CSE381:Introduction to Machine Leaning
Spring 2025
Stroke Prediction Document

Presented to : Dr.Mahmoud Khalil , Eng.Engy Mahmoud

Table 1: Team members

ID	Team Member
21P0211	Yasmina Nasser Hamam
21P0173	Monica Hany Makram Derias
2100588	Zeynat Haytham Farouk Ghallab

Contents

1 Introduction

Stroke is a leading cause of death and disability worldwide. Early identification of individuals at risk is crucial for timely medical intervention, which can significantly reduce the severity of stroke events and improve patient outcomes. Machine learning (ML) offers promising tools for early stroke prediction, enabling the analysis of complex datasets to uncover patterns that may not be evident through traditional statistical methods.

This project aims to develop ML models to predict the likelihood of stroke in patients based on various clinical and lifestyle features. The dataset utilized includes attributes such as age, gender, hypertension status, heart disease presence, marital status, work type, residence type, average glucose level, body mass index (BMI), and smoking status. By applying supervised learning algorithms like Naïve Bayes and Support Vector Machines (SVM), the project seeks to accurately classify patients as at risk or not at risk of stroke. Additionally, unsupervised learning methods, including K-Means and Hierarchical Clustering, are employed to identify natural groupings within the data, potentially revealing subpopulations with elevated stroke risk.

The overarching goal is to harness the predictive power of ML to facilitate early detection of stroke risk, thereby contributing to preventive healthcare strategies and resource optimization.

2 Data Exploration and Visualization,

A thorough data exploration and visualization process was conducted to clean the dataset, understand variable distributions, detect anomalies, and prepare for effective modeling. The key steps are summarized as follows:

- **Duplicate Handling and Overview:**
  - No duplicates were found
  - Dataset structure and feature types were reviewed using `df.info()`.
- **Missing Data and Categorical Summary:**
  - Missing values were assessed with `df.isna().sum()` and visualized using a heatmap.
  - Categorical feature distributions were reviewed using `value_counts()` to detect imbalance or rare categories.
- **Descriptive Statistics:**
  - Summary statistics for numerical features were generated using `df.describe()` to evaluate spread and central tendency.
- **Univariate Visualizations:**
  - Created subplots of all features using histograms and count plots to assess distribution and class frequency.

- **Feature-Target Relationships:**
  - Count plots and box plots were used to visualize relationships between each feature and the target `stroke`.
  - Binary features like `heart_disease` and `hypertension` were visualized in relation to stroke using grouped bar plots.
- **Scatter Plot Analysis:**
  - Bivariate scatter plots were created for `age`, `avg_glucose_level`, and `bmi`, colored by stroke incidence.

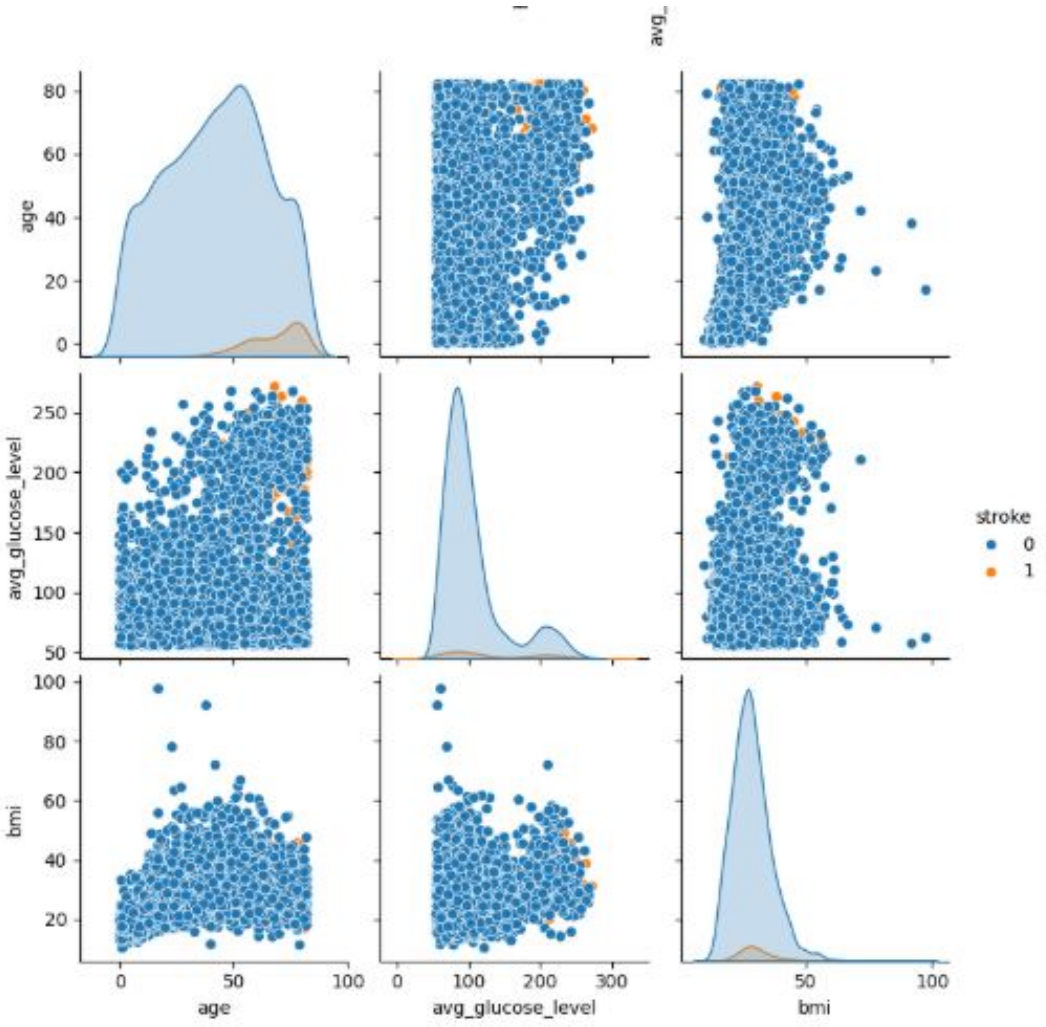


Figure 1: scatter plots of different feature combinations

- **Correlation Analysis:**
  - A correlation matrix heatmap was plotted to understand feature interdependencies.

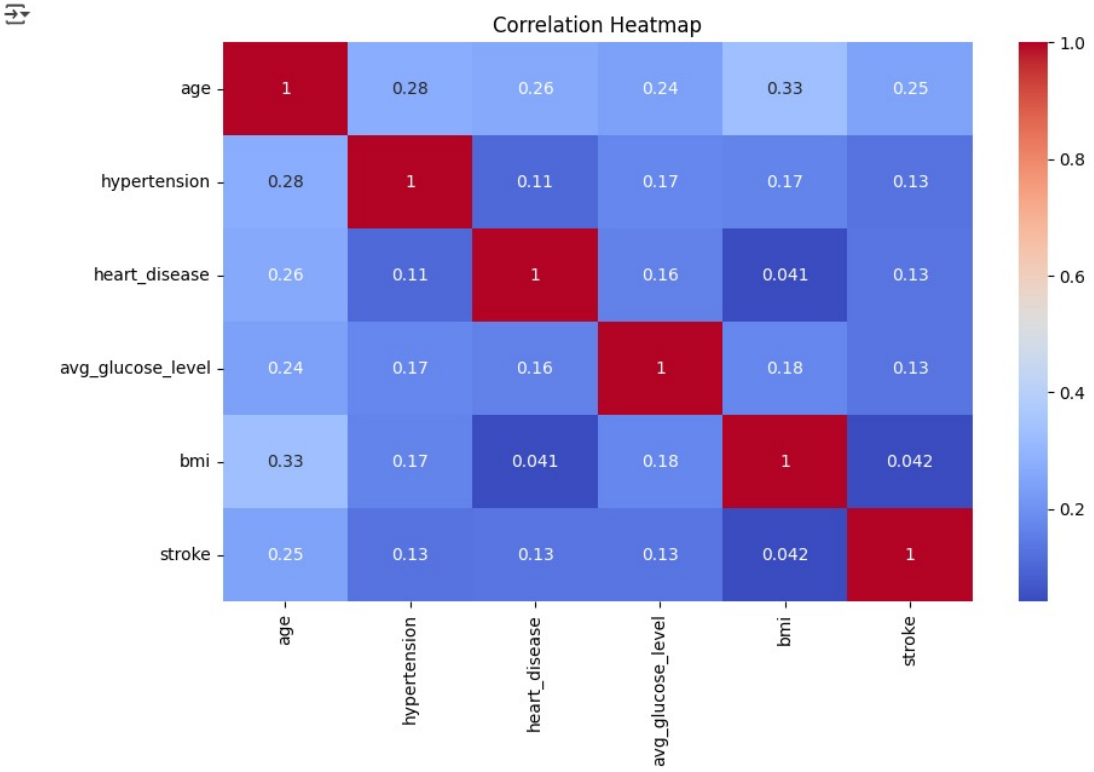


Figure 2: correlation matrix

  - Features with low correlation to stroke (e.g., `id`, `gender`) were identified for potential exclusion.

- **Dimensionality Reduction Techniques:**
  - The dataset was preprocessed (e.g., dropping `id` and `gender`, imputing missing `bmi`) and scaled using `StandardScaler`.
  - Three projection techniques were applied:
    - **PCA (Principal Component Analysis)** to reduce dimensions while preserving variance.
    - **LDA (Linear Discriminant Analysis)** to maximize class separability.
    - **t-SNE (t-distributed Stochastic Neighbor Embedding)** for non-linear feature space visualization.
  - The results of PCA, LDA, and t-SNE were visualized in 2D and 3D scatter plots to assess separability of stroke vs. non-stroke cases.

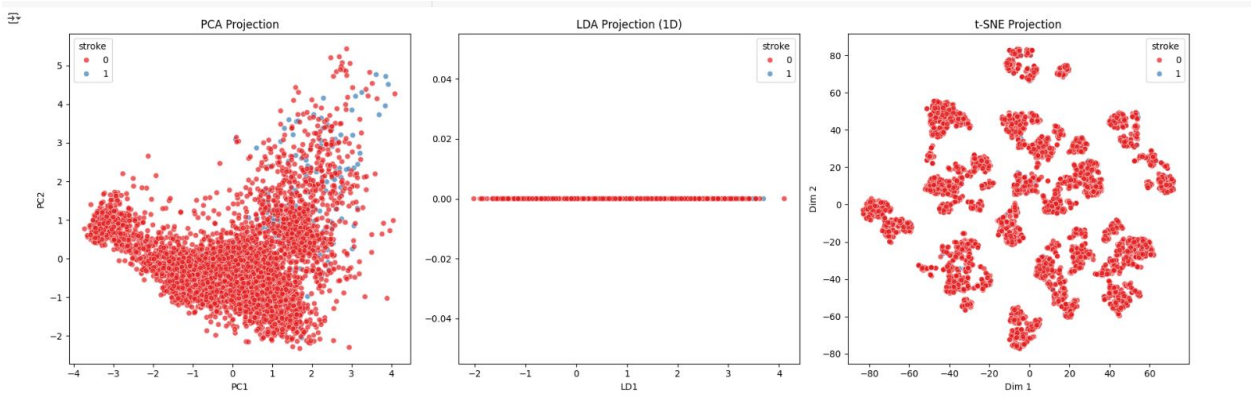


Figure 3: PCA,LDA AND t-sne plots

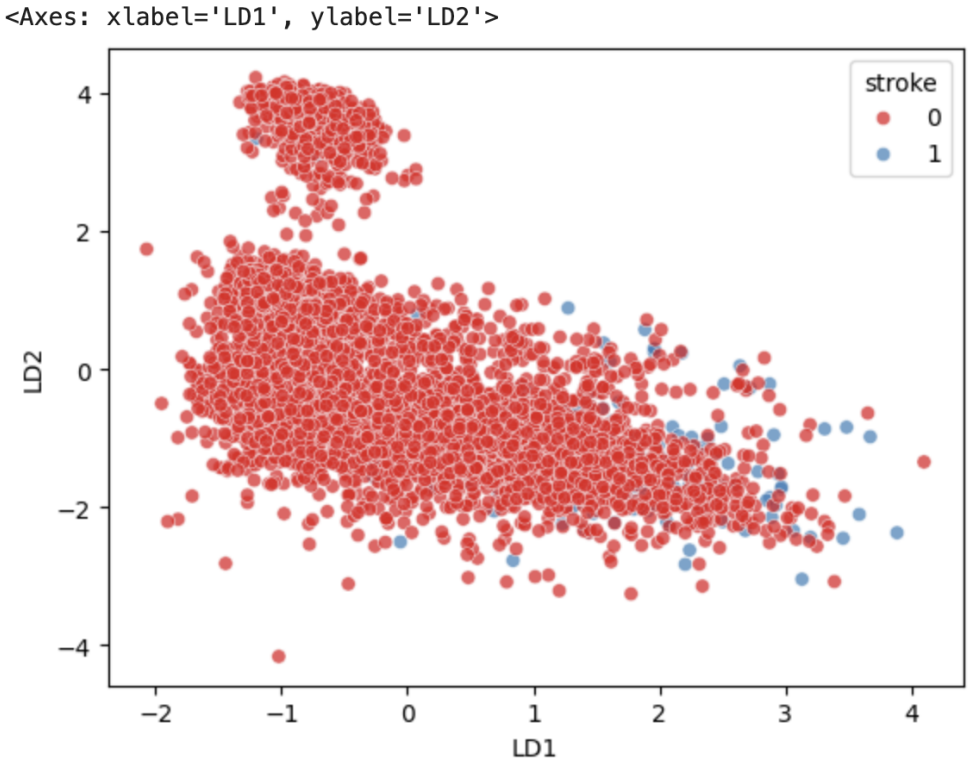


Figure 4: LDA in 2D plot

2.1 Observations

After thorough data analysis, visualization, and dimensionality reduction, the following key observations were made:

- The dataset contained **no duplicate rows**, indicating consistent data entry in terms of redundancy.
- The `bmi` column had missing values, while the `gender` column included an infrequent 'Other' category.
- The `stroke` variable is highly imbalanced, with far fewer positive cases, highlighting the need for techniques to address class imbalance.
- Stroke cases are more prevalent among individuals who are older, have higher average glucose levels, and higher BMI values.
- Visual analysis of binary features (`heart_disease`, `hypertension`) revealed meaningful variation between stroke and non-stroke groups.
- Features like `id` and `gender` were found to have weak correlation with the target variable and were excluded during preprocessing.
- Potential data quality issues such as zero values in the `age` column were identified and may require domain-driven imputation or filtering.
- Label encoding and standard scaling were applied to ensure categorical consistency and to prepare for dimensionality reduction.
- Dimensionality reduction results:
  - **PCA (Principal Component Analysis)** preserved variance but showed substantial class overlap between stroke and non-stroke groups.
  - **LDA (Linear Discriminant Analysis)** provided a one-dimensional projection with only slight class separation, indicating weak linear discriminative power.
  - **t-SNE (t-distributed Stochastic Neighbor Embedding)** showed more apparent clustering but still confirmed overlapping class boundaries.
- These findings suggest that the classes in this dataset are **not linearly separable** in the current feature space, and more sophisticated modeling techniques or feature engineering may be required to achieve strong predictive performance.

3 Data Cleaning and Preprocessing

3.1 Check for Missing Values

- **Purpose:** Identify missing values in the dataset to understand which features require imputation.
- **Details:**
  - **The count of missing values in each column is calculated.**
  - **The percentage of missing values for each column is also determined to assess their significance.**
  - **Observation: The ‘bmi‘ column contains missing values.**

3.2 Visualizing Missing Data

- **Purpose:** Create a visual representation of missing values in the dataset.
- **Details:** A heatmap is used to display the presence of missing data, providing an overview of null distribution across features.

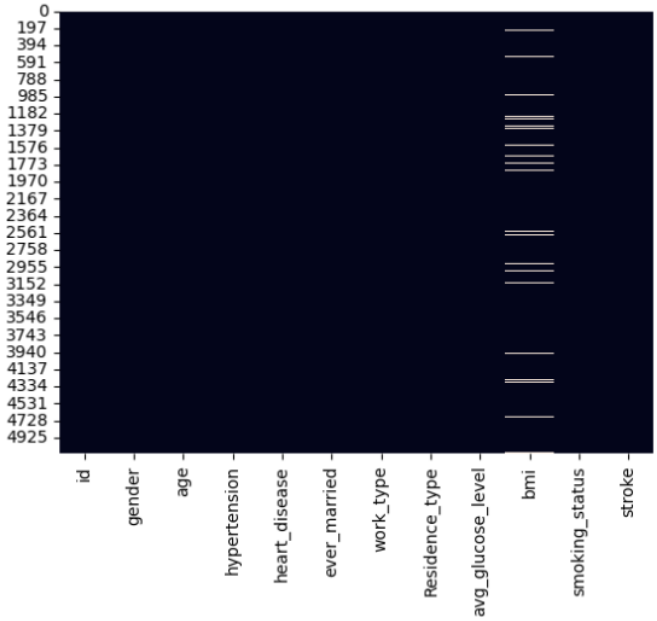


Figure 5: Heatmap showing the missing values in each column

3.3 Handling Missing Values

Upon initial data exploration, we identified that the `bmi` feature contained 201 missing values. To address this, we employed three distinct imputation strategies, each resulting in a separate DataFrame for comparative analysis:

- **Mean Imputation:** Utilized the `SimpleImputer` class from `sklearn.impute` with the `mean` strategy to replace missing `bmi` values with the mean of the existing values.
- **Random Forest Imputation:** Applied a `RandomForestRegressor` from `sklearn.ensemble` to predict missing `bmi` values based on other features, training the model on rows with known `bmi` values.
- **K-Nearest Neighbors (KNN) Imputation:** Implemented the `KNNImputer` from `sklearn.impute` with 5 neighbors to estimate missing `bmi` values by averaging the `bmi` values of the nearest neighbors.

Each imputation method was applied to create three separate DataFrames: `df_mean`, `df_rf`, and `df_knn`. These datasets are to be used in different pipelines, and their performance will be evaluated in subsequent phases to determine the most effective imputation strategy.

3.4 Drop Unnecessary Columns

- **Purpose:**Remove features that do not add value to the analysis or model performance.
- **Details:**Example: The ‘id‘ column is dropped as it is an identifier with no predictive value.

3.5 Checking Duplicate Rows

Duplicate records can distort statistical analyses and degrade model performance by introducing bias. In our dataset, we identified and confirmed the absence of duplicate entries, ensuring the integrity of our data.

3.6 Encoding Categorical Variables

Machine learning models require numerical input, necessitating the conversion of categorical variables into numerical formats. We applied the following encoding techniques:

- **Label Encoding:** For ordinal features, we used `LabelEncoder` from `sklearn.preprocessing` to assign integer labels.
- **One-Hot Encoding:** For nominal features, we applied `pd.get_dummies()` to create binary columns for each category.

3.7 Feature Scaling

Feature scaling standardizes the range of independent variables, ensuring that no single feature dominates due to its scale. We employed two scaling techniques:

- **Standard Scaling:** Transforms features to have a mean of 0 and a standard deviation of 1, suitable for algorithms sensitive to the scale of input data.
- **Min-Max Scaling:** Rescales features to a fixed range, typically [0, 1], which is beneficial for algorithms that rely on distance calculations.

3.8 Handling Class Imbalance with SMOTE

Class imbalance can lead to biased model predictions. To address this, we utilized the Synthetic Minority Over-sampling Technique (SMOTE) to generate synthetic samples for the minority class, balancing the class distribution and enhancing model training.

3.9 Outlier Detection

3.9.1 Method 1: IQR

Outliers can significantly skew machine learning models, especially when data is not normally distributed. To detect and remove outliers in the numerical features `age`, `avg_glucose_level`, and `bmi`, we employed the Interquartile Range (IQR) method:

- Calculated the first quartile (Q1) and third quartile (Q3) for each numerical feature.
- Determined the IQR as $Q3 - Q1$.
- Identified outliers as values outside the range $[Q1 - 1.5 \times \text{IQR}, Q3 + 1.5 \times \text{IQR}]$.

3.9.2 Method 2: Z-score

The Z-score method detects outliers by computing the number of standard deviations a value lies from the mean:

$$\text{Z-score} = \frac{x - \mu}{\sigma},$$

where μ and σ represent the mean and standard deviation of the feature, respectively. Data points with Z-scores less than -3 or greater than 3 were flagged as outliers.

Steps Implemented:

1. Calculate Z-scores for each numerical feature.
2. Identify outliers based on Z-score thresholds.
3. Calculate the percentage of outliers detected.

3.10 Handeling Outliers

3.10.1 Impute By Median

To avoid data loss and reduce the impact of outliers, we used median imputation to fill missing values, as the median is less sensitive to extreme values. The boxplots below illustrate the distribution of the numerical features before and after outlier removal:

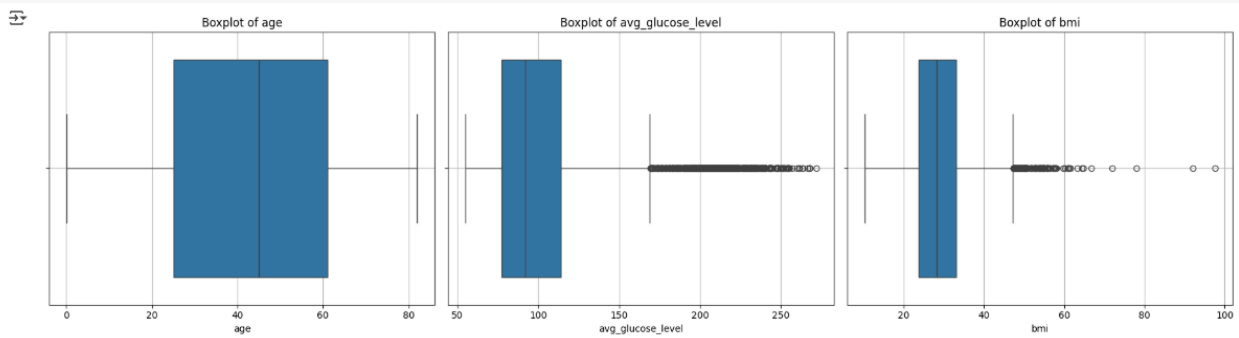


Figure 6: Boxplots of numerical features before outlier removal.

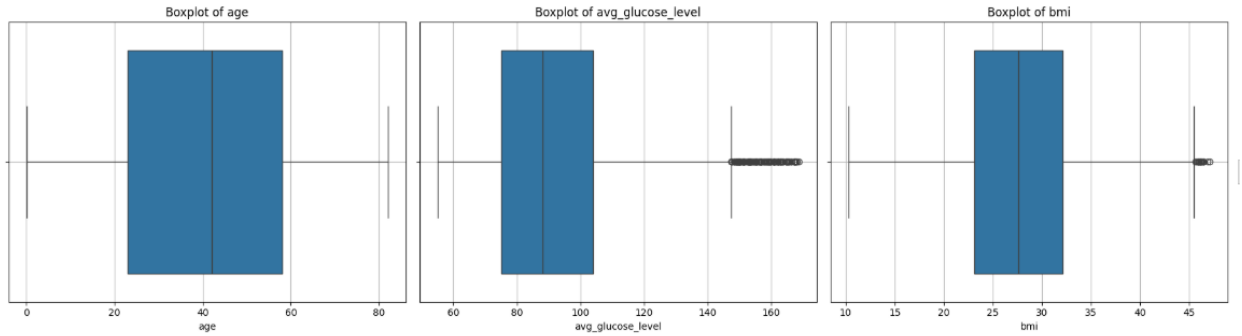


Figure 7: Boxplots of numerical features after outlier removal.

3.11 Impact of Outlier Treatment

To visualize the effects of outlier removal:

1. Box plots were generated to highlight changes in feature ranges.

Findings: Columns with significant skewness showed improved distributions after outlier treatment, demonstrating the efficacy of the IQR method in mitigating the impact of extreme values.

3.12 Conclusion

The cleaned dataset was saved for subsequent analysis. This dataset balances the removal of extreme values while minimizing information loss.

3.13 Data Splitting

To evaluate model performance, we split the dataset into training, validation, and test sets:

- **Training Set:** 70% of the data, used to train the models.
- **Validation Set:** 15% of the data, used to tune model hyperparameters.
- **Test Set:** 15% of the data, used to assess final model performance.

The split was performed using `train_test_split` from `sklearn.model_selection`, the training set was used to train the model, the validation set to tune hyperparameters, and the test set to assess the final model's performance, ensuring a robust evaluation framework.

4 Classifiers (Supervised Learning)

4.1 Decision Tree Classifier

A decision tree is a popular supervised machine learning algorithm used for both classification and regression tasks. It works by splitting the data into branches based on feature values, forming a tree-like structure. Here's a short breakdown:

1. Root Node: Represents the entire dataset and the first feature split.
2. Decision Nodes: Points where the data is split based on a condition.
3. Leaf Nodes: Represent the final output or prediction (class or value).

Each section includes the method's implementation, rationale, and expected outcomes.

4.1.1 Train-Test-validate Split

Overview

The train-test-validate split method involves dividing the dataset into three parts: a training set, a validation set and a testing set. The model is trained on the training set, hyperparameters are tuned on the validation set and evaluated on the testing set.

Implementation

```
from sklearn.model_selection import train_test_split

# Separate features and target
X = df.drop('stroke', axis=1)
y = df['stroke']

# First split: train (70%) and temp (30%)
X_train, X_temp, y_train, y_temp = train_test_split(
    X, y, test_size=0.3, stratify=y, random_state=42)

# Second split: validation (15%) and test (15%) from temp
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, stratify=y_temp, random_state=42)

# Print shapes to verify
print(f"Training set shape: {X_train.shape}, {y_train.shape}")
print(f"Validation set shape: {X_val.shape}, {y_val.shape}")
print(f"Test set shape: {X_test.shape}, {y_test.shape}")
```

4.2 Decision Tree Model Evaluation

This section presents the performance evaluation of a Decision Tree classifier used for stroke prediction. The following metrics were computed:

```
--- Decision Tree ---
              precision    recall  f1-score   support

   No Stroke       0.98      0.65      0.78       729
      Stroke       0.10      0.79      0.18        38

    accuracy                           0.65       767
   macro avg       0.54      0.72      0.48       767
weighted avg       0.94      0.65      0.75       767

Accuracy: 0.6545
```

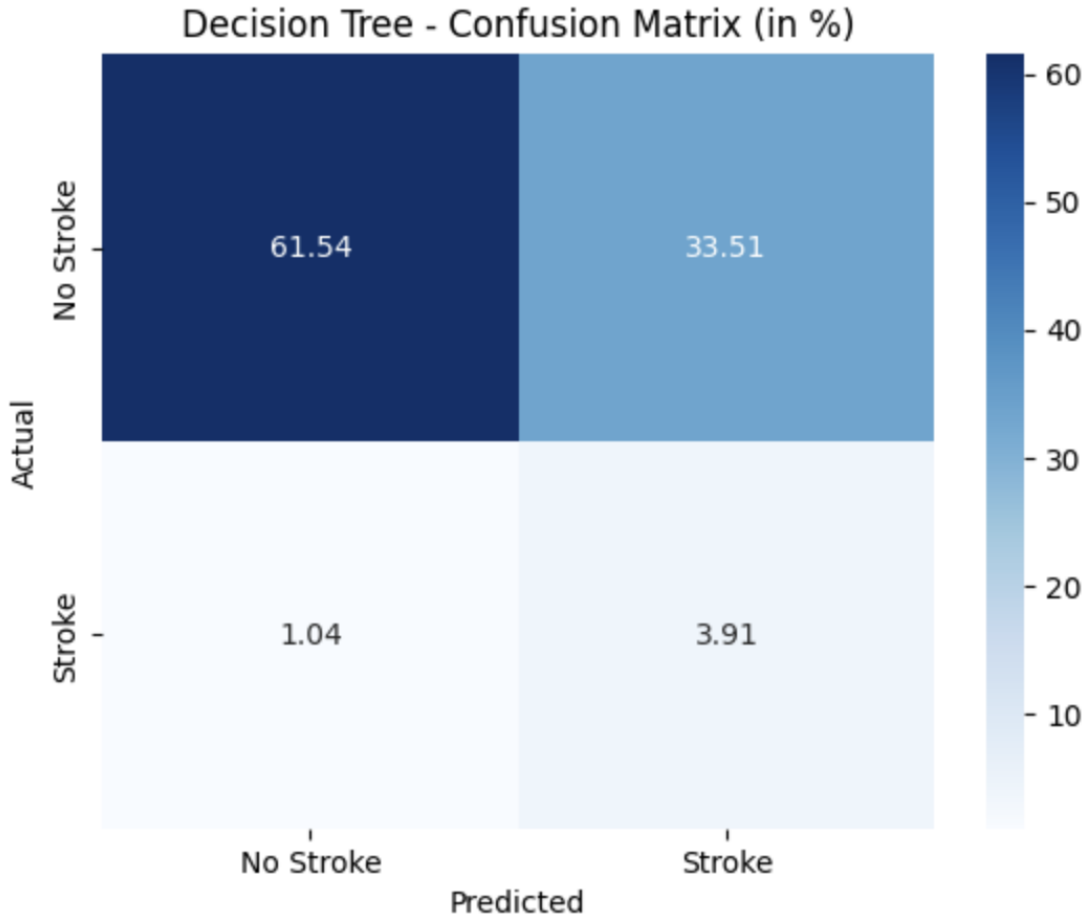


Figure 8: Decision Tree Model Evaluation and Confusion Matrix

4.2.1 Hyperparameter Tuning Results

The following hyperparameters were evaluated:

- **max_depth:** Controls the maximum depth of the tree.
- **min_samples_split:** The minimum number of samples required to split an internal node.
- **min_samples_leaf:** The minimum number of samples required to be at a leaf node.
- **criterion:** The function to measure the quality of a split (e.g., Gini impurity or entropy).

```
=== Final Accuracy Scores on Best Models ===

Decision Tree:
  → Best Params:         {'max_depth': 10, 'min_samples_split': 2}
  → Training Accuracy:   0.8345
  → Validation Accuracy: 0.8355
```

Figure 9: Decision Tree best results from GridSearch

4.3 Naive Bayes Classifier

In stroke prediction, the Naive Bayes classifier uses patient features (like age, hypertension, smoking status) to estimate the probability of a stroke. It assumes feature independence, making it fast and efficient even with limited data. Despite its simplicity, it can provide solid baseline performance for medical risk classification.

4.3.1 Gaussian Naive Bayes

Model Evaluation and Confusion Matrix

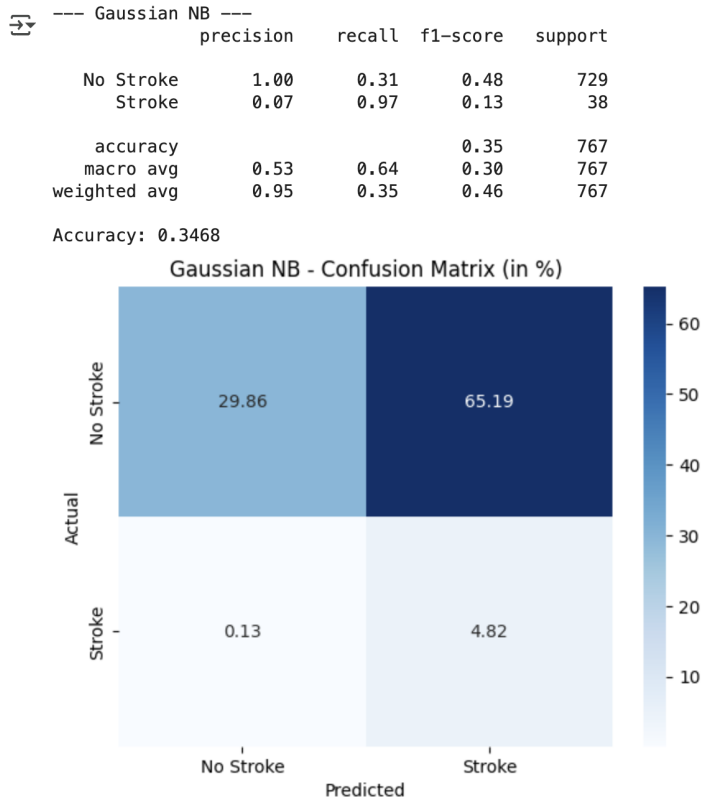


Figure 10: Gaussian Naive Bayes Model Evaluation and Confusion Matrix

Hyperparameter tuning:

```
GaussianNB:
  → Best Params:          {'var_smoothing': 1e-06}
  → Training Accuracy:    0.5006
  → Validation Accuracy: 0.5235
```

Figure 11: Gaussian Naive Bayes Best Results From GridSearch

4.3.2 Benoulli Naive Bayes

Model Evaluation and Confusion Matrix

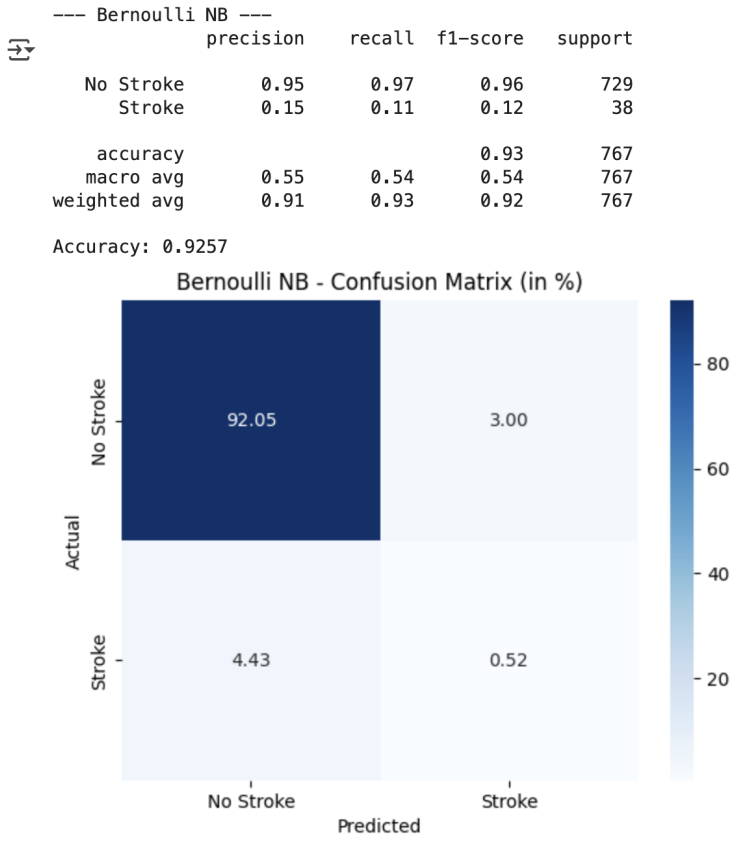


Figure 12: Benoulli Naive Bayes Model Evaluation and Confusion Matrix

Hyperparameter tuning:

```
BernoulliNB:
  → Best Params:          {'alpha': 2.0}
  → Training Accuracy:    0.9306
  → Validation Accuracy: 0.9269
```

Figure 13: Bernoulli Naive Bayes Best Results From GridSearch

Conclusion

Benoulli Naive Bayes proved to be an effective classifier for the given dataset.Future work could involve exploring different classifiers, additional feature engineering, or hyperparameter tuning to further improve results.

4.4 KNN Classifier

In stroke prediction, the KNN classifier identifies patients with similar health profiles (e.g., age, BMI, medical history) and predicts stroke risk based on the outcomes of nearby cases. It doesn't make assumptions about data distribution, making it flexible. However, its performance can be affected by noisy data or irrelevant features.

4.4.1 KNN Model Evaluation and Confusion Matrix

This section presents the performance evaluation of a KNN classifier used for stroke prediction. The following metrics were computed:

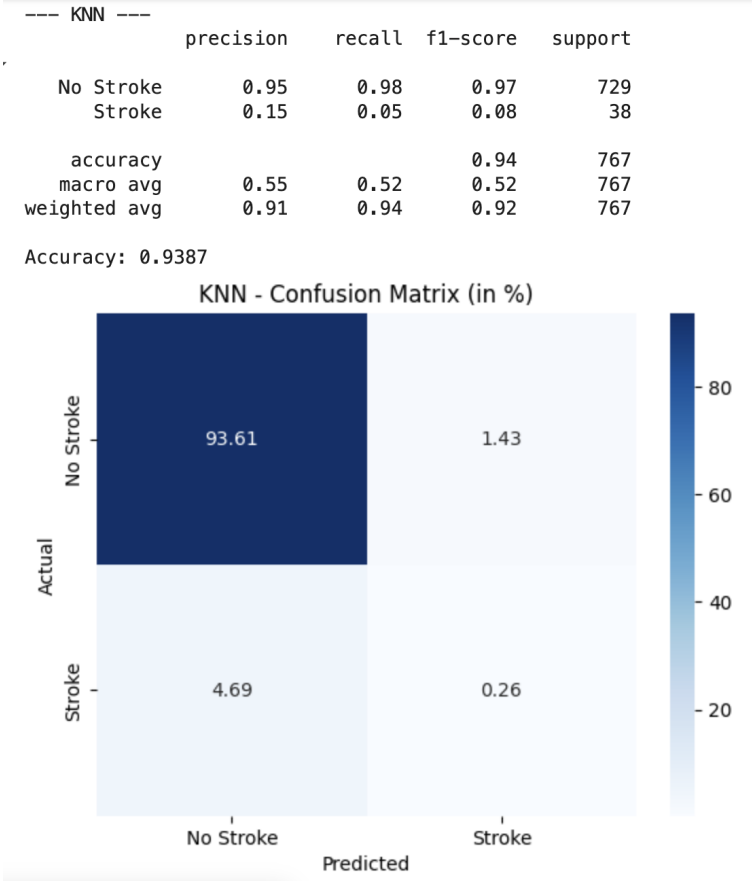


Figure 14: KNN Model Evaluation and Confusion Matrix

4.4.2 Hyperparameter Tuning

The KNN classifier has one key hyperparameter:

- `n_neighbors`: The number of neighbors to consider (k).

A grid search over the following values was conducted:

- `n_neighbors`: $\{1, 3, 5, 7, 9\}$

```
KNN:
  → Best Params:         {'metric': 'euclidean', 'n_neighbors': 10}
  → Training Accuracy:   0.9513
  → Validation Accuracy: 0.9517
```

Figure 15: Hyperparameters Accuracy results

4.5 SVM Classifier

In stroke prediction, the SVM classifier finds the optimal boundary that separates stroke and non-stroke cases using patient features. It is effective in high-dimensional spaces and works well with both linear and non-linear data using kernels. SVM is especially useful when the classes are imbalanced or overlap slightly.

4.6 Confusion Matrix and Model Evaluation

```
--- SVM ---
              precision    recall  f1-score   support

   No Stroke       0.99      0.61      0.76       729
      Stroke       0.10      0.84      0.18        38

    accuracy                           0.63       767
   macro avg       0.54      0.73      0.47       767
weighted avg       0.94      0.63      0.73       767

Accuracy: 0.6258
```

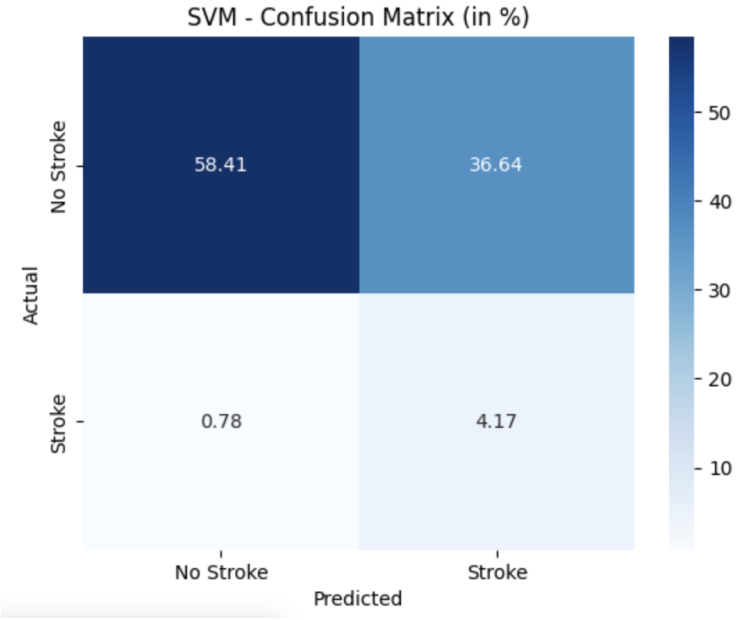


Figure 16: SVM model evaluation and confusion matrix

Hyperparameter Tuning:

```
SVM:
  → Best Params:         {'C': 100, 'gamma': 'scale', 'kernel': 'rbf'}
  → Training Accuracy:   0.9449
  → Validation Accuracy: 0.8890
```

Figure 17: SVM Best Results From GridSearch

5 Commparing Between All Models

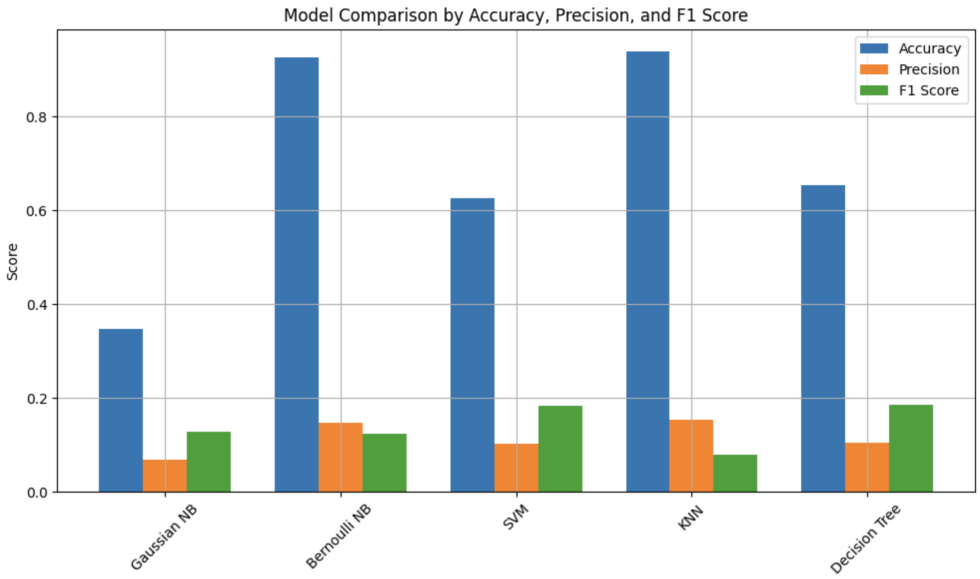


Figure 18: All Models Comparison

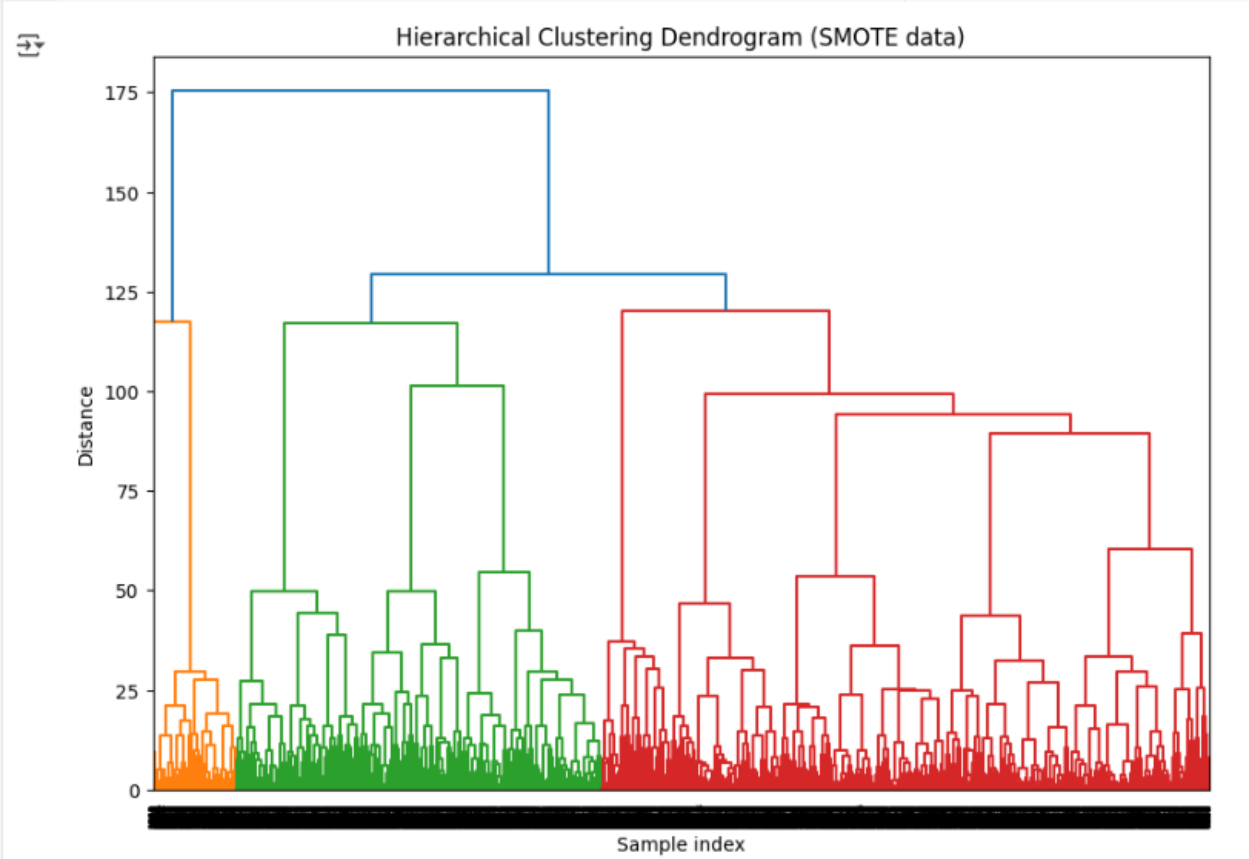
- **Accuracy:**
  - KNN and Bernoulli NB achieved the highest accuracy (approximately 0.93 and 0.91, respectively).
  - Gaussian NB had the lowest accuracy (around 0.34).
  - SVM and Decision Tree showed moderate accuracy (around 0.63 and 0.67, respectively).
- **Precision:**
  - All models exhibited relatively low precision.
  - KNN and Bernoulli NB performed best in terms of precision (around 0.15).
  - Gaussian NB had the lowest precision (below 0.1).
  - Low precision suggests a high number of false positives, which is critical in stroke prediction.
- **F1 Score:**
  - SVM and Decision Tree achieved the highest F1 scores (around 0.18), indicating a good balance between precision and recall.
  - KNN, despite high accuracy, had a low F1 score, implying poor handling of the minority class.
  - Gaussian NB had the weakest performance across all metrics.

- **Conclusion:**
  - High accuracy alone is not sufficient for imbalanced datasets like stroke prediction.
  - SVM and Decision Tree models are better suited due to their balanced performance.
  - Further improvements can be made by using techniques like SMOTE to address class imbalance.

6 Clustering

6.1 Using the smoted training data set only

Hierarchical Clustering




```
Stroke counts per cluster:
hierarchical_cluster  stroke
1                     0          510
                      1           24
2                     0         2892
                      1         3378
dtype: int64

Stroke percentages per cluster:
                      No Stroke (%)  Stroke (%)
hierarchical_cluster
1                         95.505618    4.494382
2                         46.124402   53.875598
```

Figure 19: Hierarchical Clustering Dendrogram (SMOTE data)

The dendrogram in Figure 21 shows the hierarchical clustering structure of the SMOTE-balanced dataset. Two main clusters were identified. The stroke distribution within each cluster is summarized in Table 2 and Table 3.

Table 2: Stroke counts per hierarchical cluster

Cluster	No Stroke	Stroke
1	510	24
2	2892	3378

Table 3: Stroke percentages per hierarchical cluster

Cluster	No Stroke (%)	Stroke (%)
1	95.51	4.49
2	46.12	53.88

Cluster 2 shows a notably higher proportion of stroke cases (53.88%), indicating that the hierarchical clustering successfully grouped a high-risk population.

K-Means Clustering

```
Stroke counts per cluster:
kmeans_cluster  stroke
0               0         1292
                1           28
1               0         2110
                1         3374
dtype: int64

Stroke percentages per cluster:
                No Stroke (%)  Stroke (%)
kmeans_cluster
0                   97.878788    2.121212
1                   38.475565   61.524435
```

Figure 20: K-Means clustering results (SMOTE data)

The K-Means clustering also identified two clusters, with stroke distributions shown in Table 4 and Table 5.

Table 4: Stroke counts per K-Means cluster

Cluster	No Stroke	Stroke
0	1292	28
1	2110	3374

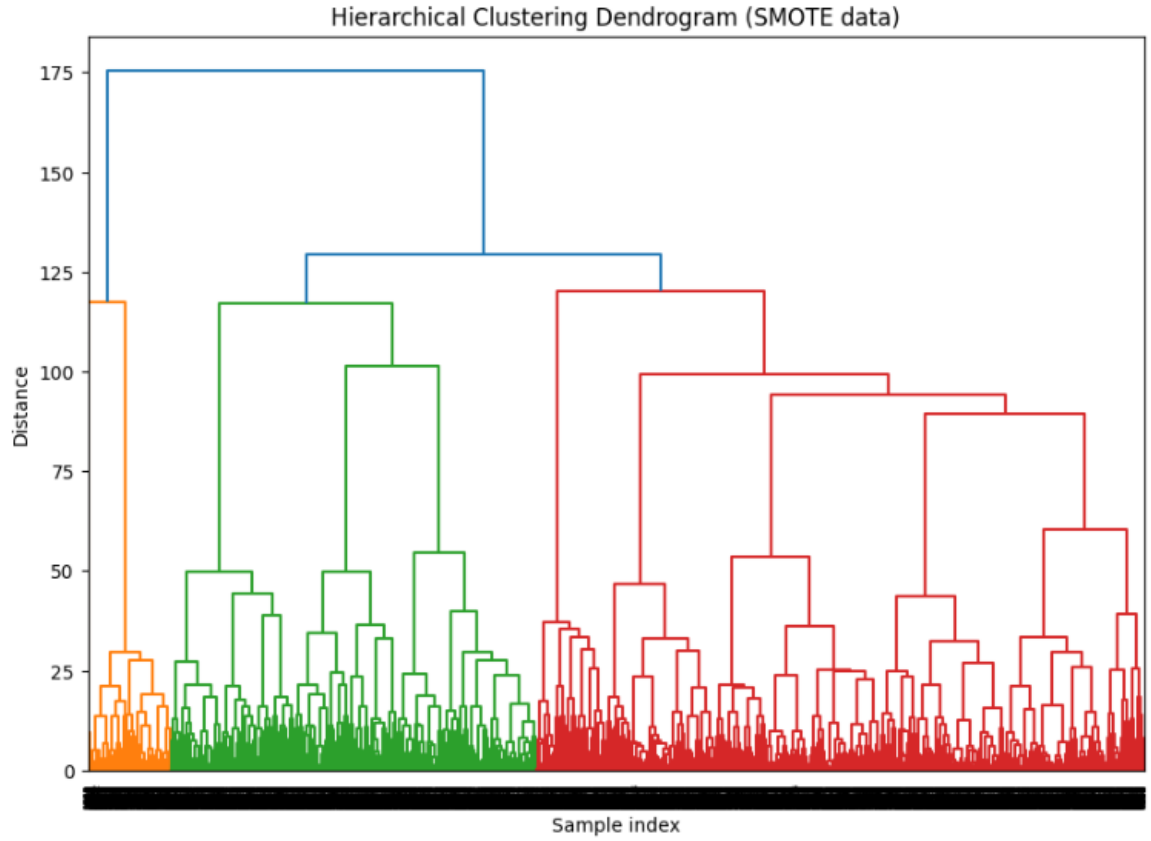
Table 5: Stroke percentages per K-Means cluster

Cluster	No Stroke (%)	Stroke (%)
0	97.88	2.12
1	38.48	61.52

Cluster 1 in K-Means contains a majority of stroke cases (61.52%), again indicating effective separation of high-risk patients.

6.2 Using the full preprocessed data set

Hierarchical Clustering




```
Stroke counts per cluster:
hierarchical_cluster  stroke
1                     0          510
                      1           24
2                     0         2892
                      1         3378
dtype: int64

Stroke percentages per cluster:
                      No Stroke (%)  Stroke (%)
hierarchical_cluster
1                         95.505618    4.494382
2                         46.124402   53.875598
```

Figure 21: Hierarchical Clustering Dendrogram (SMOTE data)

K-Means Clustering

```
Stroke counts per cluster:
kmeans_cluster  stroke
0               0         1292
                1           28
1               0         2110
                1         3374
dtype: int64

Stroke percentages per cluster:
                No Stroke (%)  Stroke (%)
kmeans_cluster
0                   97.878788    2.121212
1                   38.475565   61.524435
```

Figure 22: K-Means clustering results (SMOTE data)

Clustering Insights

Both hierarchical and K-Means clustering approaches revealed distinct groups with significantly different stroke risk profiles. In both methods, one cluster contained a much higher proportion of stroke cases, which can help in identifying and targeting high-risk patient groups for preventive care.

Project Code Notebook

The complete code for this project is available on Google Colab at the following link:

Google Colab Notebook