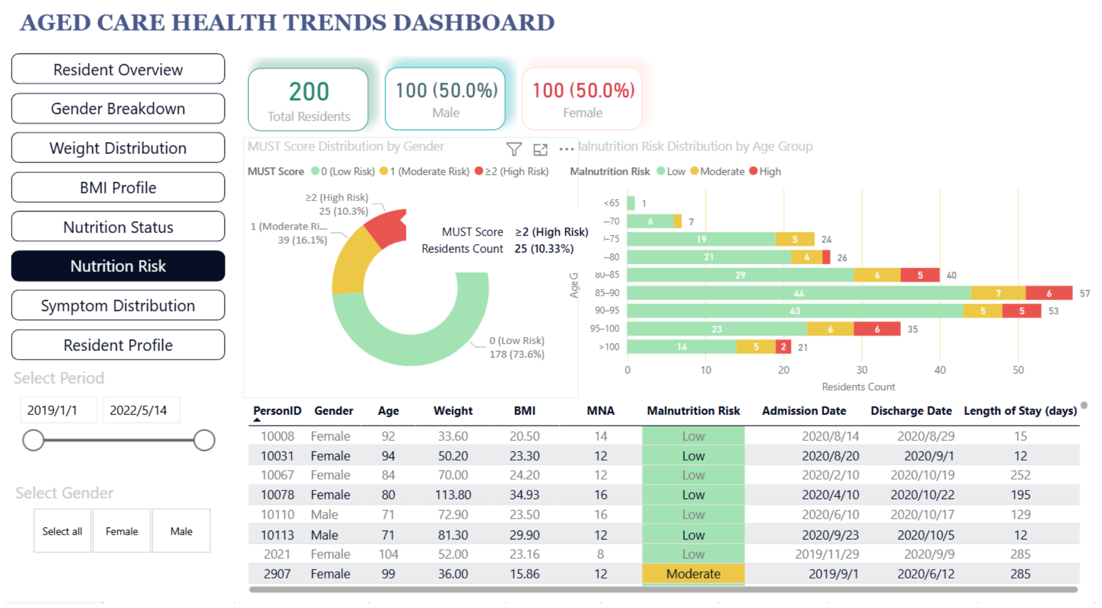
At the bottom of the page, it provides a data table listing detailed resident information at the risk level that was selected. It is related to the MUST Score Distribution pie chart — when the user selects any risk level, the table automatically displays the corresponding resident information, i.e., PersonID, Gender, Age, Weight, BMI, MNA, Malnutrition Risk, Admission Date, Discharge Date, and Length of Stay (days). There is conditional formatting in the Malnutrition Risk column through the use of colour codes — green at Low, yellow at Moderate, red at High — so that the user can readily identify the residents at higher risk. The table is also interactive so that cross-highlighting by the charts above the page can be done by selecting a particular row.

Tooltip interactions

Mousing displays sample size and proportion for every risk level. Clicking on a record brings up the same resident in charts.

Summary

This page connects group-level risk trends to individual records for targeted care. The Following figure shows the design and key features of the Nutrition Status page.



PersonID	Gender	Age	Weight	BMI	MNA	Malnutrition Risk	Admission Date	Discharge Date	Length of Stay (days)
10008	Female	92	33.60	20.50	14	Low	2020/8/14	2020/8/29	15
10031	Female	94	50.20	23.30	12	Low	2020/8/20	2020/9/1	12
10067	Female	84	70.00	24.20	12	Low	2020/2/10	2020/10/19	252
10078	Female	80	113.80	34.93	16	Low	2020/4/10	2020/10/22	195
10110	Male	71	72.90	23.50	16	Low	2020/6/10	2020/10/17	129
10113	Male	71	81.30	29.90	12	Low	2020/9/23	2020/10/5	12
2021	Female	104	52.00	23.16	8	Low	2019/11/29	2020/9/9	285
2907	Female	99	36.00	15.86	12	Moderate	2019/9/1	2020/6/12	285

Symptom Distribution

The Symptom Distribution page offers a summary of the most frequent health records among the residents and how their frequency evolves over time. The structure has a distinct three-part arrangement in sections: KPI summary, distribution charts in the visual format, and a heatmap in the monthly trend.

At the top of the page, three KPI cards summarise the key resident statistics. In the middle section, two charts work together to provide both categorical and frequency-based insights:

- On the left, a donut chart provides the percentage distribution of each symptom group (for example, Pain 11.8%, Mobility 11.4%, etc.). The various colors in the donut chart indicate various types of symptoms so that the staff can at a glance determine which groups dominate the most. The donut chart is completely dynamic and has links with the symptom bar chart and the heatmap as well. When the user points at any symptom group, the bar chart and the table automatically show the respective monthly trends of the symptoms and the occurrence in that group.
- On the right, a stacked bar chart that ranks individual symptoms by overall frequency. The format makes it possible to tell at a glance which issues most often happen across the facility.

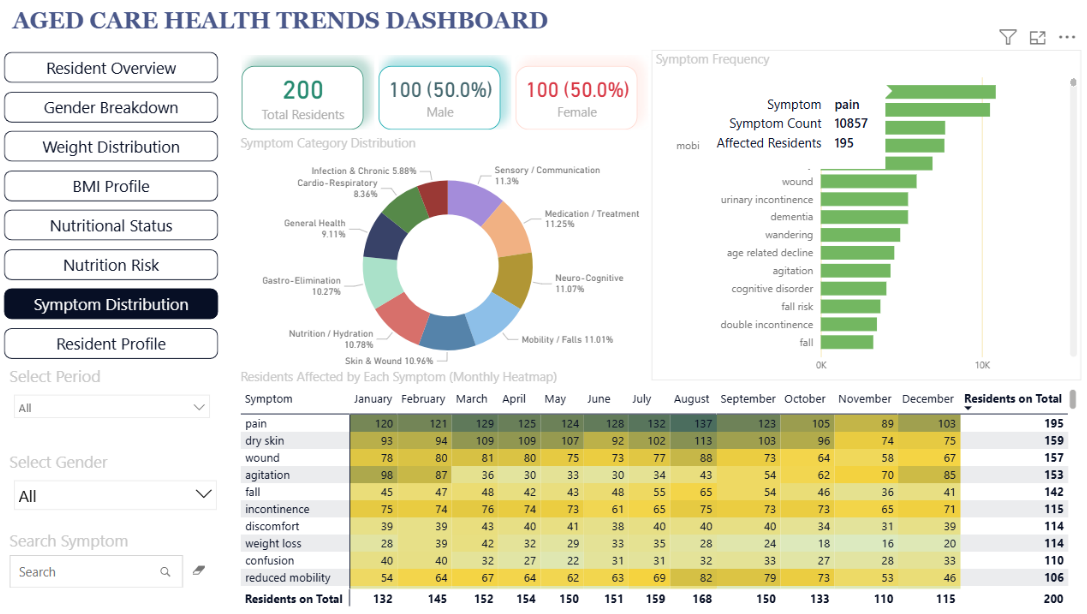
At the bottom of the page, there is a heatmap table called Residents Affected by Each Symptom (Monthly) that shows monthly trends in symptoms. Each cell shows the number of residents who were affected by a particular symptom during a particular month, and the intensity of the colour shows the frequency (darker colours = higher occurrence). The rightmost column summarises the total affected residents for each symptom (e.g. Pain: 195 residents, Dry Skin: 159). This visual allows users to detect seasonal or recurring patterns — for example, higher infection or skin problems during winter months.

On the left-hand filter panel, users can control the view using three slicers:

- Select Period – filters by time frame;
- Select Gender – filters by male, female, or all residents;
- Search Symptom – supports keyword search for specific symptoms.

Tooltip interactions
Hovering on the symptom frequency bar chart displays the symptom name, symptom count and the number of residents affected by this symptom.

Summary
This page helps clinical teams observe which symptoms are most common and when they increase during the year. The Following figure shows the page of symptoms distribution.



Symptom pain
Symptom Count 10857
Affected Residents 195

Symptom	January	February	March	April	May	June	July	August	September	October	November	December	Residents on Total
pain	120	121	129	125	124	128	132	137	123	105	89	103	**195**
dry skin	93	94	109	109	107	92	102	113	103	96	74	75	**159**
wound	78	80	81	80	75	73	77	88	73	64	58	67	**157**
agitation	98	87	36	30	33	30	34	43	54	62	70	85	**153**
fall	45	47	48	42	43	48	55	65	54	46	36	41	**142**
incontinence	75	74	76	74	73	61	65	75	73	73	65	71	**115**
discomfort	39	39	43	40	41	38	40	40	40	34	31	39	**114**
weight loss	28	39	42	32	29	33	35	28	24	18	16	20	**114**
confusion	40	40	32	27	22	31	31	32	33	27	28	33	**110**
reduced mobility	54	64	67	64	62	63	69	82	79	73	53	46	**106**
Residents on Total	**132**	**145**	**152**	**154**	**150**	**151**	**159**	**168**	**150**	**133**	**110**	**115**	**200**

Level 3: Individual Resident Profile
Level 3- the resident profile page provide a detail view of an individual resident's health record and personal health trends over time. The page layout is divide into four main areas: personal information KPI summary, health risk indicators, time-series chart, and symptom tracking heatmap.

At the top of the page, several KPI cards display the resident's key information, including Gender, Age, Admission Date, Discharge Date, Length of Stay, Weight and Malnutrition Risk. Each KPI is colour-coded according to its category. For example, the Malnutrition Risk card uses yellow to highlight a Moderate risk level, while demographic cards use neutral colours for clarity.

In the middle section, two charts visualise the residents' health composition and weight changes over time:

- On the left, a doughnut chart presents the distribution of symptom categories, aggregated by the total symptom count recorded for the resident (e.g. Pain 20.5%, Mobility 20.17%, etc.). This allows staff to see which types of symptoms occurred most frequently during the resident's stay in the aged care facility period.

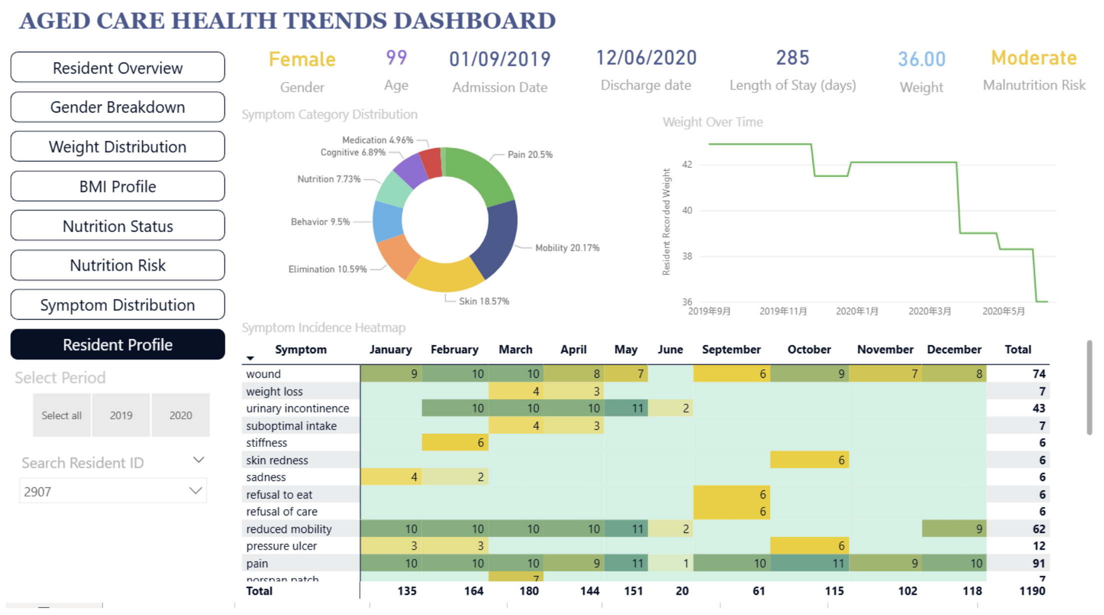
- On the right, the Weight Over Time line chart, displays the resident's recorded weight trend during the stay so that the nurse and carers can identify health changes and offer timely care intervention easily.

At the bottom of the page, a heatmap table can help users track monthly symptom occurrences for this specific resident. Each row represents a symptom, while each column corresponds to a month. The colour gradient ranges from light green (low frequency) to yellow and dark green (high frequency). The far-right column displays the total number of occurrences for each symptom across the time period. This layout allows staff to identify recurring issues and assess the relationship between weight loss and symptom frequency.

On the left panel, users can look up a resident by using the Search Resident ID filter or choose a year period for reporting (e.g. 2019, 2020). The page automatically reloads the moment the end-user chooses a new resident, and the page shows the resident's individual health details and timeline.

Tooltip interactions
Mousing over the Weight Over Time line chart displays the actual weight and the day (e.g. 2020-03-12: 38.5kg). Hovering over the pieces of the donut chart displays the name of each group of symptoms and the total symptoms' count contain in this category.

The design of the Resident Profile page see the figure below.



Female	99	01/09/2019	12/06/2020	285	36.00	Moderate
Gender	Age	Admission Date	Discharge date	Length of Stay (days)	Weight	Malnutrition Risk

Symptom	January	February	March	April	May	June	September	October	November	December	Total
wound	9	10	10	8	7		6	9	7	8	74
weight loss			4	3							7
urinary incontinence		10	10	10	11	2					43
suboptimal intake			4	3							7
stiffness		6									6
skin redness								6			6
sadness	4	2									6
refusal to eat							6				6
refusal of care							6				6
reduced mobility	10	10	10	10	11	2				9	62
pressure ulcer	3	3						6			12
pain	10	10	10	9	11	1	10	11	9	10	91
norspan patch			7								7
Total	135	164	180	144	151	20	61	115	102	118	1190

7 Further Discussions

7.1 Key Outputs

The main aim of this project is to develop a comprehensive, automated data-driven visualisation platform designed to examine health trends in aged care. This system successfully brings together various components into one functional tool. The core of the system is an interactive Power BI dashboard which enables multiple views into the resident information, including population views, demographic breakdowns and symptom distribution. This dashboard translates complex datasets into clear and intuitive visual experiences where healthcare staff can monitor population health, while finding early emerging trends without the need for complex data analysis skills.

The interactive dashboard is combined with a solid backend system to facilitate the processing of health information via a modular architecture. This incorporates a complete data processing 'pipe' to clean, transform and prepare raw health information for analysis. In addition, the project resulted in two different machine learning models, including a model to classify patients into risk categories based on nutritional status and a clustering model to group residents based on their symptom profiles. Together, these systems function to create a smooth and seamless workflow from raw data to predictive insight and visualisation for a practical method of data-driven decision making for dementia care.

7.2 Challenges and Contribution

During the project, a common difficulty that the team had to overcome was the quality and nature of the aged care dataset. The database was unbalanced, as it only included residents whose MNA score was less than 17, so all the cases were malnutrition. The imbalance played a crucial role in the subsequent steps in the capabilities of the machine learning algorithm in correctly classifying and updating nutritional risk categories. An appropriate model selection process and over- and under-sampling procedures were needed so that our prediction models can correctly identify at-risk individuals with minimalbias towards the majority class. Furthermore, as none of the students had a professional medical background, the result of the symptom grouping tables may not have been entirely accurate. From the dashboard design point of view, more drill-down and interactive features needed to have been provided to allow personalised malnutrition risk prediction and deeper decision-support information for healthcare professionals. Another significant engineering challenge was getting all the diverse components, i.e., the Python backend, the machine learning models as well as the Power BI dashboard, into a single containerized application.

The major output of this capstone project contributes to both main practice and technical support. Practically, it provides a tool to connect raw data with actionable (potentially clinical) insights for aged care professionals. The Platform allows for nurses or care staff to track health trends and proactively identify residents who may need additional support. Technically, the project provides an end-to-end data pipeline on how to use machine learning and data visualisation in a healthcare-specific context effectively. The production of a recall-optimised prediction model at a single specific and imbalanced dataset is a beneficial case study to draw upon for future studies in the local research community.

7.3 Project Limitations

Despite the successful implementation, the project has several limitations. The first limitation relates to the data, since the models were trained on a single dataset, the applicability of the predictive models to other patient populations with different demographic or clinical characteristics is not guaranteed. Furthermore, because the dataset used is static, the platform is not suitable for real-time monitoring but rather for retrospective analysis and trend identification.

From a machine learning perspective, the performance of the models presents certain constraints. The symptom clustering model produced a low silhouette score, indicating that the identified patient groups were not highly distinct and had significant overlap which limits the clinical utility of these groupings for assigning specific care protocols. In addition, the risk prediction model is also designed to have a high recall rate to reduce missing cases, but this means that it will face a trade-off with precision and accuracy. This could lead to a higher rate of false positives, which means that certain low-risk patients might be falsely flagged. If this happens in a clinical context without further refinement, it could lead to alert fatigue. On a system level, the platform lacks critical features for real-world deployment, including robust security protocols, user authentication, audit trails for data changes, and direct automated integration with live electronic health record (EHR) systems. Therefore, the system in its current state should be viewed as a functional prototype rather than a clinical tool.

8 Further Development and Conclusion

Further Development
Aging populations around the world are creating historic needs for advanced aged care solutions. Although this project has a successful prototype development, some limitations inform future enhancement priorities. The models were trained on a single research dataset, given the database accuracy and bias, limiting generalizability across diverse aged care populations, Through this project, team members realised that temporal modelling techniques could be applied to model symptom progression over time and improve the accuracy of prediction more than by just taking snapshots. The symptom clustering model must be tuned with other algorithms better suited for overlapping clinical phenotypes, while the risk prediction model's design for high recall can be enhanced with dynamic thresholding that optimizes for sensitivity against operational alert management.

Clinical readiness is achieved by the integration of automated Machine Learning Operations (MLOps) pipelines for continuous model monitoring, performance validation, and regular retraining as new data accumulates (Paleyes et al., 2022). Interoperability standards-based integration with electronic health records would harness the isolated prototype into a built-in workflow system, supporting bi-directional data exchange and real-world effect measurement. These advancements would position the system for regulatory approval pathways and effective deployment in the expanding Australian local aged healthcare field.

Conclusion
This capstone project aims to address the concern of fragmented health data in aged healthcare facilities, with scattered and underused dementia data. Our team provided an integrated, automated data-driven health system in the form of handling data processing pipelines, machine learning models for risk assessment and symptom clustering, and interactive Power BI reports that transform comprehensive datasets into user-friendly insights for clinical teams to monitor residents' statuses. This platform allows population-level health tracking, resident-level tracking, and evidence-based intervention suggestions via a containerized, modular architecture. While there are some current limitations of single-dataset training, static data processing, and non-real-time EHR merging may bring some difficulties throughout the whole development, In the future continuous development, it will prioritize MLOps automation, temporal modelling, and clinical workflow integration, which will make this platform an extension to benefit Australia's local healthcare centres, and finally enable proactive, data-informed dementia care delivery.

References

Battineni, G., Chintalapudi, N. and Amenta, F. (2019), 'Machine learning in medicine: performance calculation of dementia prediction by support vector machines (svm). inform med unlocked 16: 100200'.

Clarke, H. (2023), 'Benefits of modular architecture: Moving from monolithic to modular'.
URL: *https://www.harrisonclarke.com/blog/benefits-of-modular-architecture-moving-from-monolithic-to-modular*

Coiera, E., Chan, A., Brooke-Cowden, K., Rahimi-Ardabili, H., Halim, N. and Tufanaru, C. (2025), 'Clinical and economic impact of digital dashboards on hospital inpatient care: a systematic review', JAMIA open **8**(4), ooaf078.

Docker (2025), 'What is a container?'.
URL: *https://www.docker.com/resources/what-container/*

Dowding, D., Randell, R., Gardner, P., Fitzpatrick, G., Dykes, P., Favela, J., Hamer, S., Whitewood-Moores, Z., Hardiker, N., Borycki, E. et al. (2015), 'Dashboards for improving patient care: review of the literature', International journal of medical informatics **84**(2), 87–100.

GeeksforGeeks (2024), 'Understanding modular architecture in mern'.
URL: *https://www.geeksforgeeks.org/mern/understanding-modular-architecture-in-mern/*

GeeksforGeeks (2025), 'What is software testing?'.
URL: *https://www.geeksforgeeks.org/software-testing/software-testing-basics*

Javeed, A., Dallora, A. L., Berglund, J. S., Ali, A., Ali, L. and Anderberg, P. (2023), 'Machine learning for dementia prediction: a systematic review and future research directions', Journal of medical systems **47**(1), 17.

Kavitha, C., Mani, V., Srividhya, S., Khalaf, O. I. and Tavera Romero, C. A. (2022), 'Early-stage alzheimer's disease prediction using machine learning models', Frontiers in public health **10**, 853294.

Lampe, D., Grosser, J., Grothe, D., Aufenberg, B., Gensorowsky, D., Witte, J. and Greiner, W. (2024), 'How intervention studies measure the effectiveness of medication safety-related clinical decision support systems in primary and long-term care: a systematic review', BMC medical informatics and decision making **24**(1), 188.

Lokesh, G. (2019), 'What is rest – learn to create timeless rest apis'.
URL: *https://restfulapi.net/*

Ludlow, K., Westbrook, J., Jorgensen, M., Lind, K. E., Baysari, M. T., Gray, L. C., Day, R. O., Ratcliffe, J., Lord, S. R., Georgiou, A. et al. (2021), 'Co-designing a dashboard of predictive analytics and decision support to drive care quality and client outcomes in aged care: a mixed-method study protocol', BMJ open **11**(8), e048657.

Paleyes, A., Urma, R.-G. and Lawrence, N. D. (2022), 'Challenges in deploying machine learning: a survey of case studies', ACM computing surveys **55**(6), 1–29.

Rochin, M. A. E., Gutierrez-Garcia, J. O., Rosales, J.-H. and Rodriguez, L.-F. (2021), 'Design and evaluation of a dashboard to support the comprehension of the progression of patients with dementia in day centers', International Journal of Medical Informatics **156**, 104617.

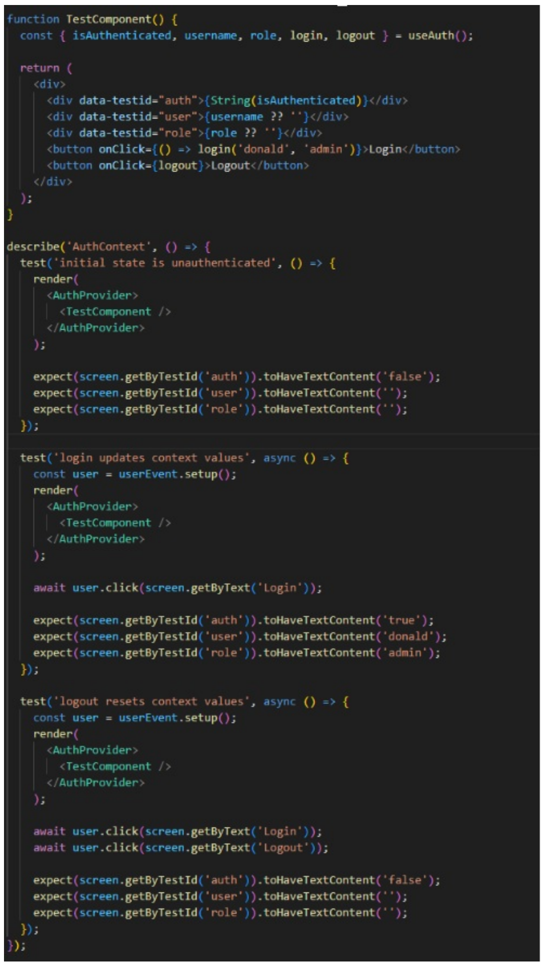
Saliba, D., Ladd, H. and Konetzka, R. (2023), ‘Dementia care is widespread in us nursing homes; facilities with the most dementia’, Health Affairs **42**, 6.

Simmons, L. (2022), ‘The difference between front-end vs. back-end’.
URL: *https://www.computerscience.org/bootcamps/resources/frontend-vs-backend/*

Velazquez, M., Lee, Y. and Initiative, A. D. N. (2021), ‘Random forest model for feature-based alzheimer’s disease conversion prediction from early mild cognitive impairment subjects’, Plos one **16**(4), e0244773.

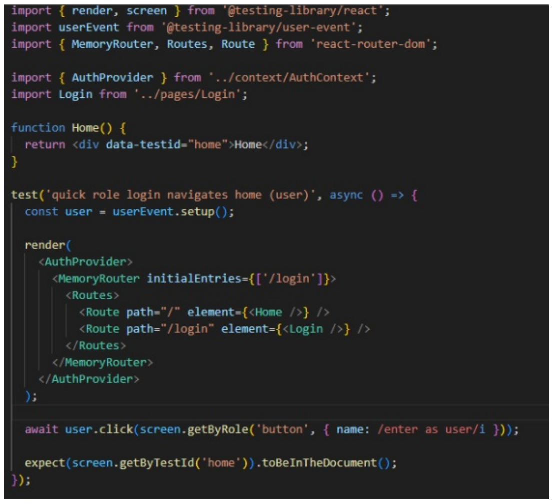
Walling, A. M., Pevnick, J., Bennett, A. V., Vydiswaran, V. and Ritchie, C. S. (2023), ‘Dementia and electronic health record phenotypes: a scoping review of available phenotypes and opportunities for future research’, Journal of the American Medical Informatics Association **30**(7), 1333–1348.

Appendices



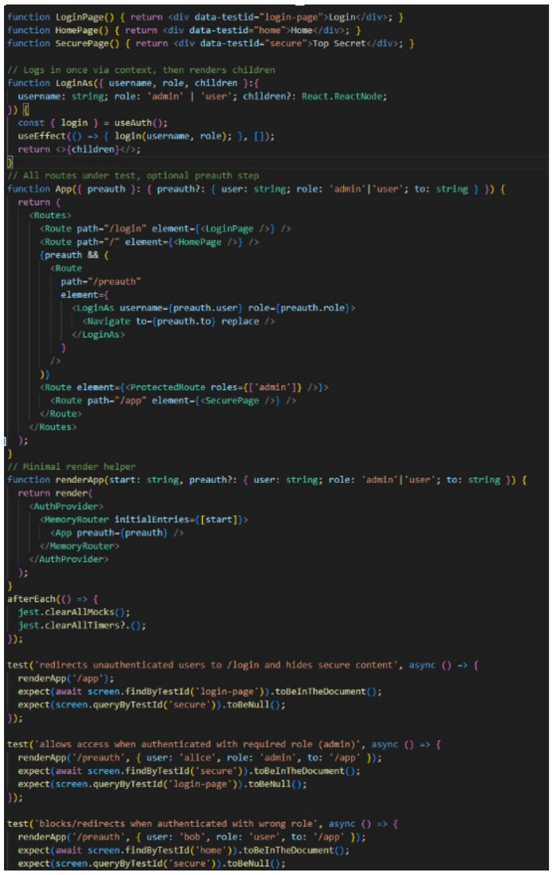
```
function TestComponent() {
  const { isAuthenticated, username, role, login, logout } = useAuth();

  return (
    <div>
      <div data-testid="auth">{String(isAuthenticated)}</div>
      <div data-testid="user">{username ?? ''}</div>
      <div data-testid="role">{role ?? ''}</div>
      <button onClick={() => login('donald', 'admin')}>Login</button>
      <button onClick={logout}>Logout</button>
    </div>
  );
}

describe('AuthContext', () => {
  test('initial state is unauthenticated', () => {
    render(
      <AuthProvider>
        <TestComponent />
      </AuthProvider>
    );

    expect(screen.getByTestId('auth')).toHaveTextContent('false');
    expect(screen.getByTestId('user')).toHaveTextContent('');
    expect(screen.getByTestId('role')).toHaveTextContent('');
  });

  test('login updates context values', async () => {
    const user = userEvent.setup();
    render(
      <AuthProvider>
        <TestComponent />
      </AuthProvider>
    );

    await user.click(screen.getByText('Login'));

    expect(screen.getByTestId('auth')).toHaveTextContent('true');
    expect(screen.getByTestId('user')).toHaveTextContent('donald');
    expect(screen.getByTestId('role')).toHaveTextContent('admin');
  });

  test('logout resets context values', async () => {
    const user = userEvent.setup();
    render(
      <AuthProvider>
        <TestComponent />
      </AuthProvider>
    );

    await user.click(screen.getByText('Login'));
    await user.click(screen.getByText('Logout'));

    expect(screen.getByTestId('auth')).toHaveTextContent('false');
    expect(screen.getByTestId('user')).toHaveTextContent('');
    expect(screen.getByTestId('role')).toHaveTextContent('');
  });
});
```

Figure 1: AuthContext.test



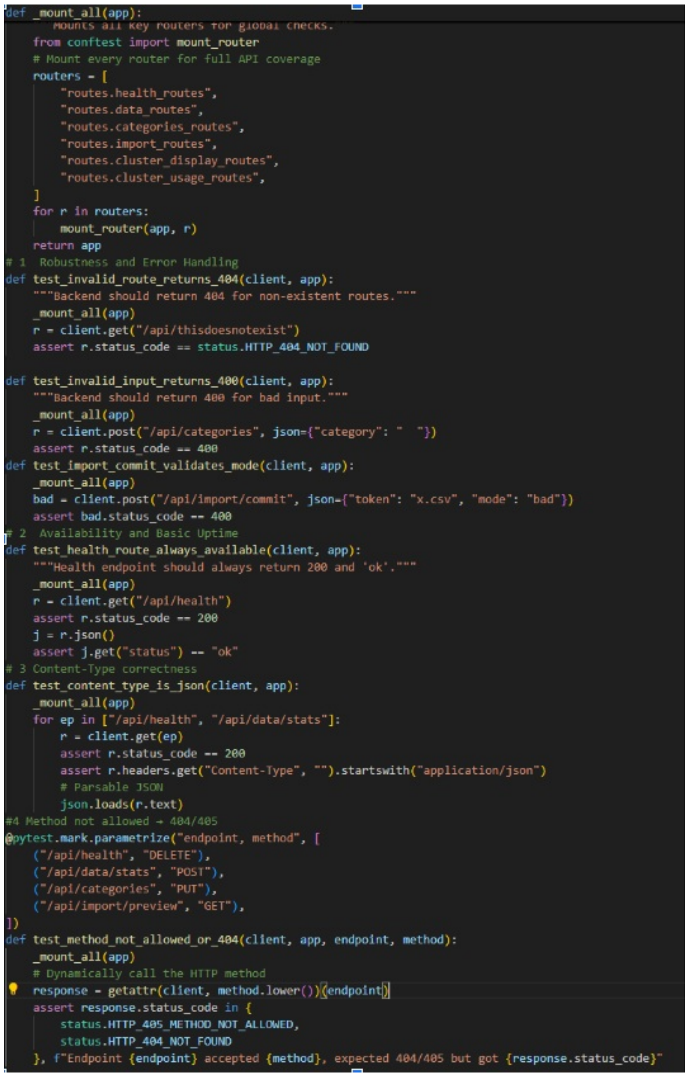
```
import { render, screen } from '@testing-library/react';
import userEvent from '@testing-library/user-event';
import { MemoryRouter, Routes, Route } from 'react-router-dom';

import { AuthProvider } from '../context/AuthContext';
import Login from '../pages/Login';

function Home() {
  return <div data-testid="home">Home</div>;
}

test('quick role login navigates home (user)', async () => {
  const user = userEvent.setup();

  render(
    <AuthProvider>
      <MemoryRouter initialEntries={['/login']}>
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/login" element={<Login />} />
        </Routes>
      </MemoryRouter>
    </AuthProvider>
  );

  await user.click(screen.getByRole('button', { name: /enter as user/i }));

  expect(screen.getByTestId('home')).toBeInTheDocument();
});
```

Figure 2: login.test



```
function LoginPage() { return <div data-testid="login-page">Login</div>; }
function HomePage() { return <div data-testid="home">Home</div>; }
function SecurePage() { return <div data-testid="secure">Top Secret</div>; }

// Logs in once via context, then renders children
function LoginAs({ username, role, children }:{
  username: string; role: 'admin' | 'user'; children?: React.ReactNode;
}) {
  const { login } = useAuth();
  useEffect(() => { login(username, role); }, []);
  return <>{children}</>;
}
// All routes under test, optional preauth step
function App({ preauth }: { preauth?: { user: string; role: 'admin'|'user'; to: string } }) {
  return (
    <Routes>
      <Route path="/login" element={<LoginPage />} />
      <Route path="/" element={<HomePage />} />
      {preauth && (
        <Route
          path="/preauth"
          element={
            <LoginAs username={preauth.user} role={preauth.role}>
              <Navigate to={preauth.to} replace />
            </LoginAs>
          }
        />
      )}
      <Route element={<ProtectedRoute roles={['admin']} />}>
        <Route path="/app" element={<SecurePage />} />
      </Route>
    </Routes>
  );
}
// Minimal render helper
function renderApp(start: string, preauth?: { user: string; role: 'admin'|'user'; to: string }) {
  return render(
    <AuthProvider>
      <MemoryRouter initialEntries={[start]}>
        <App preauth={preauth} />
      </MemoryRouter>
    </AuthProvider>
  );
}
afterEach(() => {
  jest.clearAllMocks();
  jest.clearAllTimers?.();
});

test('redirects unauthenticated users to /login and hides secure content', async () => {
  renderApp('/app');
  expect(await screen.findByTestId('login-page')).toBeInTheDocument();
  expect(screen.queryByTestId('secure')).toBeNull();
});

test('allows access when authenticated with required role (admin)', async () => {
  renderApp('/preauth', { user: 'alice', role: 'admin', to: '/app' });
  expect(await screen.findByTestId('secure')).toBeInTheDocument();
  expect(screen.queryByTestId('login-page')).toBeNull();
});

test('blocks/redirects when authenticated with wrong role', async () => {
  renderApp('/preauth', { user: 'bob', role: 'user', to: '/app' });
  expect(await screen.findByTestId('home')).toBeInTheDocument();
  expect(screen.queryByTestId('secure')).toBeNull();
```

Figure 3: Protect Route.test



```python
def _mount_all(app):
    Mounts all key routers for global checks.
    from conftest import mount_router
    # Mount every router for full API coverage
    routers = [
        "routes.health_routes",
        "routes.data_routes",
        "routes.categories_routes",
        "routes.import_routes",
        "routes.cluster_display_routes",
        "routes.cluster_usage_routes",
    ]
    for r in routers:
        mount_router(app, r)
    return app
# 1  Robustness and Error Handling
def test_invalid_route_returns_404(client, app):
    """Backend should return 404 for non-existent routes."""
    _mount_all(app)
    r = client.get("/api/thisdoesnotexist")
    assert r.status_code == status.HTTP_404_NOT_FOUND

def test_invalid_input_returns_400(client, app):
    """Backend should return 400 for bad input."""
    _mount_all(app)
    r = client.post("/api/categories", json={"category": "  "})
    assert r.status_code == 400
def test_import_commit_validates_mode(client, app):
    _mount_all(app)
    bad = client.post("/api/import/commit", json={"token": "x.csv", "mode": "bad"})
    assert bad.status_code == 400
# 2  Availability and Basic Uptime
def test_health_route_always_available(client, app):
    """Health endpoint should always return 200 and 'ok'."""
    _mount_all(app)
    r = client.get("/api/health")
    assert r.status_code == 200
    j = r.json()
    assert j.get("status") == "ok"
# 3 Content-Type correctness
def test_content_type_is_json(client, app):
    _mount_all(app)
    for ep in ["/api/health", "/api/data/stats"]:
        r = client.get(ep)
        assert r.status_code == 200
        assert r.headers.get("Content-Type", "").startswith("application/json")
        # Parsable JSON
        json.loads(r.text)
#4 Method not allowed → 404/405
@pytest.mark.parametrize("endpoint, method", [
    ("/api/health", "DELETE"),
    ("/api/data/stats", "POST"),
    ("/api/categories", "PUT"),
    ("/api/import/preview", "GET"),
])
def test_method_not_allowed_or_404(client, app, endpoint, method):
    _mount_all(app)
    # Dynamically call the HTTP method
    response = getattr(client, method.lower())(endpoint)
    assert response.status_code in {
        status.HTTP_405_METHOD_NOT_ALLOWED,
        status.HTTP_404_NOT_FOUND
    }, f"Endpoint {endpoint} accepted {method}, expected 404/405 but got {response.status_code}"
```

Figure 4: risk.test

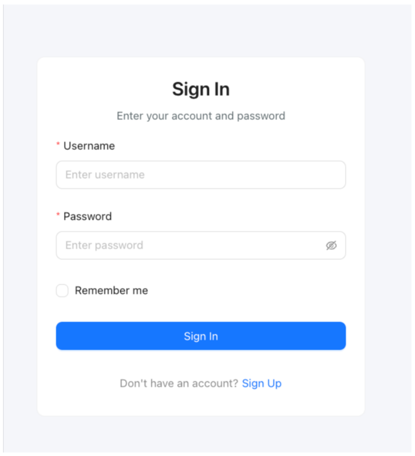
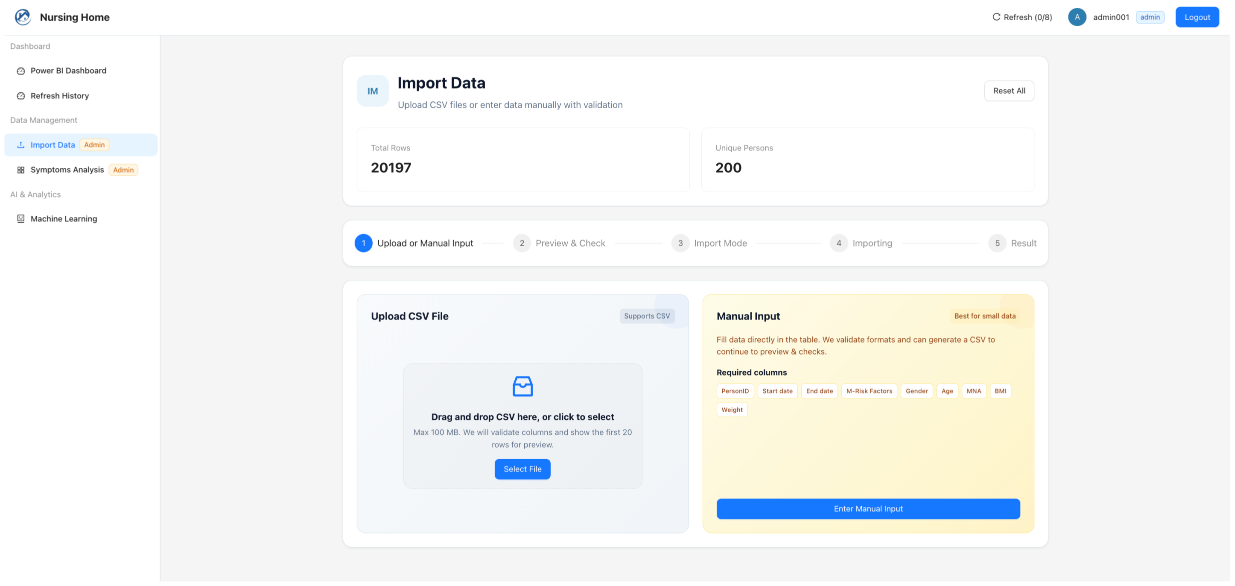


Figure 5: User Login



IM **Import Data**

Upload CSV files or enter data manually with validation

Total Rows	Unique Persons
20197	200

1 Upload or Manual Input — 2 Preview & Check — 3 Import Mode — 4 Importing — 5 Result

Upload CSV File Supports CSV

Drag and drop CSV here, or click to select

Max 100 MB. We will validate columns and show the first 20 rows for preview.

Select File

Manual Input Best for small data

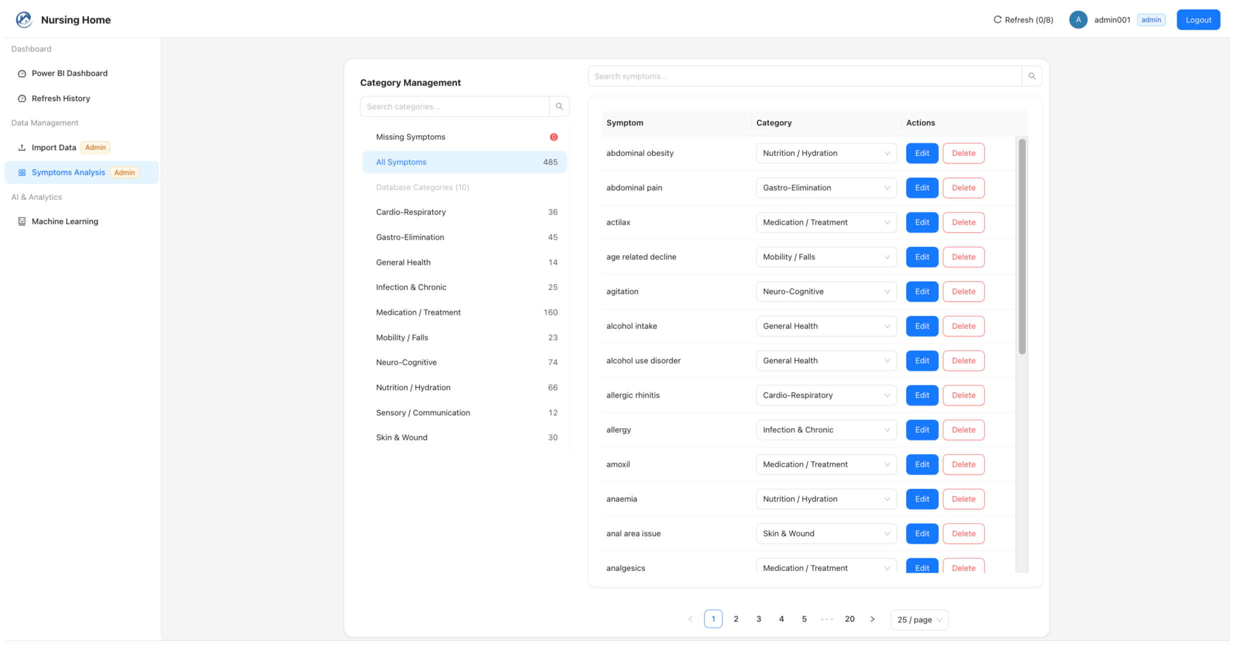
Fill data directly in the table. We validate formats and can generate a CSV to continue to preview & checks.

Required columns

PersonID, Start date, End date, M-Risk Factors, Gender, Age, MNA, BMI, Weight

Enter Manual Input

Figure 6: Data Import Function



Category Management

Missing Symptoms	0
All Symptoms	485
Database Categories (10)	
Cardio-Respiratory	36
Gastro-Elimination	45
General Health	14
Infection & Chronic	25
Medication / Treatment	160
Mobility / Falls	23
Neuro-Cognitive	74
Nutrition / Hydration	66
Sensory / Communication	12
Skin & Wound	30

Symptom	Category	Actions
abdominal obesity	Nutrition / Hydration	Edit Delete
abdominal pain	Gastro-Elimination	Edit Delete
actilax	Medication / Treatment	Edit Delete
age related decline	Mobility / Falls	Edit Delete
agitation	Neuro-Cognitive	Edit Delete
alcohol intake	General Health	Edit Delete
alcohol use disorder	General Health	Edit Delete
allergic rhinitis	Cardio-Respiratory	Edit Delete
allergy	Infection & Chronic	Edit Delete
amoxil	Medication / Treatment	Edit Delete
anaemia	Nutrition / Hydration	Edit Delete
anal area issue	Skin & Wound	Edit Delete
analgesics	Medication / Treatment	Edit Delete

Figure 7: Symptoms Analysis and Management