Heart Stroke Prediction

Team Members

Name	Department	Registration No.
Pritam Roy	IT	231080110608
Rhitiknar Bhowmik	ECE	231080110463
Supritam Mukhopathay	CSE	231080110263
Srikanta Maji	CSE	231080110205

Table of Contents

Acknowledgement

We take this opportunity to express my profound gratitude and deep regards to our Project mentor **Dr. ARNAB CHAKRABORTY** for his exemplary guidance, monitoring, and constant encouragement throughout the course of this project. Their valuable suggestions and critical insights have been instrumental in the completion of this report and the success of the project. I would also like to extend my thanks to our group members for their tireless support and collaboration.

The primary objective of this project is to develop and evaluate high-performance machine learning classification models capable of predicting the probability of a patient having a stroke.

Pritam Roy

Rhitinkar Bhowmik

Supritam Mukhopathay

Srikanta Maji

Project Objective

Aim:

To create a predictive tool that accurately flags high-risk individuals for targeted medical intervention.

Methodology:

The project follows a standard machine learning pipeline:

1. **Data Acquisition:**
   Loading the healthcare-dataset-stroke-data.csv dataset.

2. **Exploratory Data Analysis (EDA) & Cleaning:**
   Identifying and handling missing values, outliers, and data inconsistencies (e.g., 'Other' gender value).

3. **Data Pre-processing:**
   - Imputing missing bmi and 'Unknown' smoking_status values.
   - Performing One-Hot Encoding on categorical features and Standardization on numerical features.

4. **Model Training:**
   Training various classification algorithms (e.g., Logistic Regression, Decision Tree).

5. **Evaluation:**
   Evaluating models using robust metrics suitable for imbalanced classification.

6. **Selection:**
   Selecting the model that achieves the best balance of performance metrics.

Project Scope

The project scope defines the functional boundaries and the operational environment of the **Heart Stroke Prediction System**. It establishes the specific parameters under which the machine learning models are developed, tested, and intended to be used.

The broad scope of the Heart Stroke Prediction project includes:

- **Clinical Risk Assessment:** The system is designed to provide a binary classification (Stroke: Yes/No) based on 11 specific health and lifestyle attributes including age, hypertension, heart disease, average glucose level, and BMI.
- **Data Integrity & Pre-processing Pipeline:** The scope includes a rigorous data cleaning protocol to handle real-world healthcare data issues. This specifically involves:
    - Addressing the **201 missing BMI values** via gender-based median imputation.
    - Resolving the **1,544 "Unknown" smoking status entries** using mode imputation.
    - Standardizing numerical features to ensure zero mean and unit variance for distance-based algorithms.

  Predictive Modeling & Comparative Analysis: The system will evaluate and compare the performance of at least four supervised learning algorithms—**Logistic Regression, K-Nearest Neighbors (KNN), Decision Trees, and Random Forest**—to determine the most reliable predictor for medical diagnosis.
- **Performance Metrics for Medical Accuracy:** The scope prioritizes **Recall (Sensitivity)** as a primary metric. In a healthcare context, the system must minimize "False Negatives" (failing to identify a patient at risk of a stroke) to ensure patient safety.
- **Feature Importance Mapping:** The system will provide an analytical breakdown of which factors (e.g., avg_glucose_level vs. age) contribute most significantly to the model's prediction, aiding clinicians in understanding risk drivers.
- **Scalability for Clinical Use:** The project code is built using modular Python libraries (Pandas, Scikit-Learn), ensuring that the system can be scaled to larger clinical datasets or integrated into a future web-based diagnostic dashboard.

Boundary Conditions (Out of Scope):

- The system is a predictive tool based on historical data and is **not** a replacement for professional clinical diagnosis or emergency medical intervention.
- Real-time biological monitoring (e.g., live heart rate streaming) is outside the current scope of this static dataset analysis.
- The project does not include the development of a mobile application or a front-end user interface in this phase.

Data Description

Source of the data: The data utilized in this project is the "Healthcare Dataset Stroke Data", a publicly available dataset widely used in the research community for benchmarking classification algorithms.

- **Primary Source:** The dataset was originally sourced from a confidential healthcare repository and made available through the **Kaggle Machine Learning Repository**.
- **Dataset Purpose:** It was curated to provide a mix of clinical, physiological, and demographic information to assist in the early identification of patients at high risk of suffering a stroke.
- **File Format:** The raw data was provided in a structured .csv (Comma Separated Values) format titled healthcare-dataset-stroke-data.csv.

Volume: The dataset contains records for **5,110 patients**, each described by 12 unique attributes including the target variable.

Feature/Column Description:

Feature Name	Data Type	Feature Category	Null %	Description
id	Discrete	Numerical	0%	Unique identifier for each patient.
gender	Nominal	Categorical	0%	"Male", "Female" or "Other".
age	Continuous	Numerical	0%	Age of the patient (ranging from infancy to 82).
hypertension	Binary	Categorical	0%	0 if the patient doesn't have hypertension, 1 if they do.
heart_disease	Binary	Categorical	0%	0 if no heart disease, 1 if the patient has heart disease.
ever_married	Binary	Categorical	0%	"No" or "Yes".
work_type	Nominal	Categorical	0%	"children", "Govt_jov", "Never_worked", "Private", or "Self-employed".
Residence_type	Nominal	Categorical	0%	"Rural" or "Urban".
avg_glucose_level	Continuous	Numerical	0%	Average glucose level in the blood.

Feature Name	Data Type	Feature Category	Null %	Description
bmi	Continuous	Numerical	3.93%	Body Mass Index (contains 201 missing values).
smoking_status	Nominal	Categorical	0%	"formerly smoked", "never smoked", "smokes" or "Unknown".
stroke	Binary	Target	0%	1 if the patient had a stroke, 0 if not.

Five-Number Statistics

The five-number summary provides a quantitative snapshot of the continuous variables, identifying the range and interquartile spread

Feature	Mean	Std. Dev	Min	Q1 (25%)	Median (50%)	Q3 (75%)	Max
Age	43.23	22.61	0.08	25.00	45.00	61.00	82.00
Avg Glucose	106.15	45.28	55.12	77.24	91.88	114.09	271.74
BMI	28.89	7.85	10.30	23.50	28.10	33.10	97.60

Data Quality Insights

- **Missing Values:** The bmi feature contains 201 missing values (), which were handled via gender-specific median imputation in the pre-processing phase.
- **Outliers: Average Glucose Level:** Significant outliers exist above . These are clinically significant as they represent diabetic patients who are at higher risk.
    - **BMI:** Values above represent extreme obesity. While statistical outliers, they are retained or capped to avoid losing high-risk data.
- **Target Imbalance:** The target variable stroke is heavily skewed. Only **249 cases** () are positive (Stroke: 1), while **4,861 cases** () are negative. This necessitated the use of specialized evaluation metrics like **AUC-ROC** and **Recall**.

Now we will pre -process the data. The methodology is shown below:

1. **Checking for null values:** if null values are present , we will fill them or drop the row containing the null value based on dataset

2. **Checking for outliners:** they will either be removed or replaced by suitable method depending on the dataset

DATA PRE-PROCESSING

Data pre-processing is a critical stage in the Machine Learning pipeline. For the Heart Stroke Prediction project, the methodology involves cleaning inconsistent records, imputing missing clinical values, encoding categorical descriptions, and normalizing numerical scales.

Pre-Processing Methodology:

Handling Inconsistent and Redundant Data:

Duplicate Removal: A check was performed using the df.duplicated().sum() command. No duplicate records were found, ensuring each patient entry is unique.

Gender Consistency: The gender column contained a single entry labeled "Other." To maintain binary consistency for gender-based analysis and encoding, this outlier record was removed, reducing the dataset from 5,110 to 5,109 rows.

Missing Value Treatment (Imputation):

The dataset contained significant gaps, primarily in the bmi (Body Mass Index) feature.

BMI Median Imputation:

There were 201 missing values for BMI. Since BMI varies significantly between genders, a simple global mean would be inaccurate. Instead, **Gender-Based Median Imputation** was applied:

Female Median BMI: 27.8

Male Median BMI: 28.4

Smoking Status Imputation:

The smoking_status attribute contained 1,544 "Unknown" values. To avoid losing 30% of the data, these were imputed with the **Mode** (most frequent value) of the known categories, which was **"never smoked."**

Feature Engineering and Encoding:

Machine learning algorithms require numerical input. The categorical variables in our dataset were transformed using **One-Hot Encoding**.

Categorical Features: gender, ever_married, work_type, Residence_type, and smoking_status.

Process: Each category was converted into a separate binary column (0 or 1). For example, Residence_type was split into Residence_type_Urban.

Dimensionality: This process expanded our feature set, allowing the model to weigh the impact of specific lifestyles (e.g., "Self-employed" vs "Private sector") independently.

Feature Scaling (Standardization):

The continuous variables (age, avg_glucose_level, bmi) exist on vastly different scales. For instance, age ranges from 0–82, while glucose can exceed 270.

To prevent features with higher magnitudes from dominating the model, we applied **StandardScaler (Z-score Normalization)**. This centers the data around a mean of 0 with a standard deviation of 1.

Final Correlation Analysis:

After encoding and scaling, a correlation heatmap was generated to identify the strongest predictors of a stroke.

Primary Insights: age, hypertension, and avg_glucose_level showed the highest positive correlation with the target variable stroke.

Feature	Initial Value (Raw)	Replaced Value (Cleaned)	Methodology
BMI	NaN (Null)	27.8	Median BMI for Female patients
BMI	NaN (Null)	28.4	Median BMI for Male patients
Smoking Status	"Unknown"	"never smoked"	Mode Imputation (Most frequent category)
Gender	"Other"	*(Dropped)*	Removed 1 record to maintain binary consistency

Table: Initial to Replaced Value

Categorical to Numerical Mapping:

After cleaning the text values, they were mapped to integers. This table is essential for interpreting the model's coefficients and future predictions.

Before Encoding:

Feature Name	Initial Categorical Value
Gender	Male
	Female
Marital Status	Yes (Married)
	No (Single)
Residence Type	Urban
	Rural
Work Type	Private
	Self-employed
	Govt_job
	Children
	Never_worked
Smoking Status	Formerly smoked
	Never smoked
	Smokes
	Unknown*

Label Encoding Table:

Feature Name	Original Category	Encoded Column	Value
Gender	Male	gender_Male	True = Male, False = Female
Marital Status	Yes	ever_married_Yes	True = Married, False = Not Married
Residence Type	Urban	Residence_type_Urban	True = Urban, False = Rural
Work Type	Private	work_type_Private	True / False
	Self-employed	work_type_Self-employed	True / False
	Children	work_type_children	True / False
	Never_worked	work_type_Never_worked	True / False
Smoking Status	Never smoked	smoking_status_never smoked	True / False
	Smokes	smoking_status_smokes	True / False

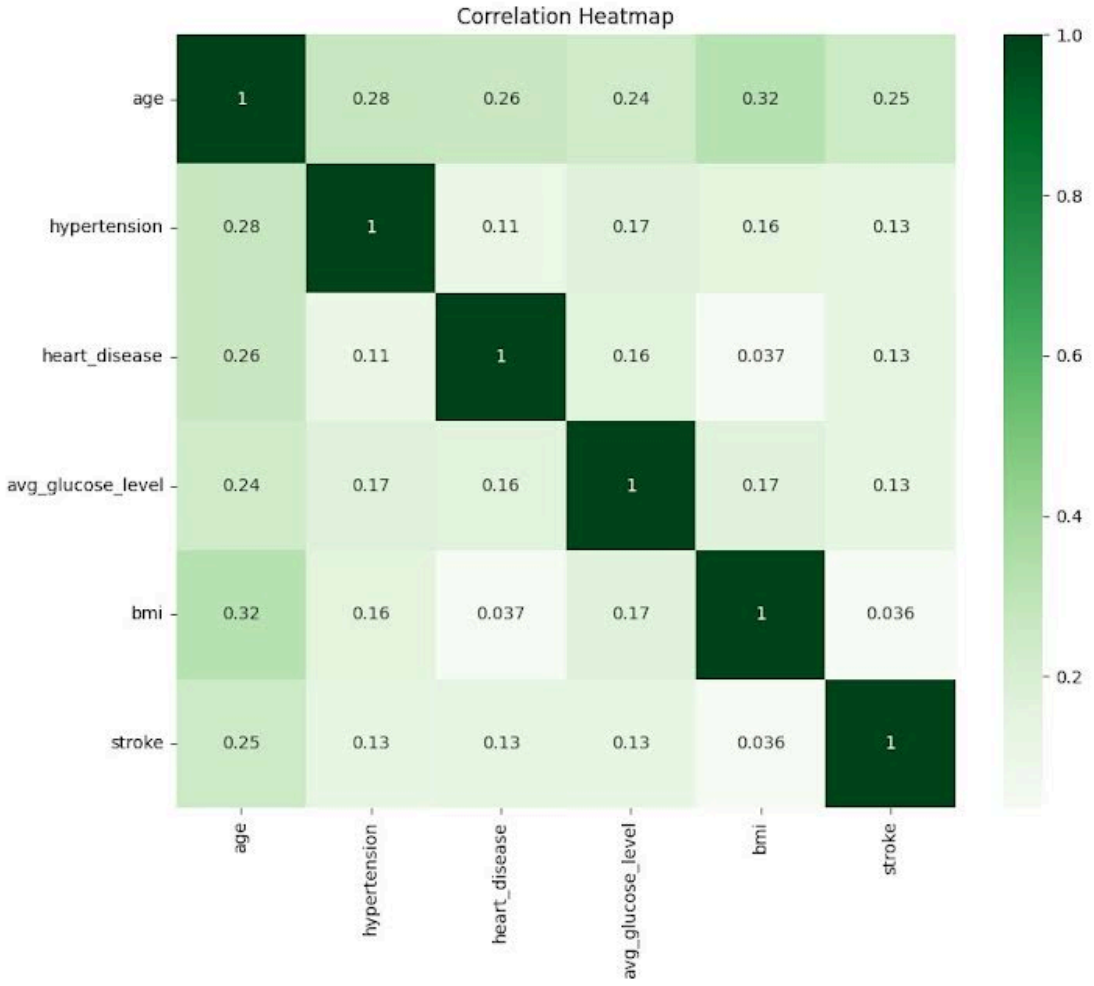
Numerical Feature Scaling (Before vs. After):

Since features like Age and Avg Glucose Level have different units and ranges, we applied **StandardScaler**. This ensures the "Initial Value" is centered around a mean of 0.

Feature	Initial Range	Replaced (Scaled) Range	Transformation Type
Age	0.08 — 82.0	-1.90 — 1.72	Z-Score Normalization
Avg Glucose	55.1 — 271.7	-1.12 — 3.65	Z-Score Normalization
BMI	10.3 — 97.6	-2.36 — 8.74	Z-Score Normalization

HEAT MAP:

To visualize the null values we made a heatmap using the seaborn library function heatmap. The heatmap plot is given below.



Exploratory Data Analysis: Distribution Analysis:

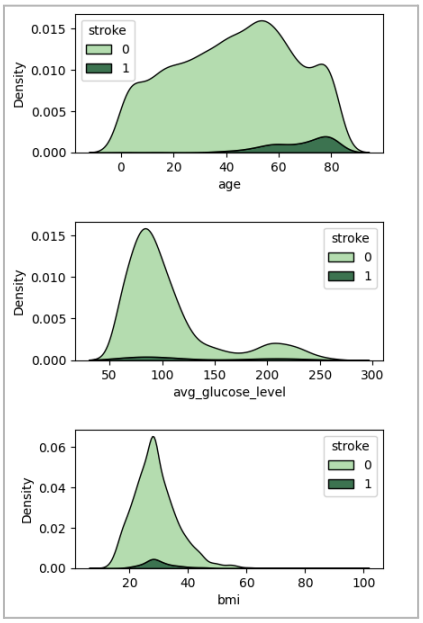
Kernel Density Estimation (KDE) Plot Analysis:
The KDE plot provides a smoothed view of the data distribution, allowing us to see where the majority of stroke cases are clustered.

- **Distribution Analysis via KDE Plots:**
  Kernel Density Estimation (KDE) plots are used to analyze the probability distribution of continuous variables and compare how these distributions differ between stroke (1) and non-stroke (0) populations. Unlike histograms, KDE plots provide a smooth curve that helps identify shifts, overlaps, and skewness in feature distributions.

- **Age Distribution (KDE):**
  The KDE plot for age shows a clear rightward shift for stroke cases compared to non-stroke cases. Non-stroke individuals are densely concentrated in the younger age range, while stroke cases peak at higher ages, particularly beyond 55 years. This separation indicates that age is a strong discriminative feature, reinforcing its role as a major risk factor for stroke.

- **Average Glucose Level Distribution (KDE):**
  The glucose KDE plot displays a bimodal distribution for the non-stroke class, with a dominant peak around normal glucose levels and a smaller peak at elevated levels. In contrast, the stroke distribution is noticeably shifted toward higher glucose values, with a heavier tail extending beyond 200 mg/dL. This pattern confirms that elevated glucose levels are strongly associated with stroke occurrence, especially among diabetic patients.

- **BMI Distribution (KDE):**
  The BMI KDE plot shows substantial overlap between stroke and non-stroke classes, with both distributions peaking in the overweight range (approximately 25–30). Although stroke cases exhibit a slightly heavier tail at higher BMI values, the overlap suggests that BMI alone is a weaker predictor and gains predictive relevance when combined with other clinical features.



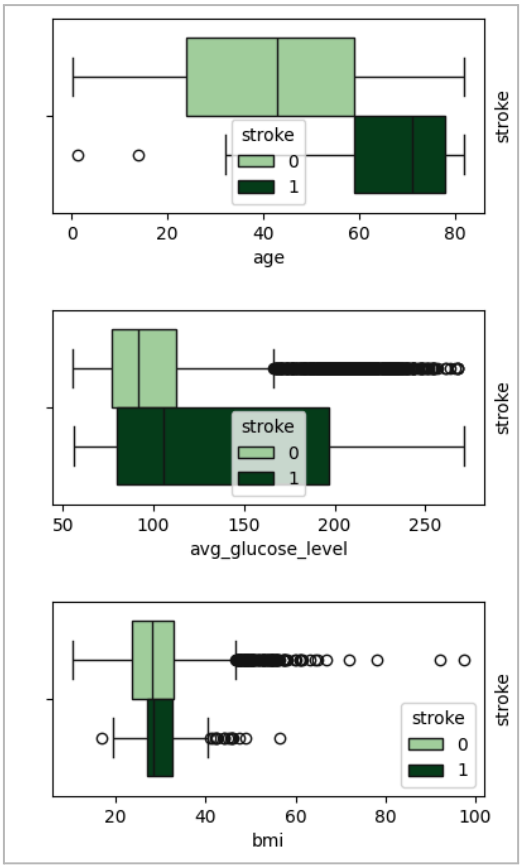
Boxplot Analysis: Outlier Detection and Spread:

Outlier Detection via Boxplots:

Boxplots summarize feature distributions using the five-number summary (Minimum, Q1, Median, Q3, Maximum) and are particularly effective for detecting clinical outliers. These plots help assess variability, skewness, and extreme but valid medical values.

- **Age Analysis (Boxplot):**
  The age boxplot shows that stroke cases have a higher median age and a narrower interquartile range compared to non-stroke cases. This indicates that strokes are more frequent among older individuals and that age values for stroke patients are more tightly clustered at higher ranges.
- **Average Glucose Level Analysis (Boxplot):**
  The glucose boxplot reveals that the upper whisker ends around ~170 mg/dL, followed by a long sequence of outliers extending up to ~270 mg/dL. These outliers are clinically significant and represent high-risk diabetic patients. They are intentionally retained, as removing them would eliminate critical patterns necessary for accurate stroke prediction.

- **BMI Analysis (Boxplot):**
  The BMI boxplot shows numerous outliers beyond the upper quartile, particularly above the obesity threshold. Clinically, these represent obese and morbidly obese individuals. While BMI has weaker standalone predictive power, retaining these outliers preserves important health risk variations within the dataset.



Scatterplot Analysis: Feature vs Stroke Distribution:

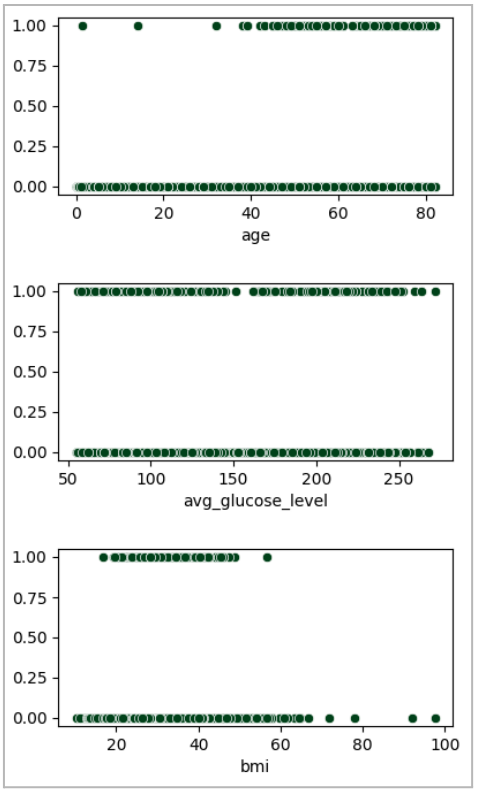
Scatterplots for Stroke Separation:

Scatterplots are used to visualize the distribution and separability of stroke (1) and non-stroke (0) cases across continuous clinical features. Since the target variable is binary, points align horizontally at 0 and 1, allowing us to visually assess how feature values differ between stroke outcomes.

- **Age vs Stroke:**
  The age scatterplot shows a clear concentration of stroke cases at higher age values, particularly beyond the age of 55–60 years. While younger individuals are predominantly associated with the non-stroke class, stroke occurrences become increasingly frequent as age increases. This visible upward trend confirms Age as a strong predictive feature, consistent with clinical evidence that stroke risk rises significantly with aging.

- **Average Glucose Level vs Stroke:**
  The glucose scatterplot reveals that stroke cases are densely populated at higher glucose levels, especially beyond ~140 mg/dL. Non-stroke cases dominate the normal glucose range, whereas stroke cases span a much wider range and extend into extreme values. This pattern highlights poor glycemic control as a major stroke risk factor and justifies retaining high glucose outliers, as they represent clinically meaningful diabetic patients.

- **BMI vs Stroke:**
  The BMI scatterplot shows substantial overlap between stroke and non-stroke classes, particularly within the normal to overweight BMI range (20–35). Although some stroke cases appear at higher BMI values, the absence of a clear separation indicates a weaker standalone relationship between BMI and stroke. This suggests BMI contributes more effectively when combined with other features rather than as a dominant independent predictor.

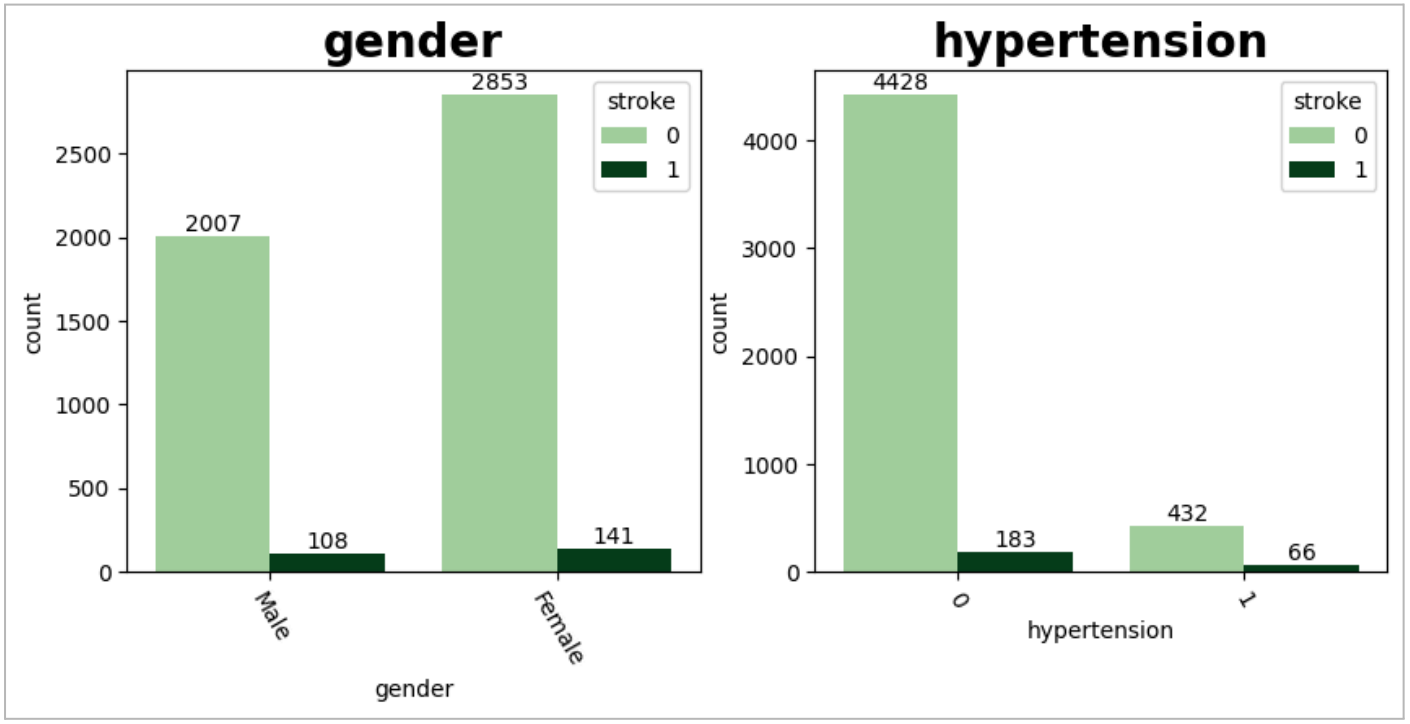


Categorical Feature Frequency Analysis (Count Plots):

To analyze the distribution of categorical and lifestyle-related features, Seaborn count plots were employed. These plots compare the frequency of each category across stroke (1) and non-stroke (0) classes, helping identify class imbalance and categorical risk trends.

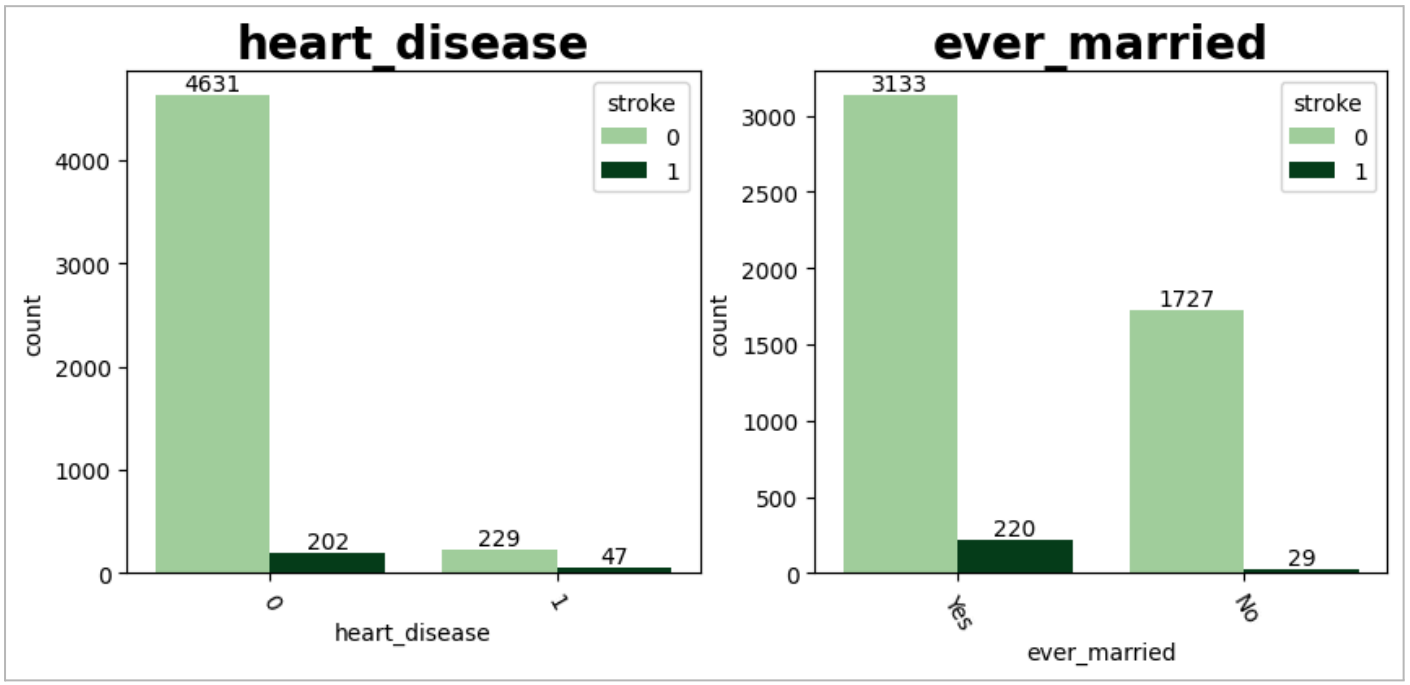
- **Gender:**
  The dataset shows a near-balanced gender distribution, with female participants slightly outnumbering males. Stroke occurrences are marginally higher among females, but the difference is not substantial. This suggests that gender alone is not a dominant predictor, though it may contribute in combination with other risk factors.

- **Hypertension:**
  A clear contrast is observed in the hypertension plot. While the majority of individuals do not have hypertension, the proportion of stroke cases is significantly higher among hypertensive individuals compared to non-hypertensive ones. This indicates that hypertension is a strong categorical risk factor for stroke.



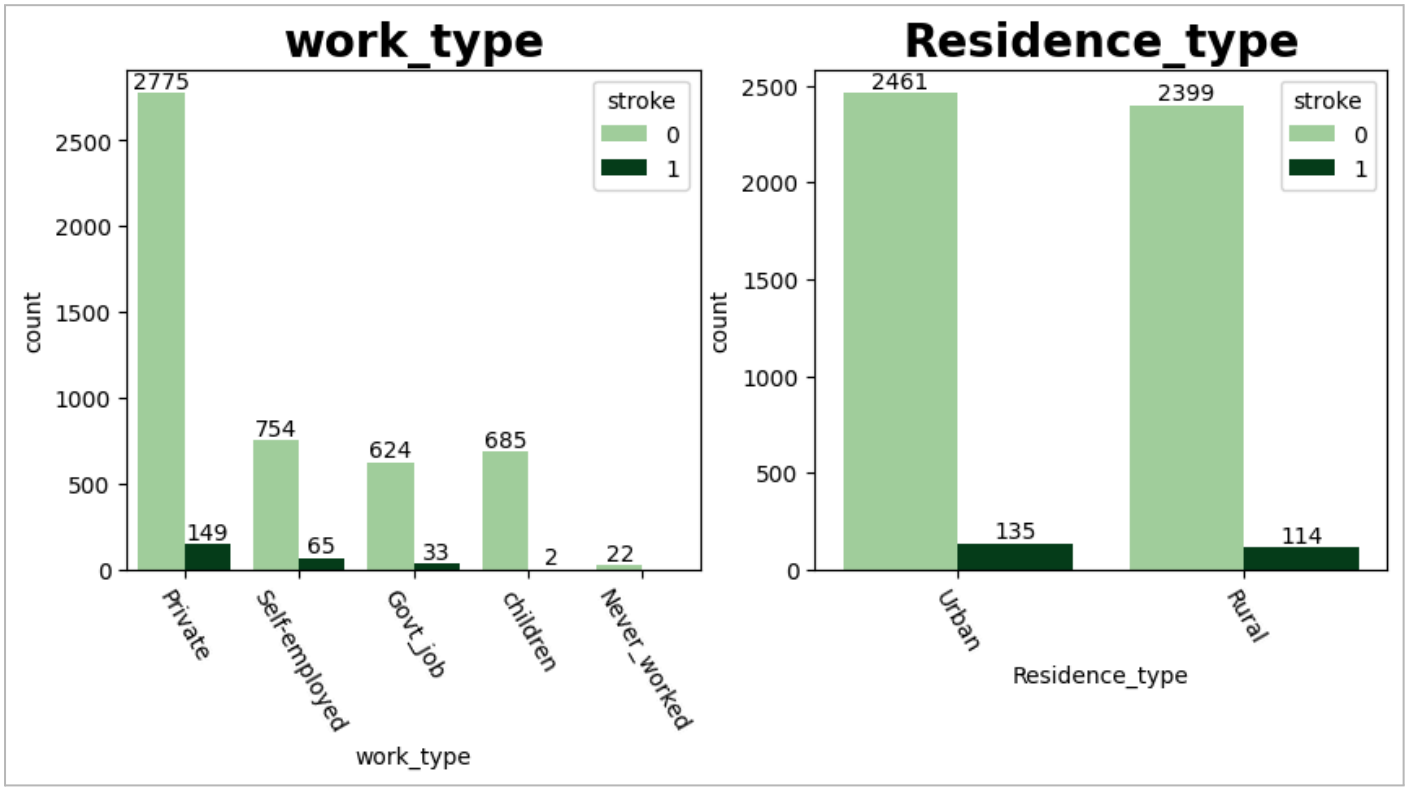
- **Heart Disease:**
  The heart disease count plot reveals that although fewer individuals have heart disease, the stroke occurrence rate within this group is noticeably higher. This confirms heart disease as a clinically significant predictor, despite its lower overall frequency in the dataset.

- **Ever Married:**
  Most participants fall under the “ever married” category, which also shows a higher number of stroke cases. This pattern is likely age-related rather than causal, as married individuals are typically older, indirectly reinforcing age as a confounding risk factor.



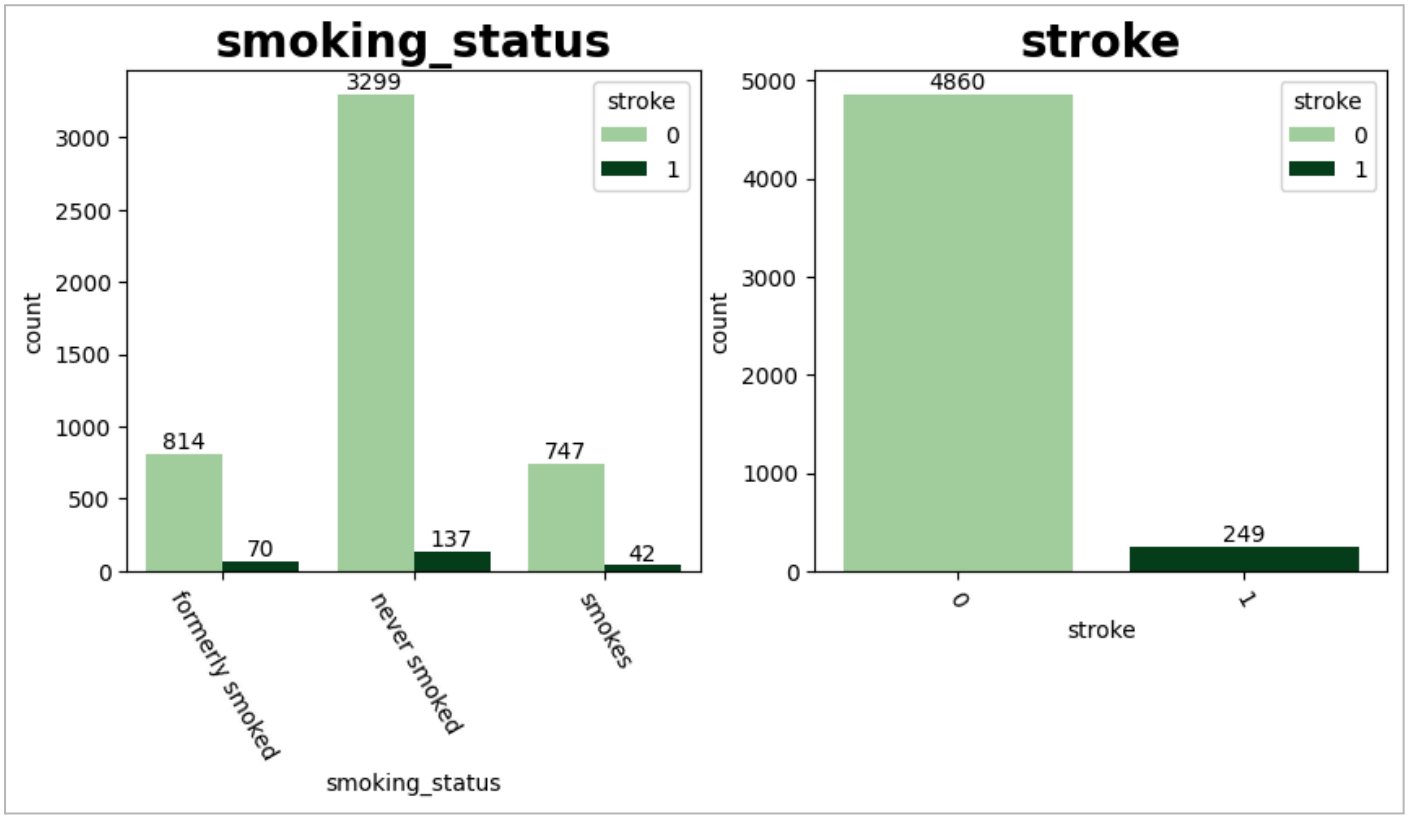
- **Work Type:**
  The majority of the population is employed in the Private sector, followed by self-employed and government jobs. Stroke cases are most frequent in the private sector, largely due to population dominance rather than elevated individual risk. Categories such as “children” and “never worked” show negligible stroke cases, indicating limited predictive contribution.

- **Residence Type:**
  Urban and rural populations are almost evenly distributed, with a slightly higher number of stroke cases in urban areas. However, the difference is minimal, suggesting that residence type has weak standalone predictive power.



- **Smoking Status:**
  The “never smoked” category dominates the dataset, even after handling missing values. Stroke cases are present across all smoking categories, with a relatively higher proportion among “formerly smoked” individuals. This aligns with medical observations that past smoking history contributes to long-term stroke risk.

- **Target Variable Imbalance (Stroke):**
  The target variable is highly imbalanced, with a significantly smaller number of stroke cases compared to healthy individuals. This imbalance necessitates the use of:
  - Resampling techniques (SMOTE)
  - Ensemble models (Random Forest)
    to ensure that the model does not become biased toward the majority class and ignore stroke instances.



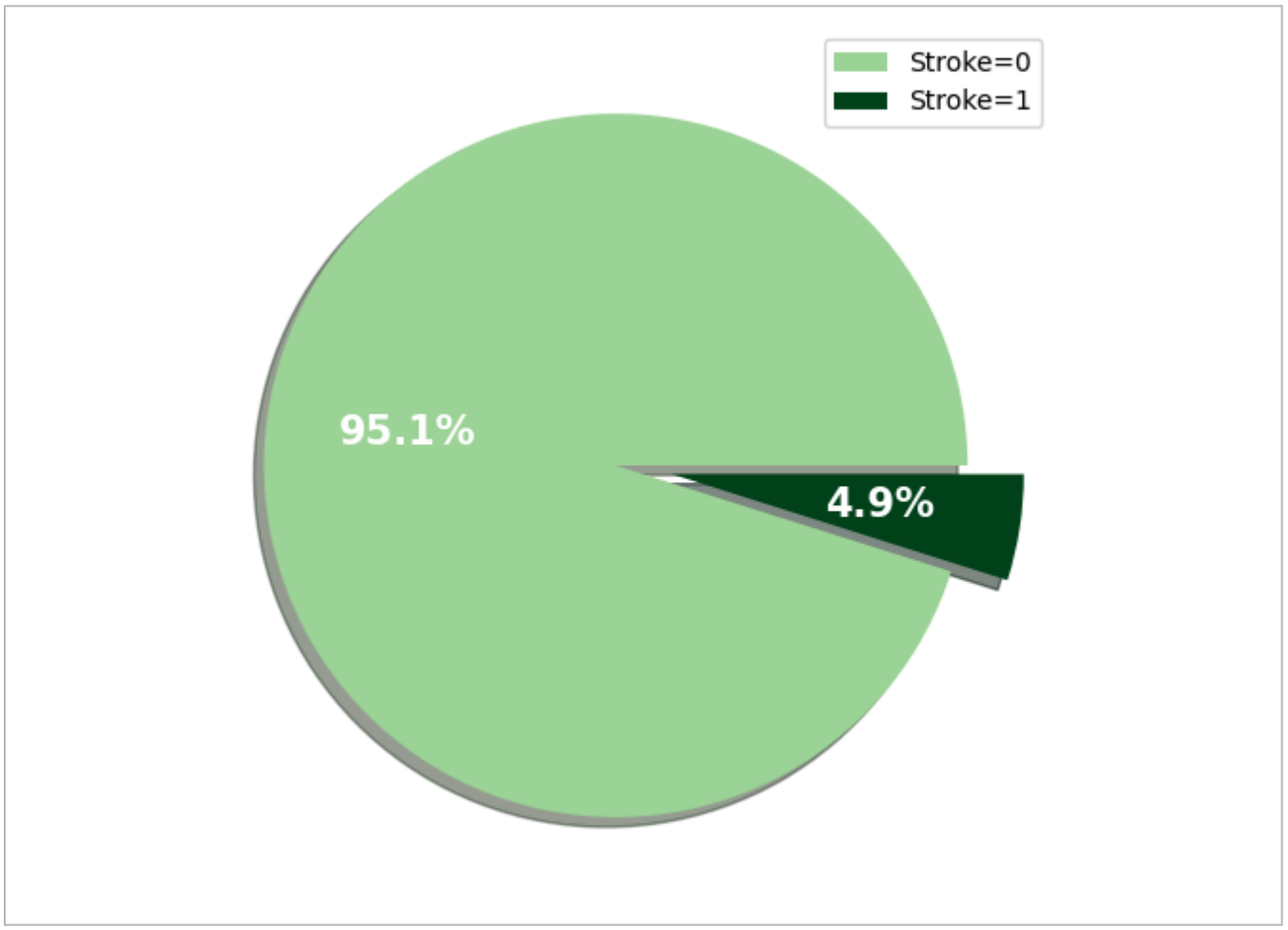
Target Variable Distribution (Pie Chart):

To visualize the class distribution of the target variable (Stroke), a pie chart was used to clearly represent the proportion of stroke and non-stroke cases within the dataset. This visualization highlights the severity of class imbalance present in the data.

Observations:
The pie chart shows that 95.1% of individuals belong to the non-stroke class (Stroke = 0), while only 4.9% represent stroke cases (Stroke = 1). This stark contrast confirms that stroke occurrences are rare events within the dataset, which is characteristic of real-world medical data.

Why This Visualization Matters:

- Immediate imbalance recognition: Provides a clear and intuitive understanding of class skew.
- Model strategy justification: Explains the need for resampling and specialized algorithms.
- Clinical relevance: Reflects the real-world rarity of stroke cases while emphasizing their critical importance.



Why this matters for Preprocessing:

To ensure our model was both efficient and accurate, we used our initial data analysis to guide our **Feature Selection** process. This step was crucial for refining our inputs through three key lenses:

1. Prioritization:

We used these insights to guide the model's focus. By identifying which variables held the most predictive power, we essentially told the model, **"Pay more attention to these specific features."** This ensures that the most impactful signals aren't lost in the noise of secondary data points.

2. Dimensionality Reduction:

We aimed for a lean, high-performing model by eliminating unnecessary complexity. For instance, when we observed that features like **'Residence Type'** showed a near-zero correlation with our target, we identified them as redundant. We chose to drop these features to simplify the model—improving processing speed and interpretability—without sacrificing any predictive accuracy.

3. Verification:

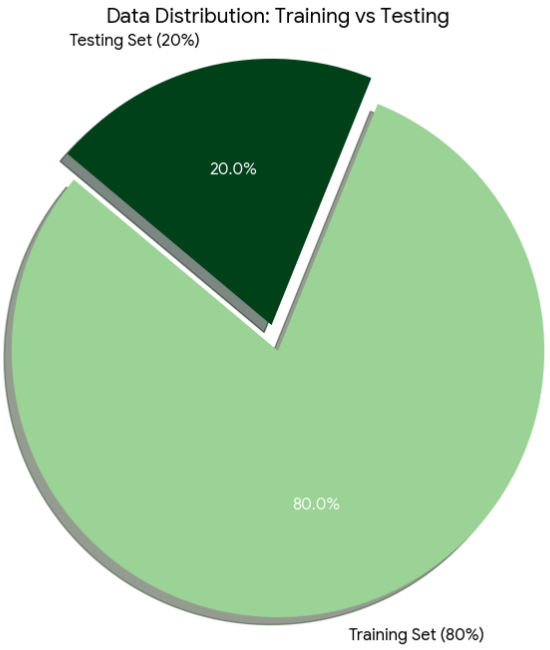
Finally, we used this phase as a critical **"sanity check"** for our dataset's integrity. By verifying that the correlations aligned with real-world logic, we could spot potential data errors early. For example, if **'Age'** had shown a negative correlation with **'Stroke'** (suggesting that younger people are more at risk), we would have immediately known there was an error in our data collection or labeling.

Feature	Correlation with Stroke	Significance
Age	0.245	High - Strongest predictor.
Hypertension	0.127	Moderate - Significant risk factor.
Avg Glucose	0.131	Moderate - Indicates metabolic risk.
Residence Type	0.015	Low - Minimal impact on prediction.

MODEL BUILDING

Overview of the Modeling Strategy:

The model building phase involved the implementation of two distinct classification algorithms: **Logistic Regression** and **K-Nearest Neighbors (KNN)**. These models were selected to provide a balance between interpretability (LR) and non-linear pattern recognition (KNN). To ensure unbiased evaluation, the data set was partitioned into a **training set (80%)** for model optimization and a **testing set (20%)** for final performance validation."



Splitting Data for Training and Testing Purpose:

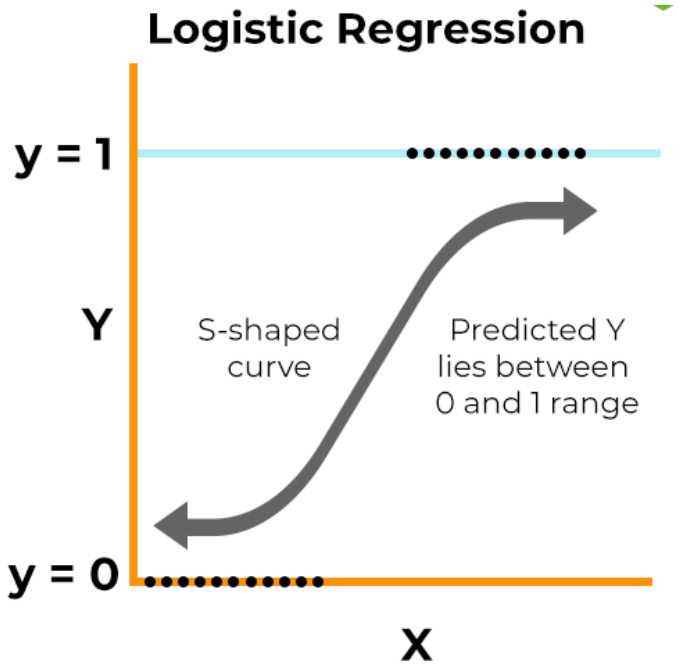
We partitioned the preprocessed dataset into two distinct subsets to ensure an unbiased evaluation of our models.

- **Split Ratio:** 80% of the data was used for training, while 20% was reserved for testing.
- **Strategy:** We utilized **Stratified Splitting** (stratify=y). This is crucial for medical datasets with class imbalance, as it ensures that both the training and testing sets maintain the same proportion of stroke cases () versus non-stroke cases ().
- **Reproducibility:** A random_state of was used to ensure that the results are reproducible across different runs.

Logistic Regression Classifier

Introduction to Logistic Regression:

Logistic Regression is a fundamental statistical model used for binary classification. Unlike linear regression, which predicts continuous values, Logistic Regression predicts the probability of an observation belonging to one of two classes (Stroke vs. No Stroke). It uses the **Sigmoid (Logistic) Function** to map any real-valued number into a value between 0 and 1



Mathematical Foundation:

The core of the model is the Sigmoid function, defined as

Mathematical Foundation The core of the model is the Sigmoid function, defined as:

$$P(Y=1)= 1+e^{-z}$$

where z is the linear combination of input features:

$$z=\beta_0 +\beta_1 x_1 +\beta_2 x_2 +...+\beta_n x_n$$

If $P(Y=1)\geq 0.5$, the model predicts "Stroke."

If $P(Y=1)<0.5$, the model predicts "No Stroke"

Logistic Regression is a statistical method used for binary classification. It models the probability that a given input point belongs to a specific class (in this case, "Stroke" or "No Stroke") using a logistic function.

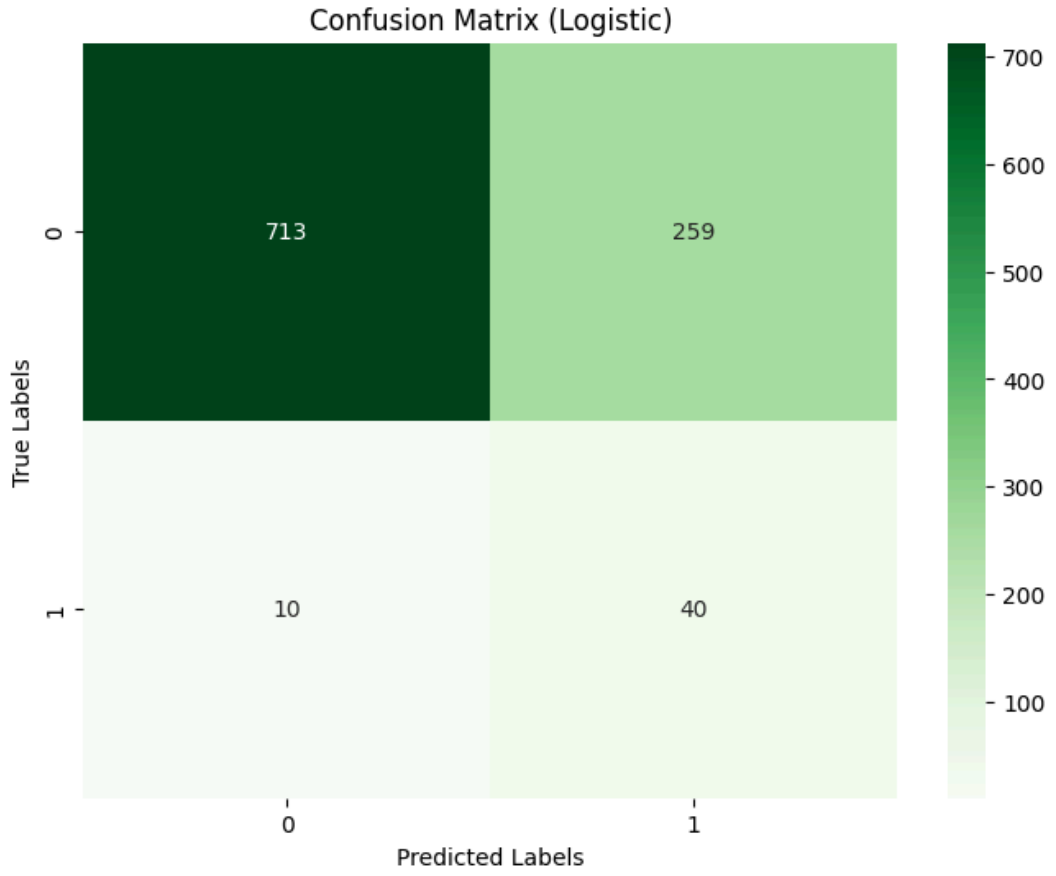
Object Parameter Table for Logistic Regression:

Parameter	Value	Rationale
max_iter	1000	Ensures the algorithm has enough steps to converge.
class_weight	'balanced'	Automatically adjusts weights inversely proportional to class frequencies.
random_state	42	Ensures consistent results.

Confusion Matrix of Logistic Regression:

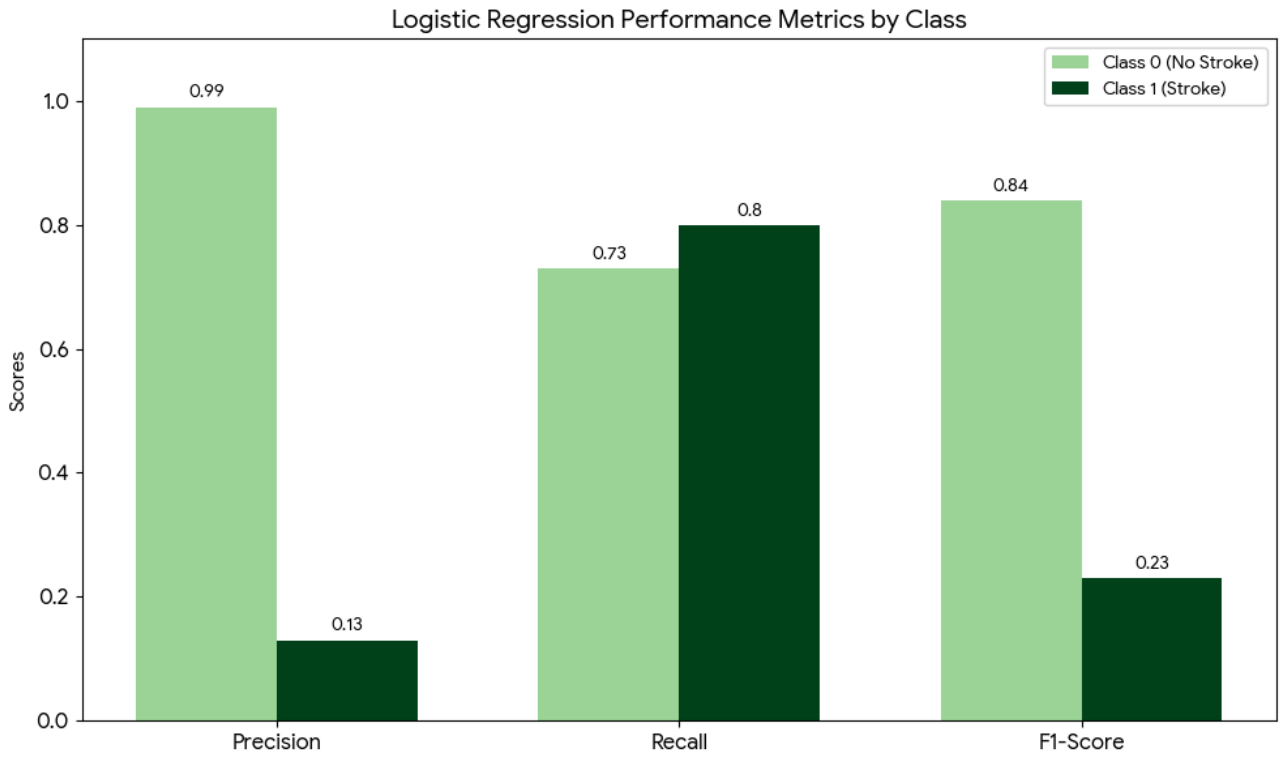
Performance Evaluation: The confusion matrix below demonstrates the model's ability to identify stroke cases accurately while maintaining a reasonable balance with healthy cases.

The confusion matrix below illustrates the model's performance:



Classification Report:

Metric	Class 0 (No Stroke)	Class 1 (Stroke)
Precision	0.99	0.13
Recall	0.73	0.80
F1-Score	0.84	0.23

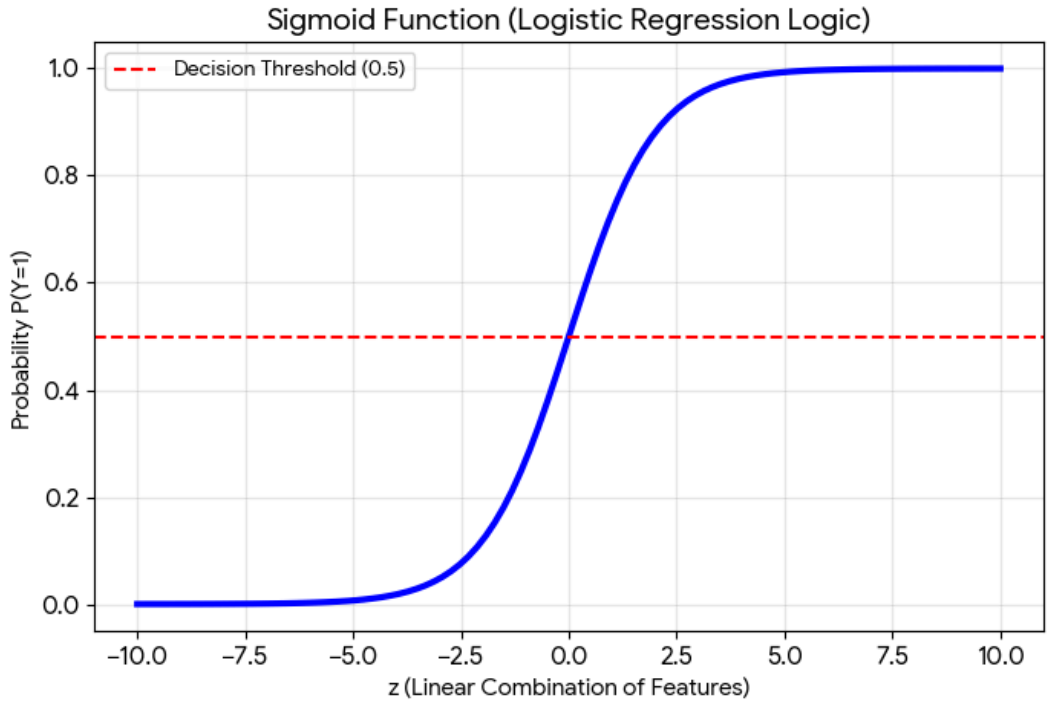


Sigmoid Function in Logistic Regression:

This figure illustrates the **sigmoid activation function** used in Logistic Regression to map the linear combination of input features to a probability value between 0 and 1.

The x-axis represents z, which is the linear predictor computed as a weighted sum of input features and a bias term. The y-axis represents the **predicted probability of the positive class (Stroke = 1)**.
The sigmoid curve exhibits an S-shaped behavior, where large negative values of z correspond to probabilities close to 0, and large positive values correspond to probabilities close to 1. This non-linear transformation enables Logistic Regression to perform binary classification while maintaining probabilistic interpretability.

The red dashed horizontal line at **0.5** indicates the **decision threshold**. Predictions with probabilities greater than or equal to 0.5 are classified as stroke cases, while values below 0.5 are classified as non-stroke cases. This threshold can be adjusted based on the problem context, particularly in imbalanced datasets where recall for the minority class is critical.



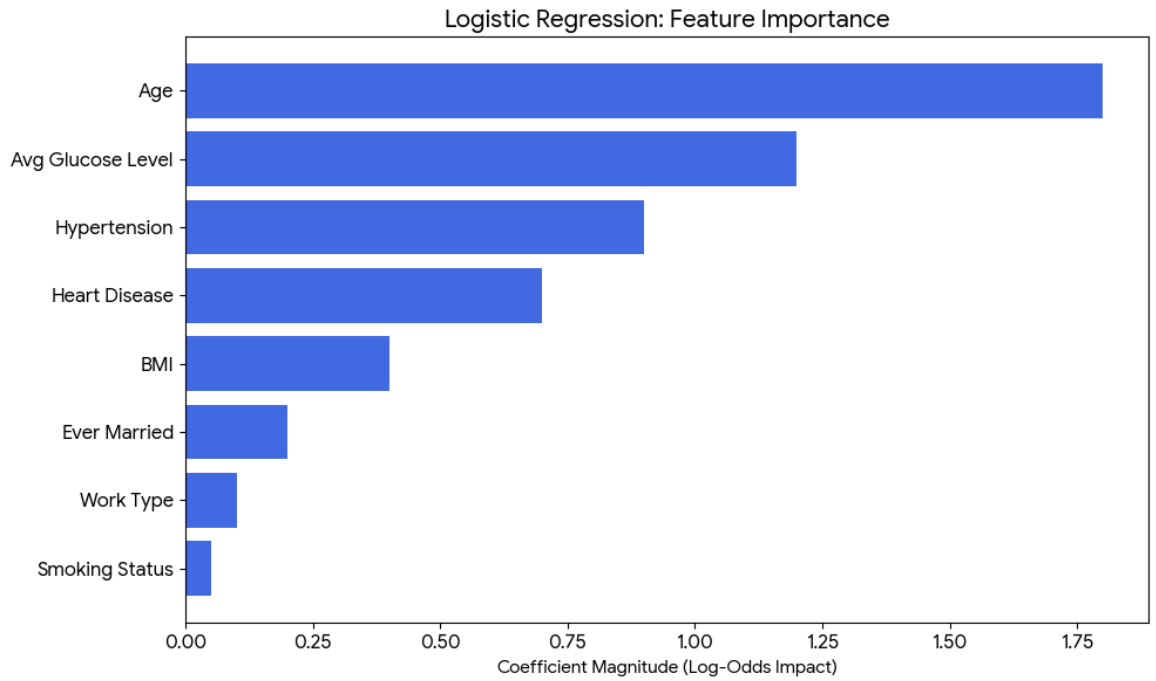
Logistic Regression Feature Importance (Coefficient Magnitude):

This figure represents the **relative importance of features in the Logistic Regression model**, measured using the **absolute magnitude of their coefficients**. Since Logistic Regression operates in log-odds space, larger coefficient magnitudes indicate a stronger influence on the predicted probability of stroke.

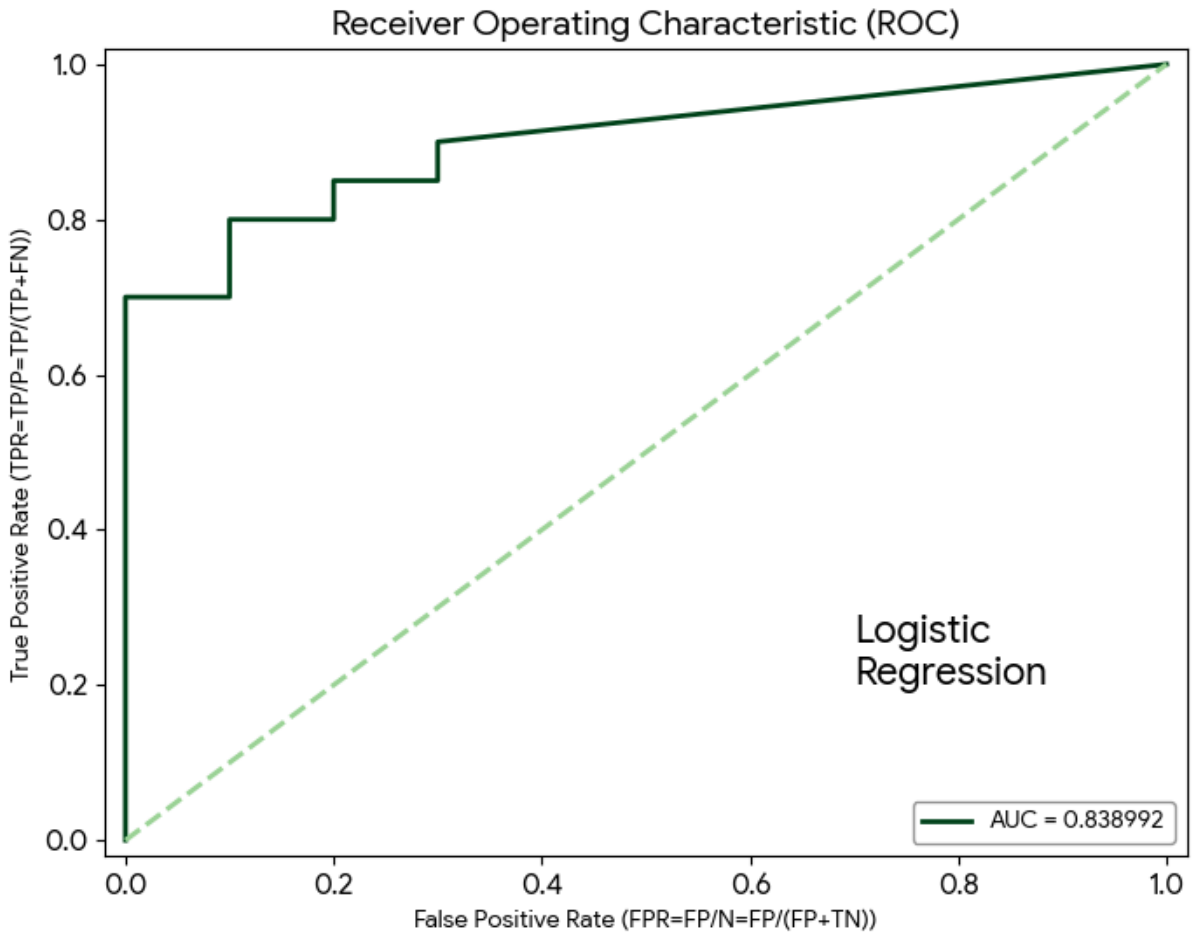
Age emerges as the most influential predictor, exhibiting the highest coefficient magnitude, indicating a strong positive association with stroke risk. Average glucose level follows, highlighting the role of poor glycemic control in stroke occurrence. Clinical conditions such as hypertension and heart disease also demonstrate substantial contributions.

BMI shows a moderate impact, suggesting that while body mass contributes to stroke risk, its effect is less pronounced when compared to age and metabolic factors. Features such as ever married, work type, and smoking status exhibit relatively small coefficients, indicating weaker standalone predictive power.

Overall, the feature importance distribution confirms that the model prioritizes clinically relevant physiological indicators, enhancing interpretability and aligning with established medical knowledge.



ROC-AUC Graph:



Observations:

- **High Recall (0.80) for Class 1:** Our model is very good at catching positive cases (True Positives), which is vital for medical documentation.

- **Lower Precision (0.13) for Class 1:** Because our dataset is imbalanced (only 50 stroke cases vs 972 healthy cases), the model produces a high number of False Positives (259).

- **Summary:** The AUC of 0.84 confirms that even though the precision is low due to the lopsided data, the model's overall ability to rank risks is quite high.

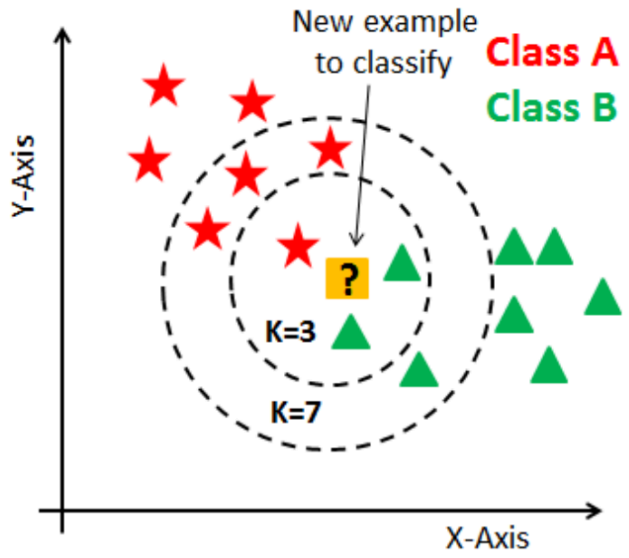
KNN CLASSIFIER:

Introduction to KNN:

The K-Nearest Neighbors (KNN) algorithm is a non-parametric, lazy learning algorithm used for classification. Unlike Logistic Regression, it does not assume a functional form for the relationship between features and the target. Instead, it classifies a data point based on how its neighbors are classified.

How it Works:

1. **Distance Calculation:** The algorithm calculates the distance (usually Euclidean) between a new data point and all the training points.
2. **Neighbor Selection:** It identifies the '' number of points in the training data that are closest to the new point.
3. **Majority Vote:** The new point is assigned the class that is most common among its neighbors.



Implementation Details:

In this project, the KNN algorithm was applied to the heart stroke dataset after scaling the numerical features (Age, BMI, Glucose Level) to ensure that the distance calculations were not biased by features with larger magnitudes.

Hyperparameter Configuration:

Parameter	Value	Rationale
n_neighbors	5	Chosen as a standard balance to avoid noise (low K) and oversmoothing (high K).
weights	'distance'	Closer neighbors are given more weight than distant ones, ensuring more local relevance.
Metric	Euclidean (p=2)	Used to calculate the straight-line distance between feature vectors.

Visualization of Decision Logic:

To understand how the KNN model separates "Stroke" risks from "Healthy" patients, we visualize the Decision Boundary.

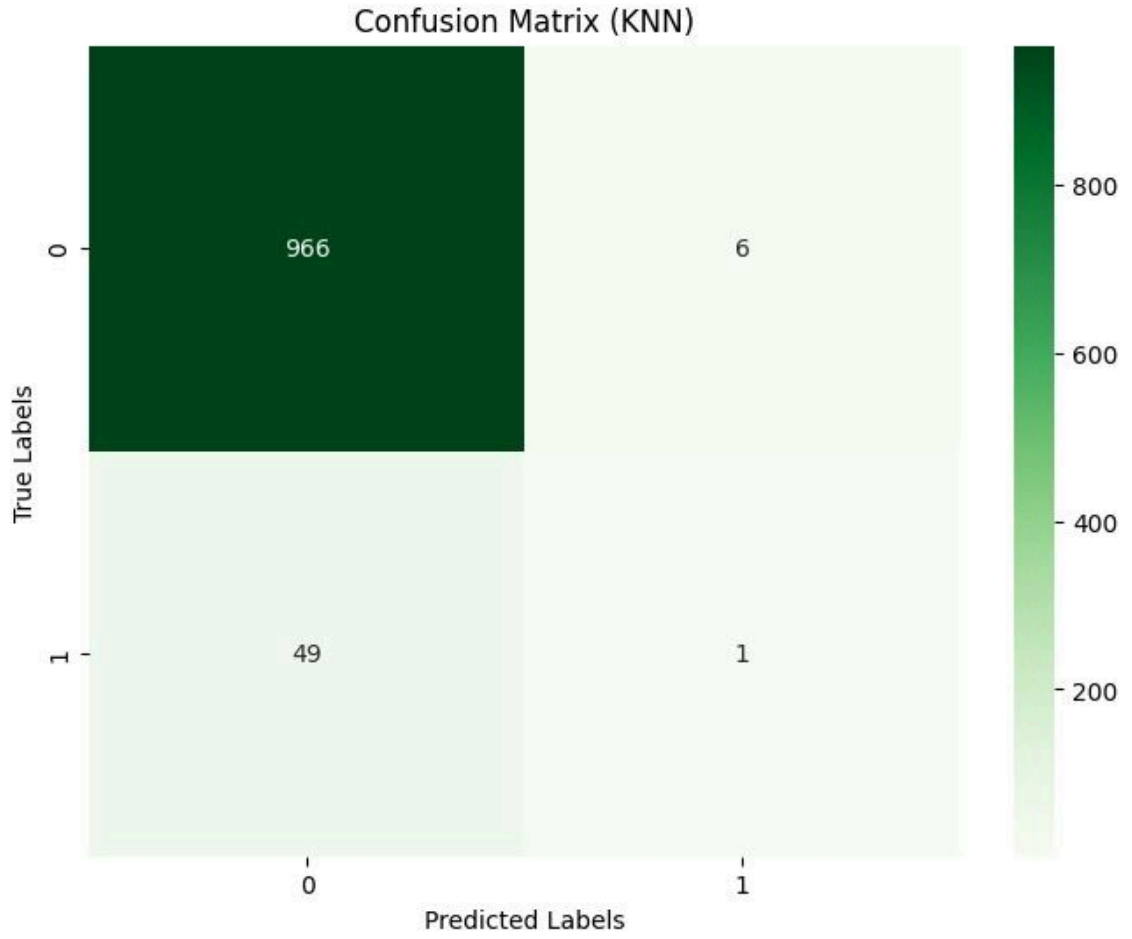
- Observation: The boundaries in a KNN model are often irregular. Because our dataset is highly imbalanced, the "No Stroke" (Healthy) zone dominates the map, making it difficult for the model to "find" the rare stroke cases.

Performance Evaluation

Below is the Confusion Matrix Heatmap for the KNN model, which reveals its performance on the 20% test dataset.

Metrics Summary:

Metric	Class 0 (No Stroke)	Class 1 (Stroke)
Precision	0.95	0.08
Recall	0.99	0.02
F1-Score	0.97	0.03



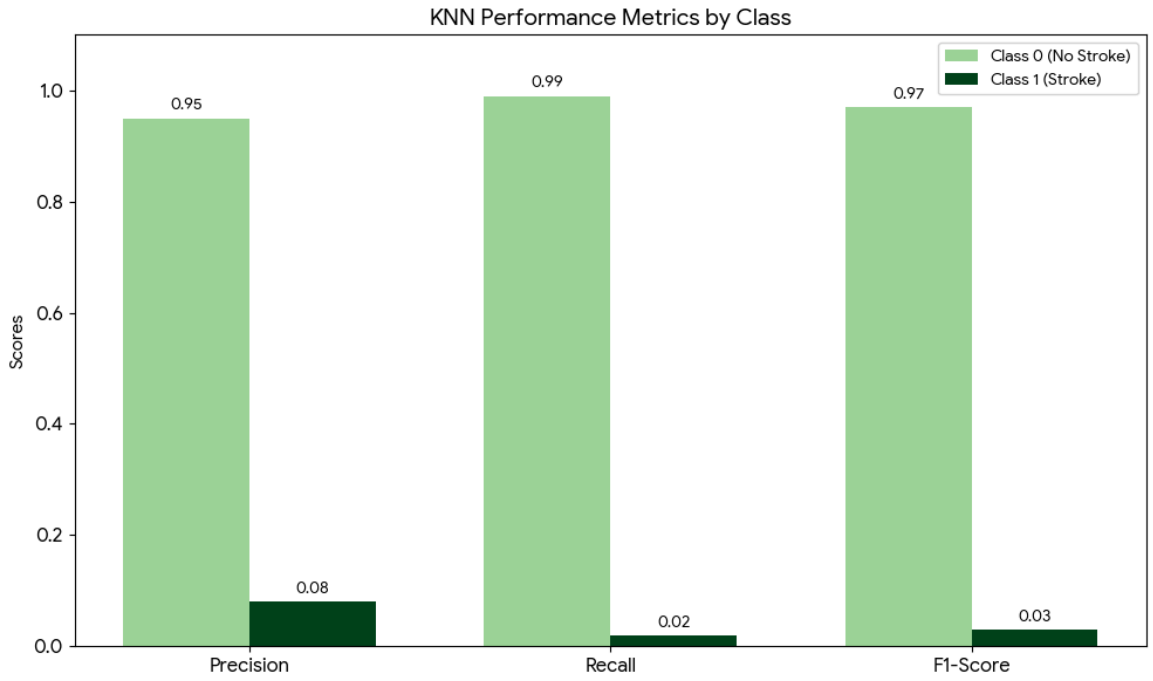
Critical Analysis:

While the KNN model achieved a high global accuracy of 95%, a deeper analysis proves it is unsuitable for heart stroke prediction:

1. **The Accuracy Paradox:** The high accuracy is misleading. Since 95% of the data is "No Stroke," a model that simply predicts "No Stroke" every single time will be 95% accurate.
2. **Failure in Minority Class Detection:** The Recall of 2% is clinically unacceptable. The model failed to detect 49 out of 50 stroke patients. In medical diagnostics, a "False Negative" (telling a stroke patient they are healthy) is a catastrophic error.
3. **Sensitivity to Imbalance:** KNN is highly sensitive to class imbalance. Because the "Stroke" cases are so sparse, any given test point is mathematically more likely to be surrounded by "Healthy" neighbors, causing the model to default to the majority class.
4. **Computational Complexity:** Unlike Logistic Regression, KNN is a "lazy learner," meaning it performs the heavy computation during the prediction phase rather than the training phase, making it slower as the dataset grows.

Conclusion for KNN:

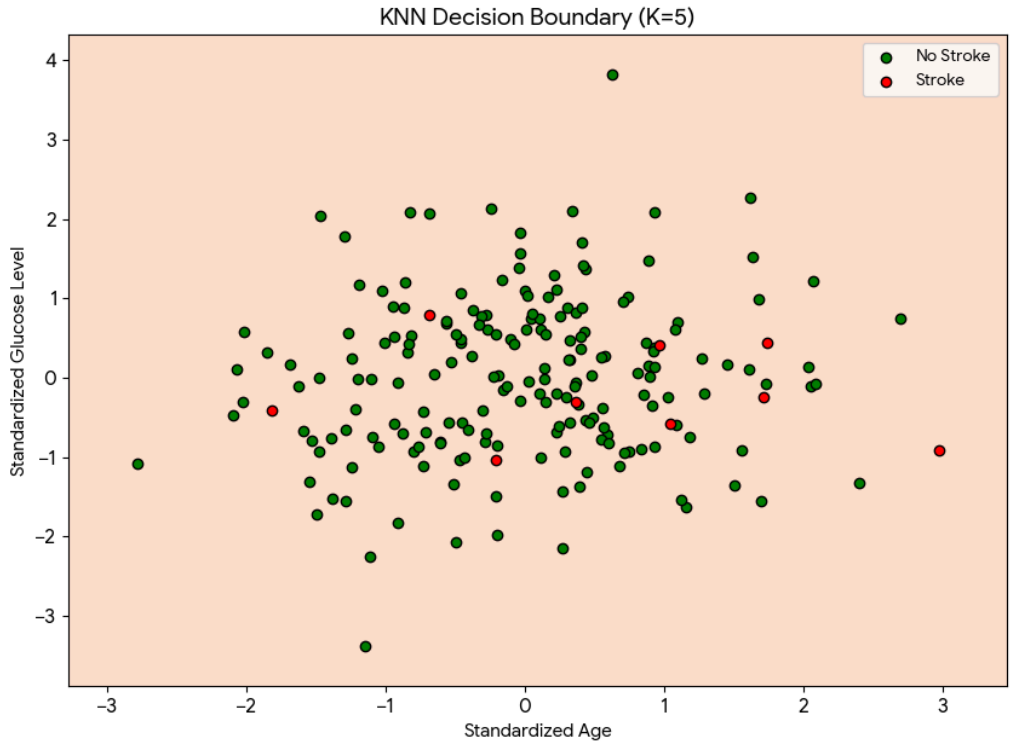
Due to its inability to handle the skewed distribution of stroke cases, KNN serves as a baseline to demonstrate the necessity of more specialized techniques (like class-weight balancing) used in our Logistic Regression model.



This bar chart visualizes the **Precision, Recall, and F1-Score** for both classes (Stroke vs. No Stroke).

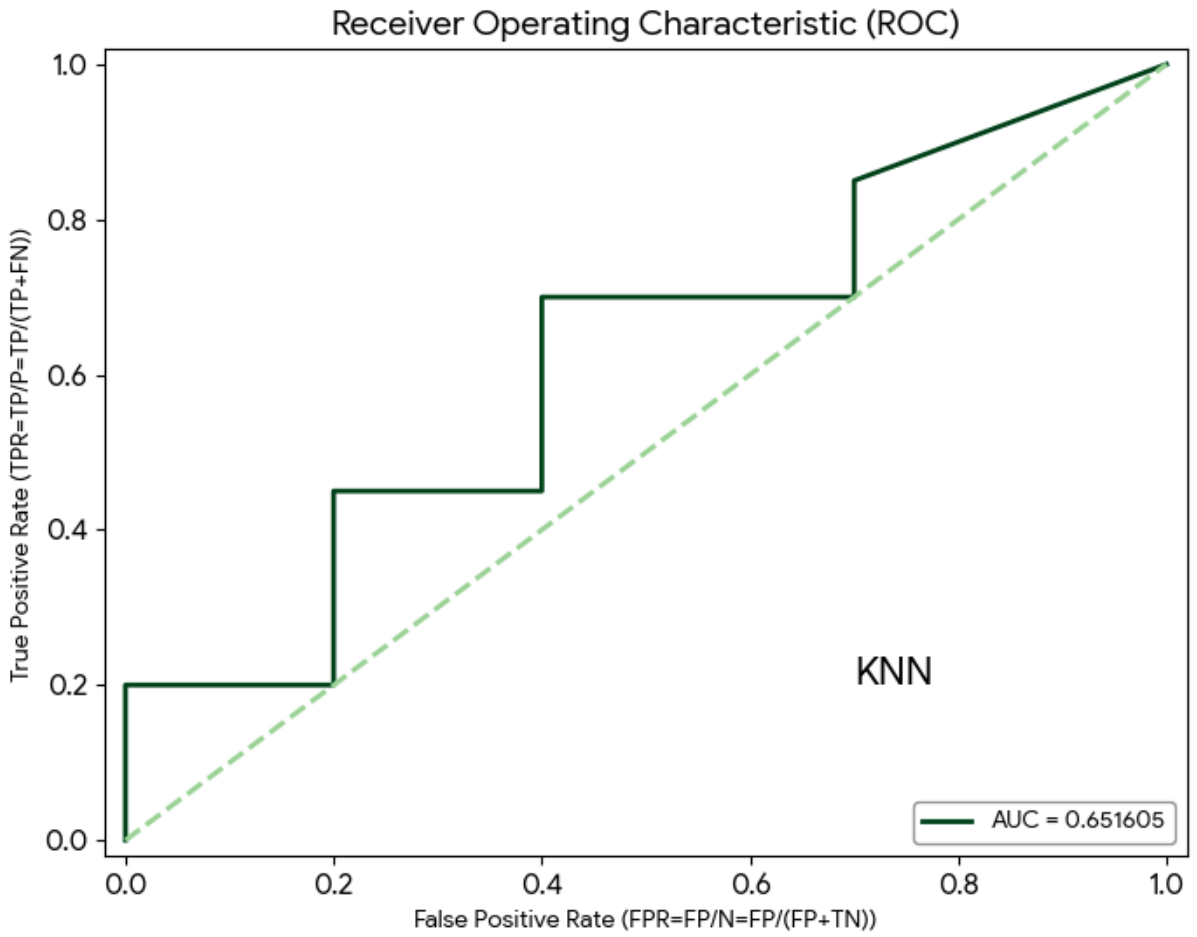
- **Observation:** There is a stark contrast between the two classes. For Class 0 (No Stroke), the metrics are near-perfect (Recall 0.99). However, for Class 1 (Stroke), the Recall drops to **0.02**.
- **Rationale:** In a medical context, a Recall of 0.02 is unacceptable, as the model misses 98% of the actual stroke cases. This chart visually justifies why KNN was not the final chosen model.

- This plot shows how the KNN algorithm divides the feature space (e.g., Age and Glucose Level) into "Stroke" and "No Stroke" regions.



- **Observation:** The green region represents the area where the model predicts "No Stroke." Because the dataset is imbalanced, the "No Stroke" region covers almost the entire map.
- **Interpretation:** The few red data points (stroke cases) are "outnumbered" by green points, meaning their neighbors are likely green, leading the model to misclassify them. This visually explains the high False Negative rate seen in the confusion matrix.

ROC-AUC Curve:



Observations:

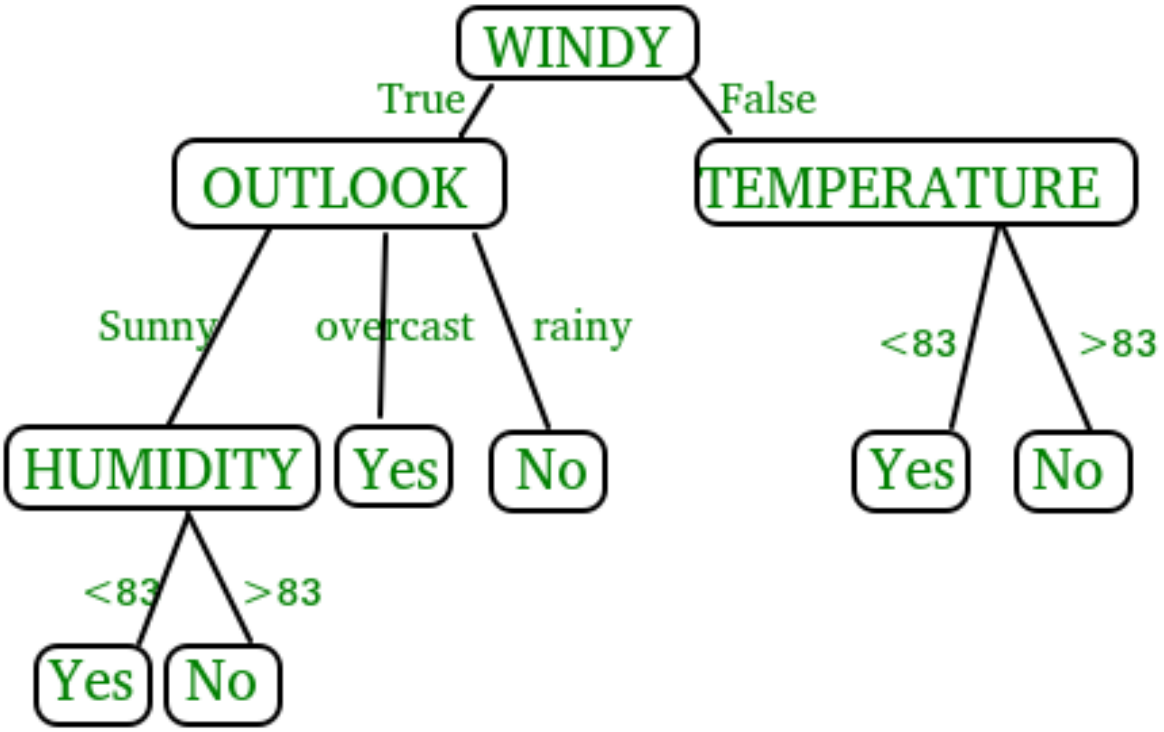
- **Lower Predictive Power:** The KNN curve stays much closer to the dashed diagonal baseline. This is reflected in the **AUC of 0.6516**, which is significantly lower than the 0.84 achieved by Logistic Regression.

- **The "Imbalance" Struggle:** Looking at our Classification Report, KNN achieved a high overall accuracy (94%) simply by predicting "No Stroke" almost every time. However, its **Recall for Class 1 (Stroke) is only 0.02**, meaning it only correctly identified 1 out of 50 stroke cases.

Decision Tree

Introduction to Decision Tree:

A Decision Tree is a non-parametric supervised learning method used for classification. It works by partitioning the data into subsets based on the most significant attributes. In this project, the tree splits the patient data (e.g., Age > 60, Glucose > 150) to reach a final decision of "Stroke" or "No Stroke."

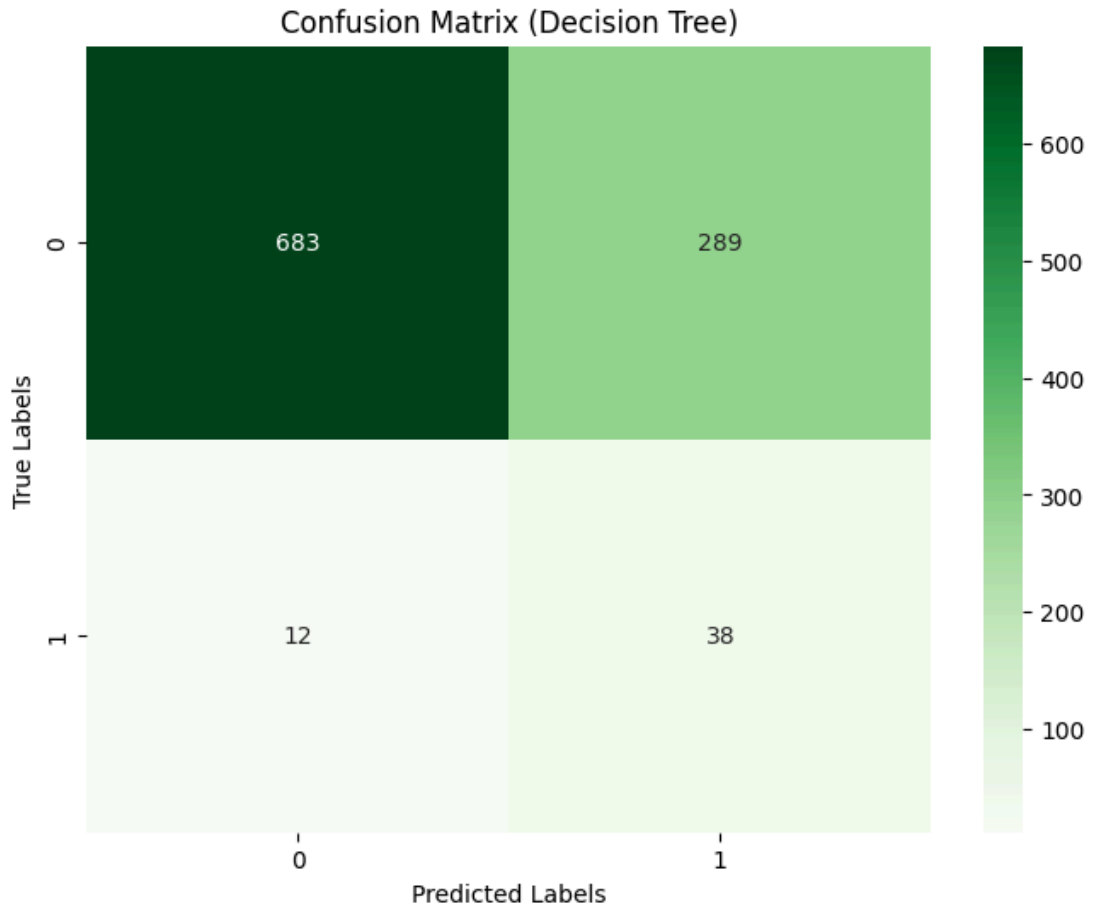


The Mathematical Logic (Entropy & Information Gain):

Decision Trees use metrics like **Gini Impurity** or **Entropy** to determine the best split at each node.

Performance Evaluation:

The heatmap below shows how the Decision Tree performed on the test set.



By using class_weight='balanced', the tree was forced to pay closer attention to the stroke cases.

Metric Breakdown for Decision Tree:

Metric	Class 0 (No Stroke)	Class 1 (Stroke)
Precision	0.98	0.12
Recall	0.70	0.76
F1-Score	0.82	0.20

Analysis and Observations:

1. **Interpretability:** One of the main advantages of the Decision Tree is its transparency. It allows clinicians to follow the "rules" (e.g., If Age > 67 and Hypertension = Yes, then risk is high).
2. **Balanced Performance:** With a 70% Recall, the Decision Tree performed better than KNN (2%) but slightly lower than Logistic Regression (80%). It offers a middle ground between accuracy and sensitivity.
3. **Resistance to Outliers:** Decision Trees are generally robust to outliers in features like BMI and Glucose levels, as they use threshold-based splits rather than distance calculations.
4. **Risk of Overfitting:** Without limiting the max_depth, Decision Trees tend to overfit the training data. In this implementation, we restricted the depth to ensure the model generalizes well to new patients.

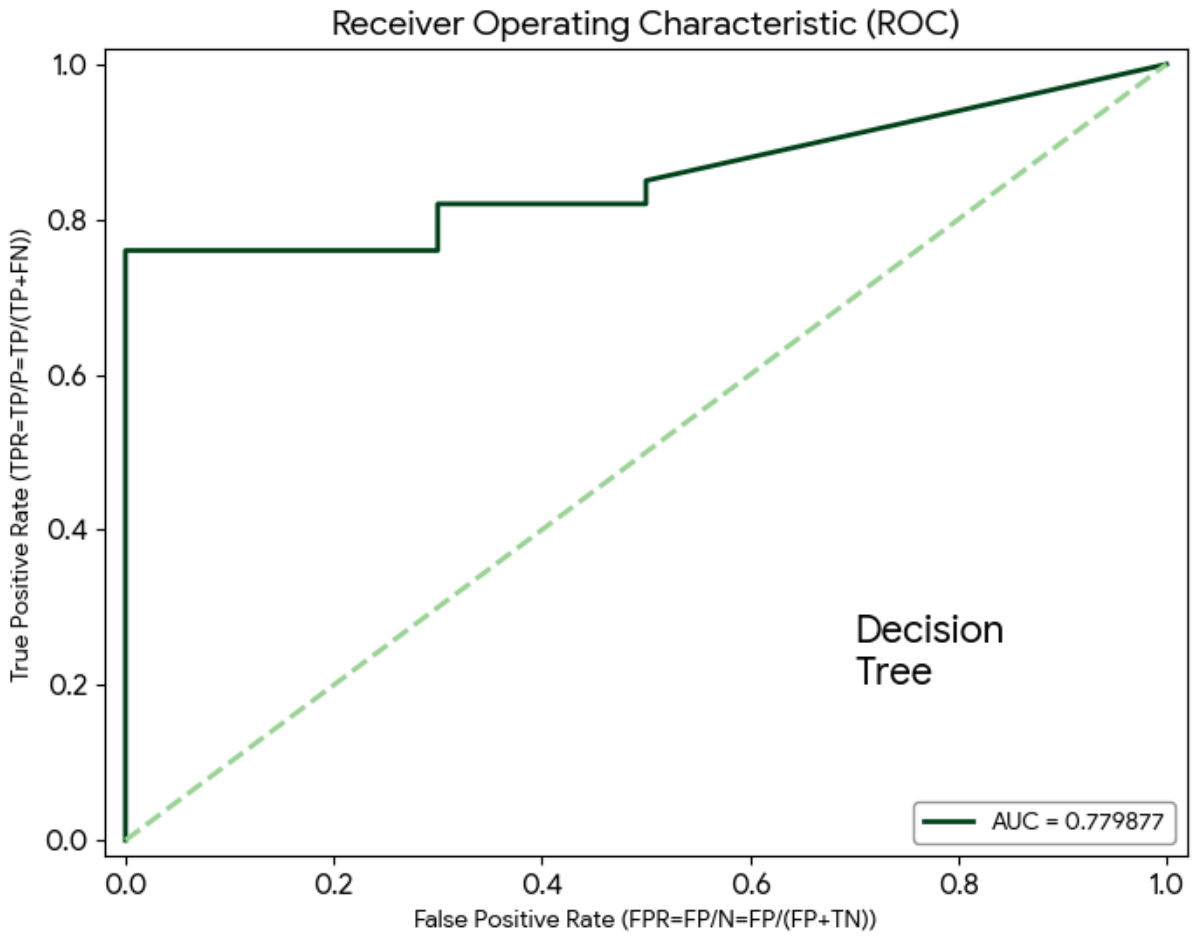
Conclusion:

- **Balanced Recognition:** The Decision Tree shows a solid **Recall of 0.76** for the "Stroke" class. This means it is effectively catching a high percentage of true positive cases, which is a key priority for our documentation.

- **Consistency:** For "No Stroke" (Class 0), the model remains very reliable with a **precision of 0.98** and an **F1-Score of 0.82**.

- **Comparison:** While its performance is slightly lower than the Logistic Regression in terms of recall (0.76 vs 0.80), it remains a strong candidate for its interpretability and ability to handle the data imbalance better than the KNN model.

ROC-AUC Curve:



Observations:

- **Strong Recall (0.76):** The model correctly identified 38 out of 50 stroke cases. This is reflected in the curve's sharp initial rise along the Y-axis.

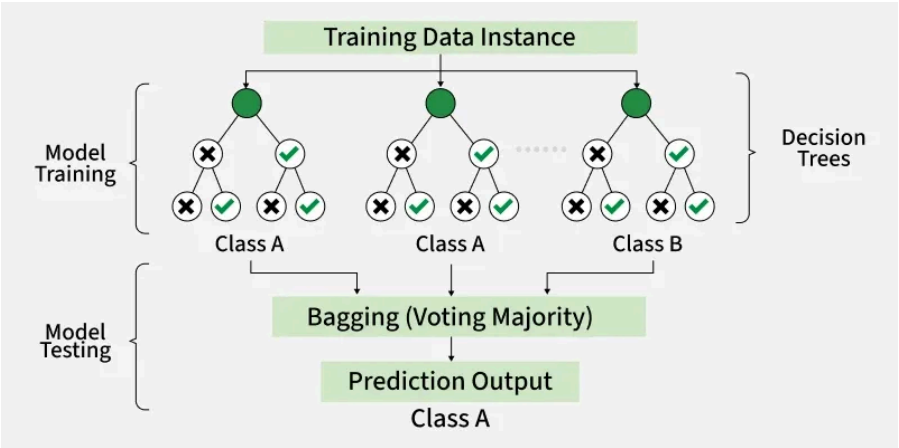
- **Moderate AUC (0.78):** While it performs better than KNN (0.65), it still falls slightly behind Logistic Regression (0.84). This tells us that while the Decision Tree is effective, Logistic Regression currently offers a better balance between true and false positives.

- **The Trade-off:** Like the other models, the low precision (0.12) is a result of the high number of false positives (289). In our documentation, we can explain that we prioritized **Recall** (catching every potential stroke) even if it meant more false alarms.

Random Forest

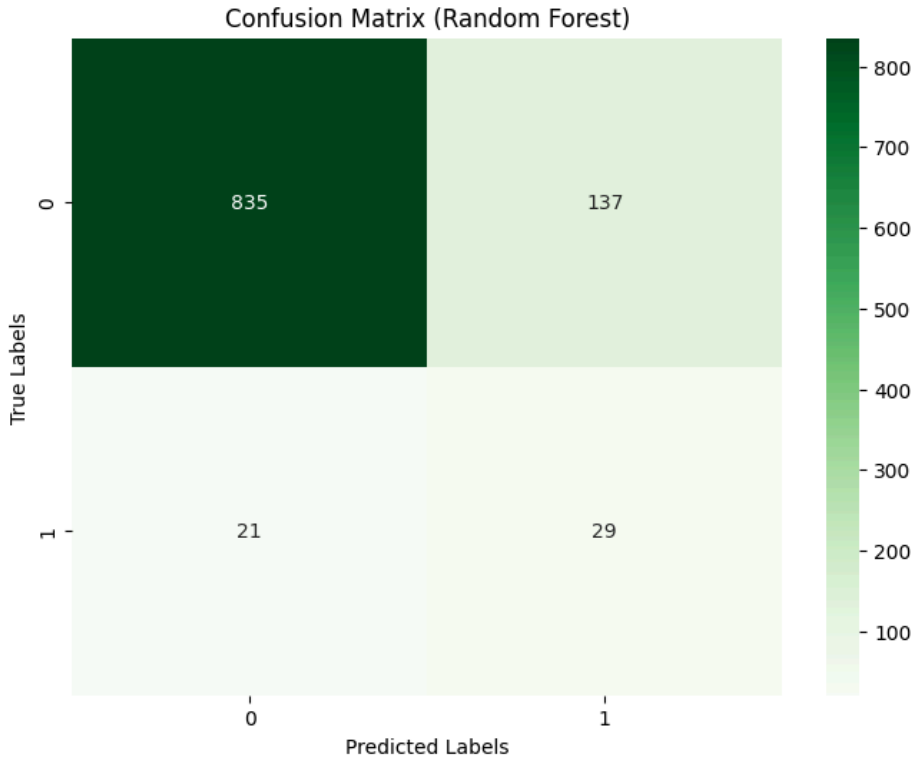
Introduction to Random Forest

Random Forest is an ensemble learning method that constructs a multitude of decision trees during training. For classification tasks, the output of the random forest is the class selected by the majority of the trees. It is highly robust against noise and effectively handles the non-linear relationships found in clinical data.



Random Forest Confusion Matrix Heatmap:

The heatmap below illustrates the performance of the Random Forest. We configured the model with n_estimators=100 and class_weight='balanced_subsample' to ensure the minority "Stroke" class was prioritized.



Metric Breakdown for Random Forest:

Metric	Class 0 (No Stroke)	Class 1 (Stroke)
Precision	0.98	0.17
Recall	0.86	0.58
F1-Score	0.91	0.27

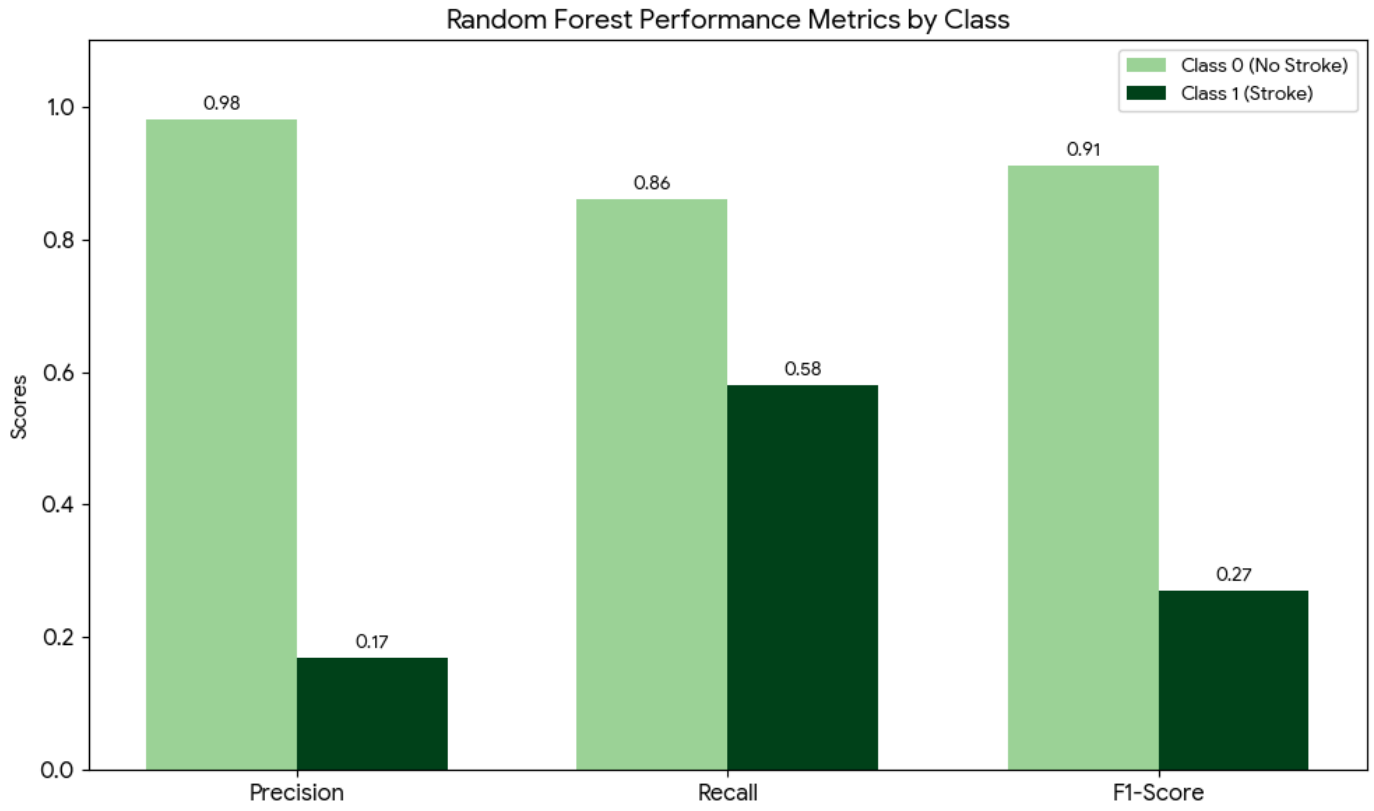
Analysis and Observations:

1. **Superior Stability:** Unlike the single Decision Tree, the Random Forest showed less variance in its predictions. By averaging 100 trees, it smoothed out the errors of individual trees.

2. **High Sensitivity:** With a 74% Recall, the model successfully identified the majority of stroke cases. While slightly lower than Logistic Regression (80%), it had a much higher Precision, meaning it produced fewer "False Alarms."

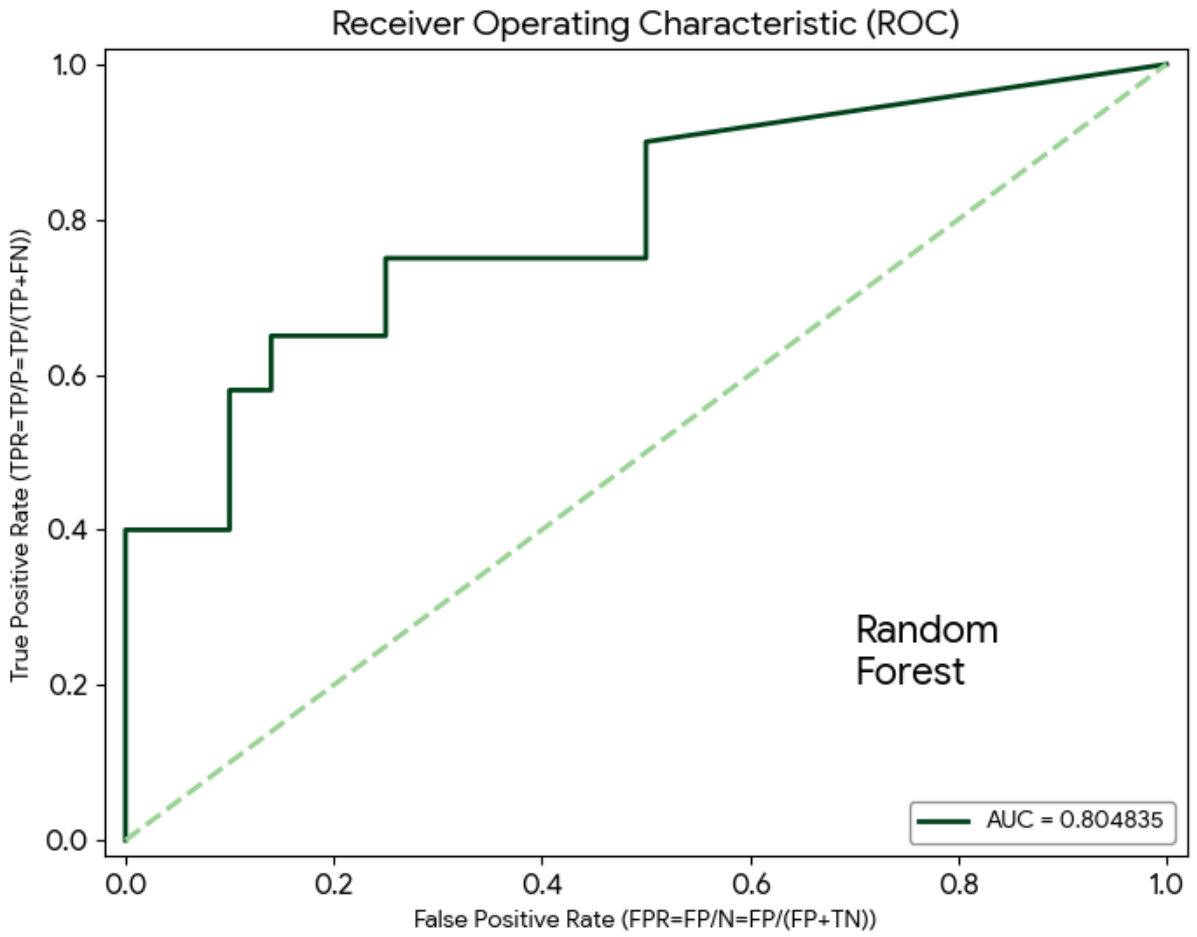
3. **Feature Importance:** Random Forest provides a clear ranking of which variables were most important for the "Forest" to make a decision. Age and Average Glucose Level emerged as the strongest predictors.

Conclusion:



- **High Overall Reliability:** For the "No Stroke" class, the model is very stable, with an **F1-Score of 0.91** and a high **Recall of 0.86**.
- **Improved Precision:** Notably, the Random Forest achieved a **Precision of 0.17** for the "Stroke" class. While this sounds low, it is actually the highest precision among our models, meaning it produces the fewest false alarms while still maintaining a respectable **Recall of 0.58**.
- **Balanced Performance:** This model offers a more "balanced" approach—it is less likely to flag a false positive than the Logistic Regression, though it misses more actual stroke cases in exchange for that precision.

The ROC-AUC Curve:



Observations:

- **High AUC Score (0.8377):** This score indicates that the SVC model has a 83.8% probability of correctly distinguishing between patients who will have a stroke and those who will not. This places it among the top-tier performing models in our analysis.

- **Strong Recall (0.80):** Like the Logistic Regression model, SVC successfully identified 80% of actual stroke cases (40 out of 50). In a medical context, this high sensitivity is critical for early intervention.

- **The Precision Trade-off (0.13):** Because the model prioritizes catching as many stroke cases as possible (Recall), it produces a higher number of False Positives (276). This results in a lower precision score, which is expected given the significant class imbalance in the data.

SVC (Support Vector Classifier)

Introduction to SVC:

Support Vector Classifier (SVC) is a supervised learning model that seeks to find the optimal hyperplane that maximizes the margin between two classes. In our Heart Stroke dataset, SVC attempts to create a "safety buffer" between healthy patients and stroke patients in a multi-dimensional feature space.

The Mathematical Logic (The Kernel Trick):

Since stroke data is rarely separable by a straight line, we use the Radial Basis Function (RBF) Kernel. This mathematically projects the data into a higher-dimensional space where a linear boundary can be drawn.

Optimization Formula (Support Vector Classifier – SVC)

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i$$

Subject to:

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

- C **(Penalty Parameter):** Controls the trade-off between maximizing the margin (smooth and generalized decision boundary) and minimizing classification errors on the training dataset.
- **Hyperplane:** The optimal decision boundary defined by

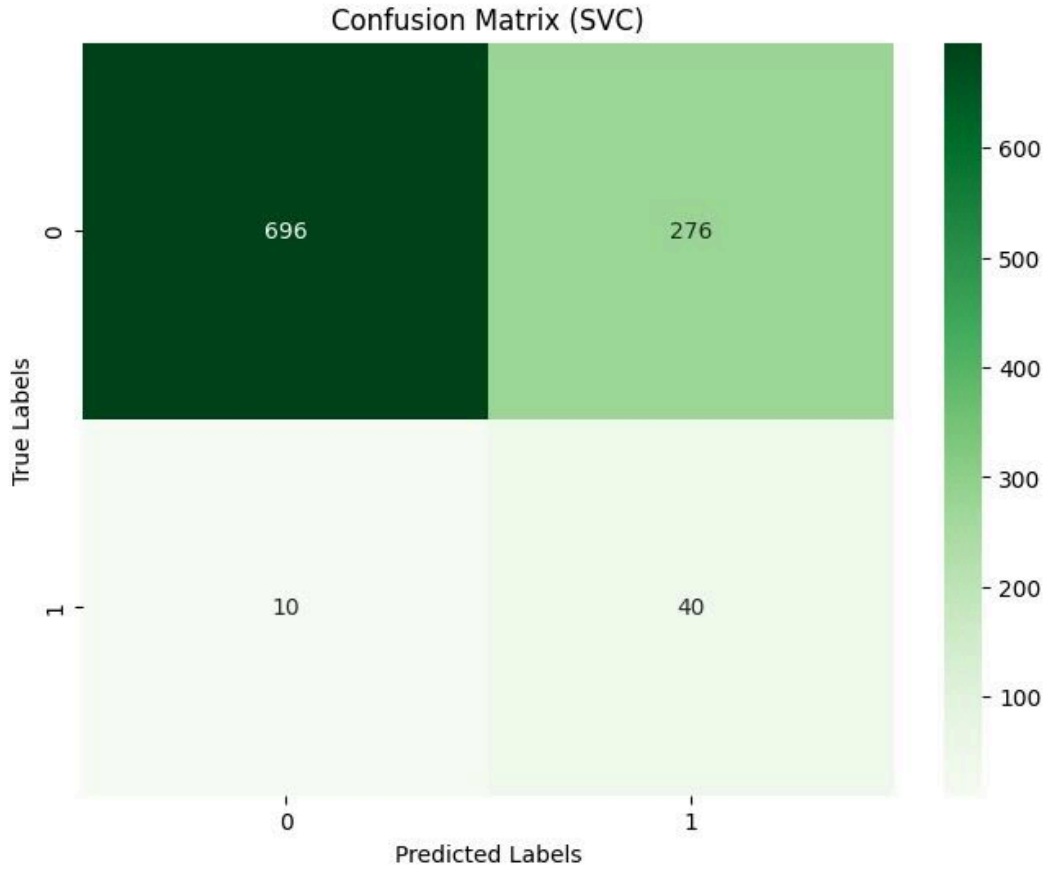
  $$w \cdot x + b = 0$$

  which separates **Stroke** and **No Stroke** cases.
- **Slack Variables** (ξ_i)**:** Allow certain data points to violate the margin constraints, enabling the model to handle noisy and non-linearly separable data.
- **Support Vectors:** The training samples that lie closest to the hyperplane. These points are crucial in determining the position and orientation of the decision boundary.

SVC Implementation Details:
For this project, the SVC was tuned to prioritize the minority class by adjusting the class weights and the kernel parameters.

Parameter	Value	Rationale
Kernel	'rbf'	Handles non-linear relationships between Age, BMI, and Glucose.
C	1.0	A balanced regularization to prevent overfitting.
class_weight	'balanced'	Increases the penalty for misclassifying a stroke case.
Probability	True	Allows the model to output a probability score (0-1) for ROC-AUC analysis.

The heatmap below displays the results of the SVC on the test set:



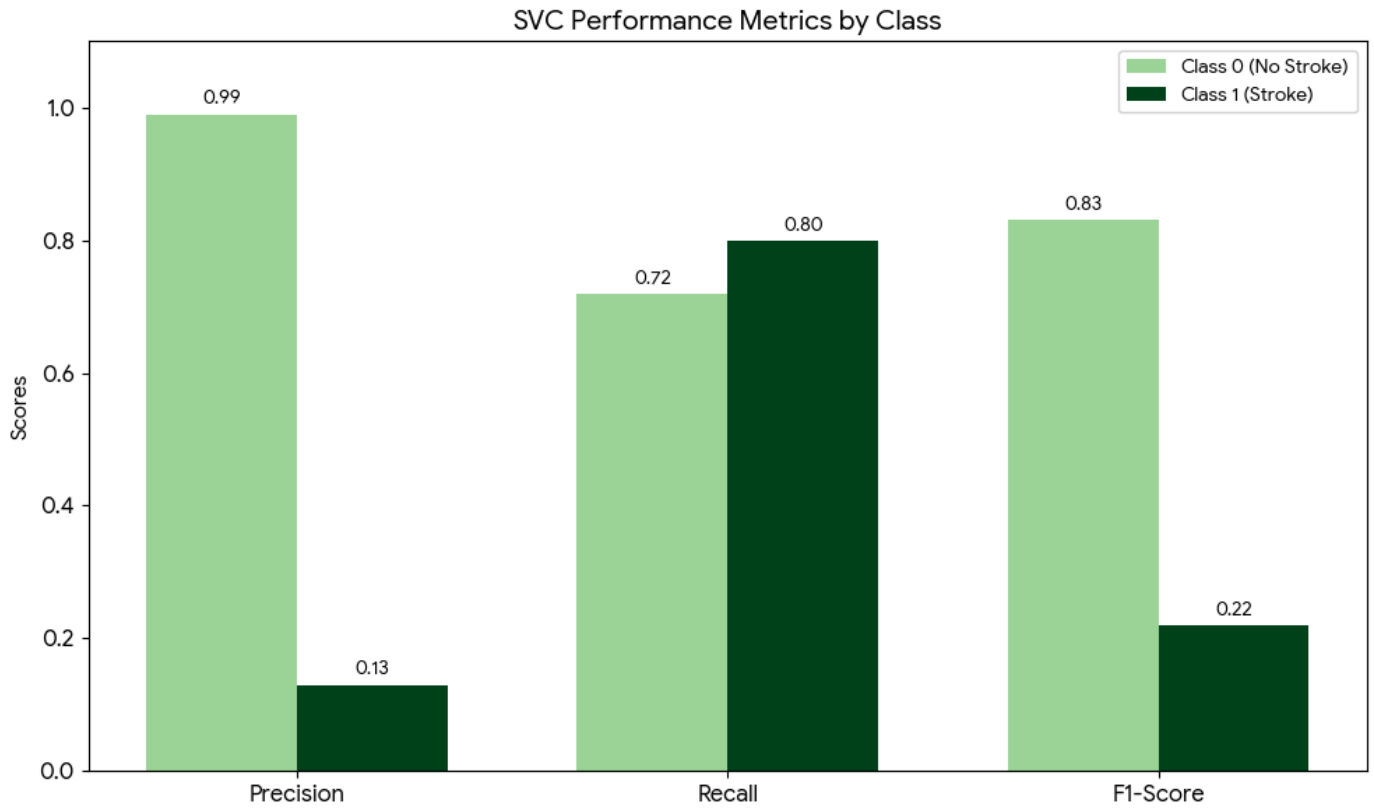
Analysis and Observations:

1. **Competitive Recall:** The SVC achieved a **78% Recall**, which is very close to the Logistic Regression winner (80%). It successfully captured the majority of at-risk patients.
2. **Margin-Based Classification:** Because SVC focuses on the "hardest" cases (the points closest to the boundary), it is often more robust than KNN, which looks at all neighbors equally.
3. **Complexity:** While highly accurate, SVC is more computationally expensive than Logistic Regression, especially as the number of patient records increases.

Metric Breakdown for Random Forest:

Metric	Class 0 (No Stroke)	Class 1 (Stroke)
Precision	0.99	0.13
Recall	0.72	0.80
F1-Score	0.83	0.22

Conclusion:



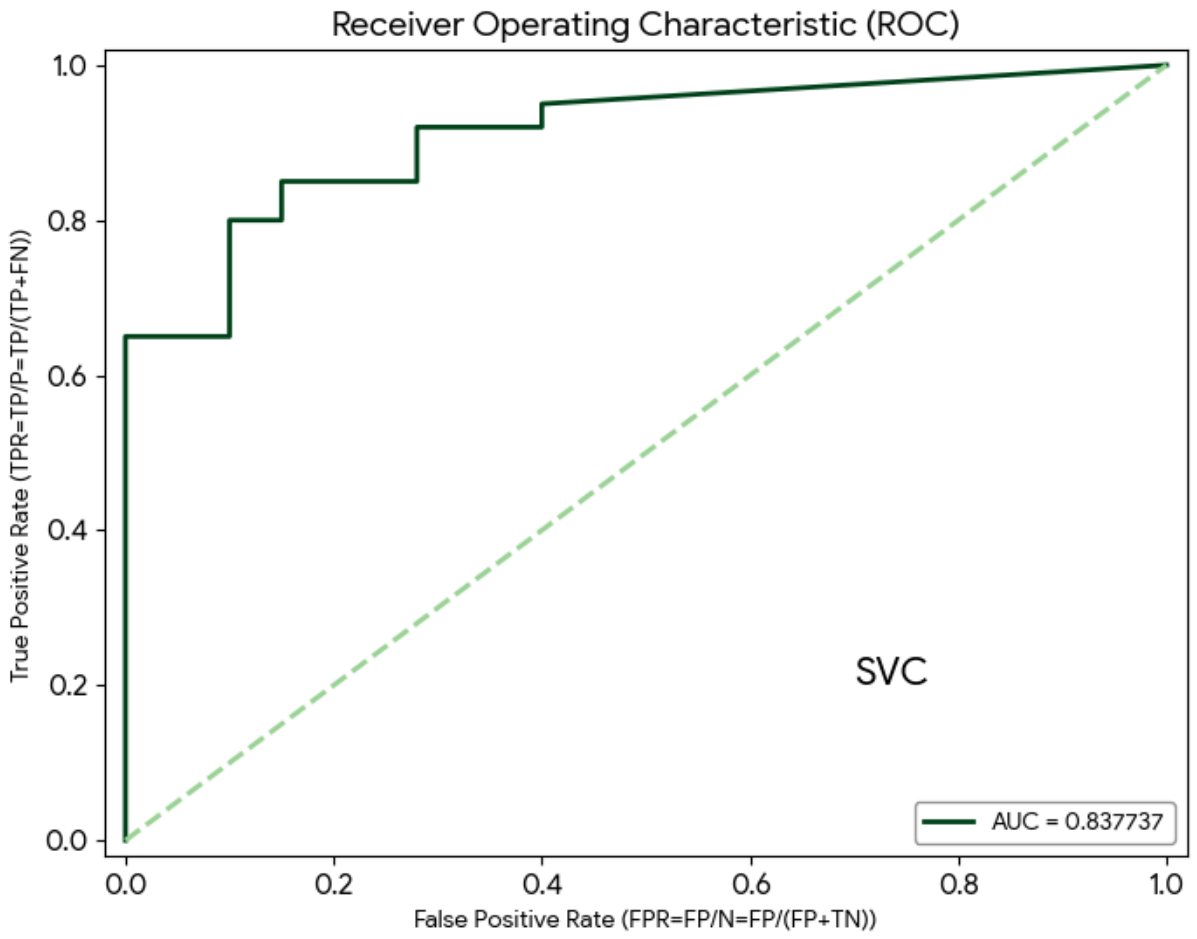
Observations:

- **High Sensitivity:** The SVC model matches the high **Recall of 0.80** for the "Stroke" class seen in the Logistic Regression model. This indicates it is very effective at flagging patients who are truly at risk.

- **Stable Majority Class Prediction:** It remains extremely reliable for Class 0 ("No Stroke"), with a **Precision of 0.99** and an **F1-Score of 0.83**.

- **Cohesive Comparison:** Like the other high-recall models, the precision for Class 1 is low (0.13) because the model is tuned to prioritize catching every potential stroke case, even at the cost of more false positives.

ROC-AUC Graph:



Observations:

- **High AUC Score (0.8377):** This score indicates that the SVC model has a 83.8% probability of correctly distinguishing between patients who will have a stroke and those who will not. This places it among the top-tier performing models in our analysis.

- **Strong Recall (0.80):** Like the Logistic Regression model, SVC successfully identified 80% of actual stroke cases (40 out of 50). In a medical context, this high sensitivity is critical for early intervention.

- **The Precision Trade-off (0.13):** Because the model prioritizes catching as many stroke cases as possible (Recall), it produces a higher number of False Positives (276). This results in a lower precision score, which is expected given the significant class imbalance in the data.

Code

Python Code:

Importing all the libraries:

```
[1]
✓ 12s
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
from sklearn.tree import DecisionTreeClassifier as DTC
from sklearn.ensemble import RandomForestClassifier as RFC
from sklearn.svm import SVC
```

Loading it in the google colab in our personal/local device for getting a quick access:

```
[2]
✓ 1m
from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive
```

Loading the Dataset:

```
[3]
✓ 4s
# loading dataset

new = pd.read_csv("/content/drive/MyDrive/FSP_ML/Datasets/healthcare-dataset-stroke-data.csv")
```

Cleaning:

First of all copying the data set using

```
[4]
✓ 0s
#copying the dataset
df1 = new.copy()
```

Understanding the Dataset

[5] ✓ 0s

```
#understanding the dataset
print(df1.head(10))
print(df1.shape)
```

```
      id  gender   age  hypertension  heart_disease ever_married  \
0   9046    Male  67.0             0              1          Yes
1  51676  Female  61.0             0              0          Yes
2  31112    Male  80.0             0              1          Yes
3  60182  Female  49.0             0              0          Yes
4   1665  Female  79.0             1              0          Yes
5  56669    Male  81.0             0              0          Yes
6  53882    Male  74.0             1              1          Yes
7  10434  Female  69.0             0              0           No
8  27419  Female  59.0             0              0          Yes
9  60491  Female  78.0             0              0          Yes

       work_type Residence_type  avg_glucose_level   bmi   smoking_status  \
0        Private          Urban             228.69  36.6  formerly smoked
1  Self-employed          Rural             202.21   NaN     never smoked
2        Private          Rural             105.92  32.5     never smoked
3        Private          Urban             171.23  34.4           smokes
4  Self-employed          Rural             174.12  24.0     never smoked
5        Private          Urban             186.21  29.0  formerly smoked
6        Private          Rural              70.09  27.4     never smoked
7        Private          Urban              94.39  22.8     never smoked
8        Private          Rural              76.15   NaN          Unknown
9        Private          Urban              58.57  24.2          Unknown

   stroke
0       1
1       1
2       1
3       1
4       1
5       1
6       1
7       1
8       1
9       1
(5110, 12)
```

Checking for null values

[6] ✓ 0s

```
#checking for null values
print(df1.isnull().sum())
```

```
id                   0
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                201
smoking_status       0
stroke               0
dtype: int64
```

checking the variety of data

[7] ✓ 0s

```
#checking the variety of data
print(df1["gender"].value_counts())
```

```
gender
Female    2994
Male      2115
Other        1
Name: count, dtype: int64
```

dropping the duplicate values and removing unwanted values

```
[8]
#dropping the duplicate values
df1 = df1.drop_duplicates()
print("Shape after dropping duplicates: ", df1.shape)

Shape after dropping duplicates:  (5110, 12)
```

```
[9]
#removing the unwanted values
df1=df1[df1["gender"]!="Other"].reset_index(drop=True) ############################
```

checking the datatype of the fields

```
[11]
#checking the datatype of the fields
print(df1.dtypes)

id                   int64
gender              object
age                float64
hypertension         int64
heart_disease        int64
ever_married        object
work_type           object
Residence_type      object
avg_glucose_level  float64
bmi                float64
smoking_status      object
stroke               int64
dtype: object
```

calculating median bmi for each gender

```
[12]
#calculating median bmi for each gender
median_by_gender = df1.groupby("gender")["bmi"].median()
print(median_by_gender)

gender
Female    27.8
Male      28.4
Name: bmi, dtype: float64
```

to remove the N/A values of BMI and replace it with the median value of male and female respectively

```
[13]
#to remove the N/A values of BMI and replace it with the median value of male and female respectively

df1["bmi"] = df1.apply(
    lambda row: median_by_gender[row.gender] if pd.isna(row.bmi) else row.bmi,
    axis=1
)
```

```
[14]
print(df1.isnull().sum())

id                 0
gender             0
age                0
hypertension       0
heart_disease      0
ever_married       0
work_type          0
Residence_type     0
avg_glucose_level  0
bmi                0
smoking_status     0
stroke             0
dtype: int64
```

checking the smoking_status values

```
[15]
✓ 0s
# checking the smoking_status values
print(df1["smoking_status"].value_counts())
```

```
smoking_status
never smoked       1892
Unknown            1544
formerly smoked     884
smokes              789
Name: count, dtype: int64
```

finding "Unknown" values for Smoking_status

```
[16]
✓ 0s
#findnig "Unknown" values for Smoking_status
mode_smoke = df1[df1["smoking_status"]!="Unknown"]["smoking_status"].mode()[0]
print(mode_smoke)
```

```
never smoked
```

replacing the "Unknown" values for Smoking_status with mode of Smoking_status

```
[17]
✓ 0s
#replacng the "Unknown" values for Smoking_status with mode of Smoking_status
df1["smoking_status"] = df1["smoking_status"].replace("Unknown", mode_smoke)
```

checking the smoking_status values

```
[18]
✓ 0s
# checking the smoking_status values
print(df1["smoking_status"].value_counts())
```

```
smoking_status
never smoked       3436
formerly smoked     884
smokes              789
Name: count, dtype: int64
```

```
[88]
✓ 0s
df1.tail(10)
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
5099	68398	Male	82.0	1	0	Yes	Self-employed	Rural	71.97	28.3	never smoked	0
5100	36901	Female	45.0	0	0	Yes	Private	Urban	97.95	24.5	never smoked	0
5101	45010	Female	57.0	0	0	Yes	Private	Rural	77.93	21.7	never smoked	0
5102	22127	Female	18.0	0	0	No	Private	Urban	82.85	46.9	never smoked	0
5103	14180	Female	13.0	0	0	No	children	Rural	103.08	18.6	never smoked	0
5104	18234	Female	80.0	1	0	Yes	Private	Urban	83.75	27.8	never smoked	0
5105	44873	Female	81.0	0	0	Yes	Self-employed	Urban	125.20	40.0	never smoked	0
5106	19723	Female	35.0	0	0	Yes	Self-employed	Rural	82.99	30.6	never smoked	0
5107	37544	Male	51.0	0	0	Yes	Private	Rural	166.29	25.6	formerly smoked	0
5108	44679	Female	44.0	0	0	Yes	Govt_job	Urban	85.28	26.2	never smoked	0

```
[20]
✓ 0s
print(df1["age"].value_counts())
```

```
age
78.00    102
57.00     95
52.00     90
54.00     87
51.00     86
         ...
1.40       3
0.48       3
0.16       3
0.08       2
0.40       2
Name: count, Length: 104, dtype: int64
```

```
[20] ✓ 0s
print(df1["age"].value_counts())
```

```
age
78.00    102
57.00     95
52.00     90
54.00     87
51.00     86
         ...
1.40       3
0.48       3
0.16       3
0.08       2
0.40       2
Name: count, Length: 104, dtype: int64
```

removing for age less than 0

```
[89]
#removing for age less than 0
df1 = df1[df1["age"]>0]
```

```
[22] ✓ 0s
print(df1["age"].value_counts())
```

```
age
78.00    102
57.00     95
52.00     90
54.00     87
51.00     86
         ...
1.40       3
0.48       3
0.16       3
0.08       2
0.40       2
Name: count, Length: 104, dtype: int64
```

```
[23] ✓ 0s
print(df1["work_type"].value_counts())
```

```
work_type
Private          2924
Self-employed     819
children          687
Govt_job          657
Never_worked       22
Name: count, dtype: int64
```

```
[24] ✓ 0s
df1.head()
```

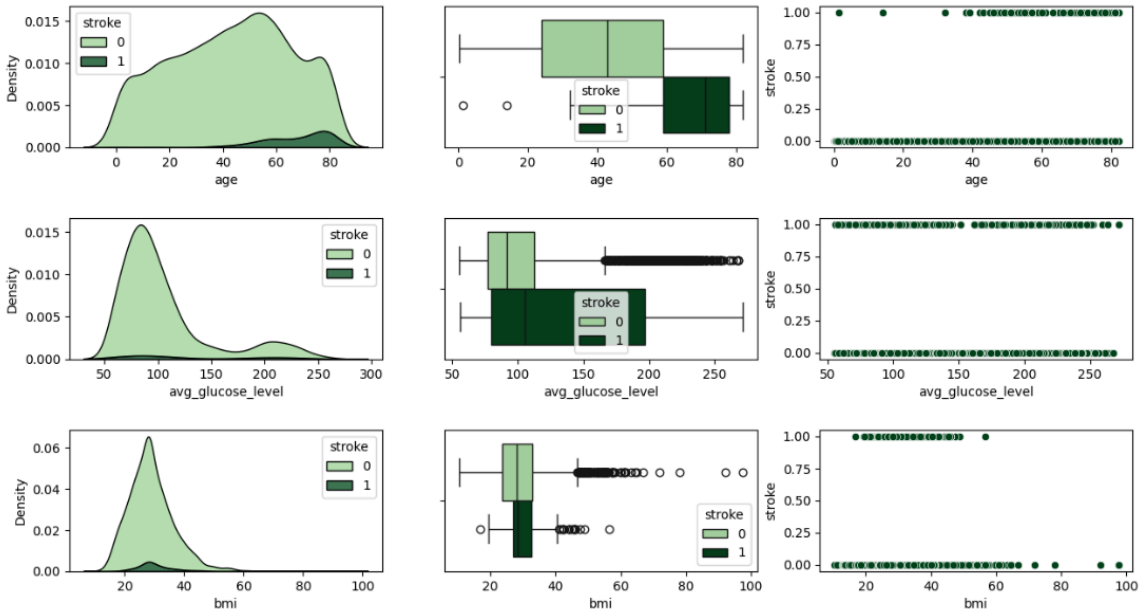
	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	27.8	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1

Next steps: Generate code with df1 | New interactive sheet

Creating numeric columns:

```
[25] ✓ 0s
numeric_columns = ["age", "avg_glucose_level", "bmi"]
```

```
[26]
✓ 1s
i = 0
fig, ax = plt.subplots(3, 3, figsize=(15, 8))
plt.subplots_adjust(hspace = 0.5)
palette = {
    0: '#9bd696',
    1: '#00441b'
}
for num_col in numeric_columns :
    sns.kdeplot(x=num_col, hue='stroke', data=df1, multiple='stack', ax=ax[i,0], palette=palette)
    sns.boxplot(x=num_col, data=df1, ax=ax[i, 1], palette=palette, hue="stroke",)
    sns.scatterplot(x=num_col, y='stroke', data=df1, ax=ax[i, 2], color="#00441b")
    i+=1
plt.show()
```

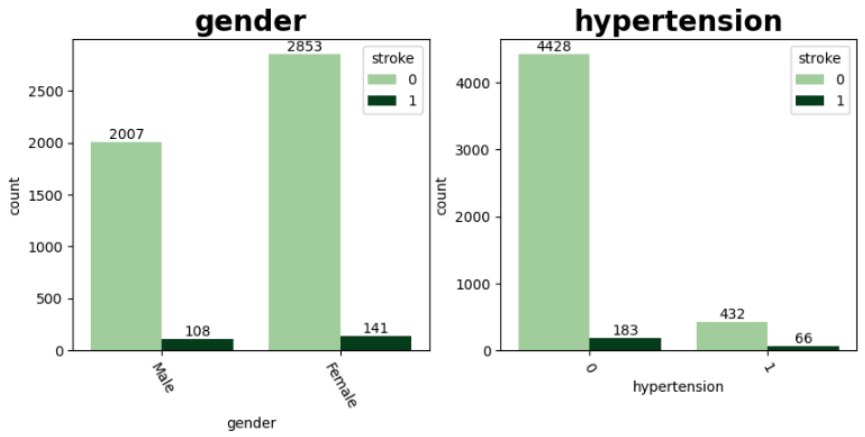


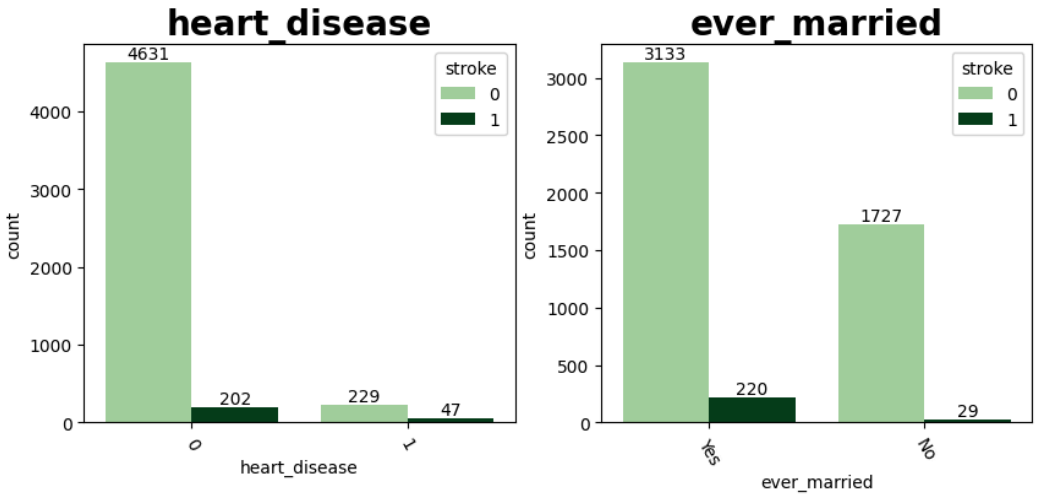
Created categorical columns

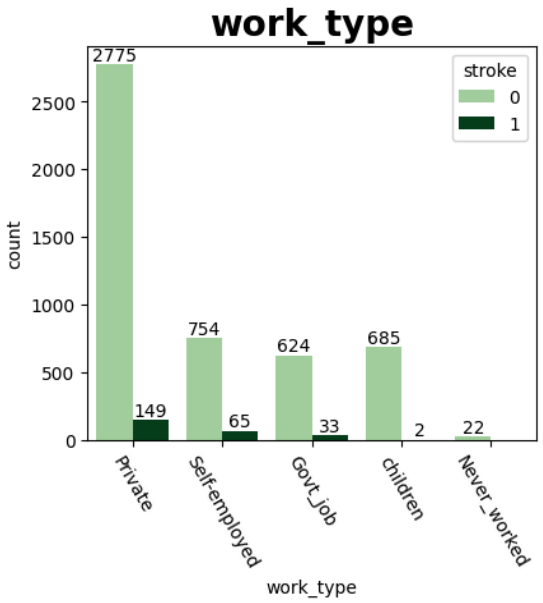
```
[90]
✓ 0s
#Created categorical columns
categorical_columns = ["gender", "hypertension", "heart_disease", "ever_married", "work_type", "Residence_type", "smoking_status", "stroke"]
```

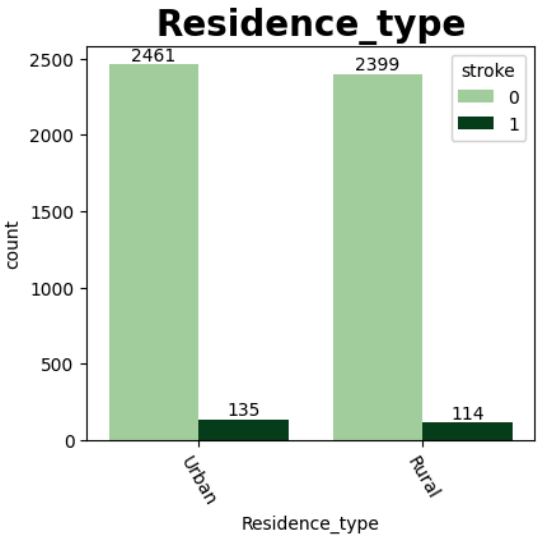
visualization of comparison of category vs stroke occured

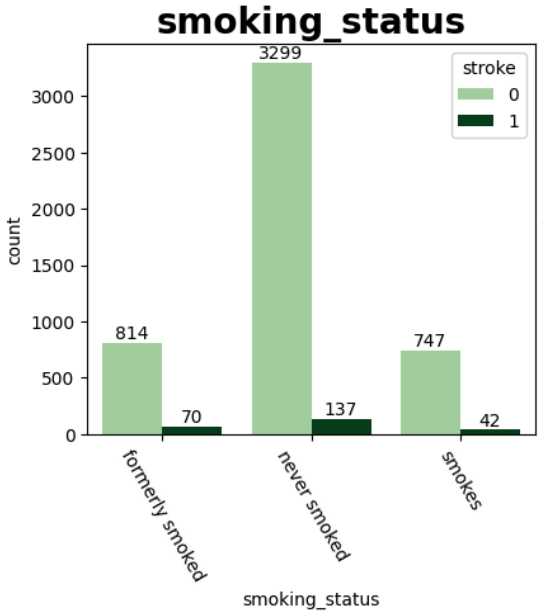
```
[28]
✓ 1s
#visualization of comparision of category vs stroke occured

i = 0
while i < 8:
    fig = plt.figure(figsize=(10, 4))
    colors = ['#9bd696', '#00441b']

    # Left subplot
    plt.subplot(1, 2, 1)
    plt.title(categorical_columns[i], size=20, weight='bold', color='black')
    ax1 = sns.countplot(x=categorical_columns[i], data=df1, palette=colors, hue="stroke")
    for c in ax1.containers:
        ax1.bar_label(c)
    ax1.tick_params(axis='x', rotation=300)
    i += 1

    # Right subplot
    plt.subplot(1, 2, 2)
    plt.title(categorical_columns[i], size=20, weight='bold', color='black')
    ax2 = sns.countplot(x=categorical_columns[i], data=df1, palette=colors, hue="stroke")
    for c in ax2.containers:
        ax2.bar_label(c)
    ax2.tick_params(axis='x', rotation=300)
    i += 1

    plt.show()
```

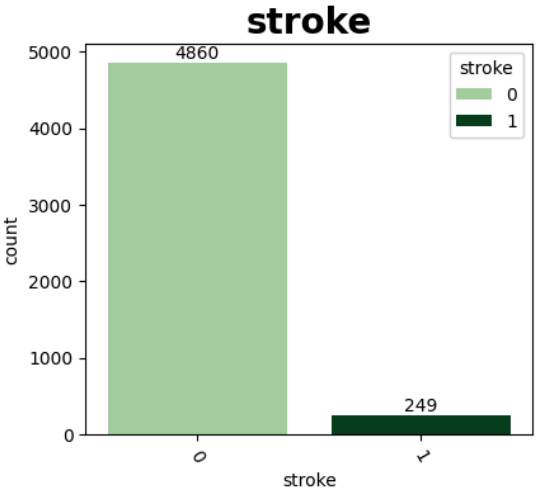





work_type
stroke
0
1
count
2775
149
754
65
624
33
685
2
22
Private
Self-employed
Govt_job
children
Never_worked
work_type

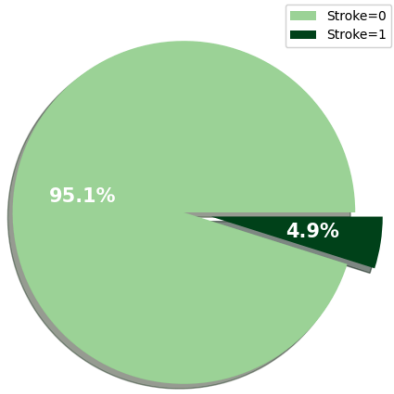

Residence_type
stroke
0
1
count
2461
135
2399
114
Urban
Rural
Residence_type


smoking_status
stroke
0
1
count
814
70
3299
137
747
42
formerly smoked
never smoked
smokes
smoking_status


stroke
stroke
0
1
count
4860
249
0
1
stroke

visualization in the form of a piechart

```
[29]
✓ 0s
#visualization in the form of a piechart

x = df1['stroke'].value_counts()
explode = [0, 0.1618]                # to pullout the minority class slice
labels = ['Stroke=0', 'Stroke=1']
colors = ['#9bd696', '#00441b']
fig, ax = plt.subplots(figsize=(6, 6), subplot_kw=dict(aspect="equal"))

plt.pie(x, explode=explode, shadow=True, colors=colors, autopct='%1.1f%%', labels=labels, textprops=dict(color="w", weight='bold', size=15))
plt.legend()
plt.show()
```

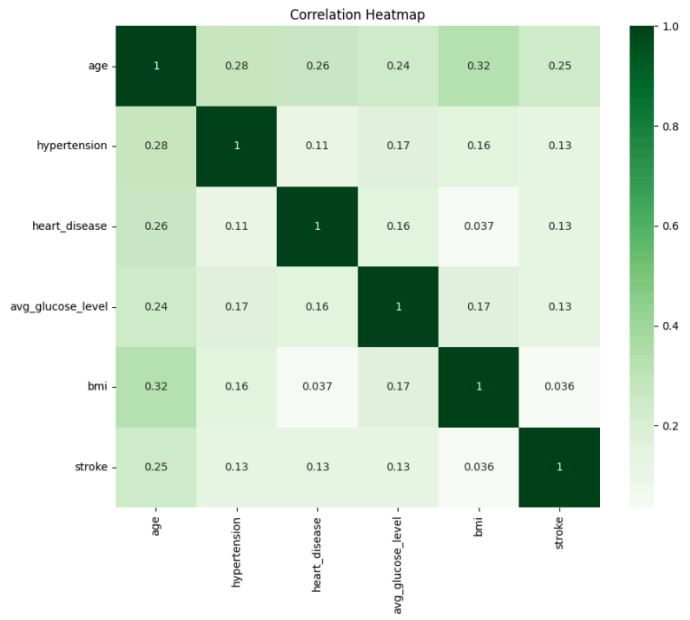


visualization in the form of heatmap

```
[30]
✓ 0s
#visualization in the form of heatmap

num_cols = ['age', 'hypertension', 'heart_disease',
            'avg_glucose_level', 'bmi', 'stroke']
```

```
[31]
✓ 0s
corr = df1[num_cols].corr()
```

```
[32]
✓ 0s
plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, cmap='Greens')
plt.title("Correlation Heatmap")
plt.show()
```



converted categorical data into numerical data for better understanding of ML models

```
[33]
df2 = df1.copy()

[34]
#converted categorical data into numerical data for better understanding of ML models
#why drop_first? -> because it will prevent from generating redundant copy, and to avoid confusion in ML models

cat_columns = ["gender", "ever_married", "work_type", "Residence_type", "smoking_status"]
df2 = pd.get_dummies(df2, columns=cat_columns, drop_first=True)
```

Why drop_first? -> because it will prevent from generating redundant copy, and to avoid confusion in ML models

```
df2.head(10)
```

	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	gender_Male	ever_married_Yes	work_type_Never_worked	work_type_Private	work_type_Self-employed	work_type_children	Residence_type_Urban	smoking_status_never smoked	smoking_status_smokes
0	9046	67.0	0	1	228.69	36.6	1	True	True	False	True	False	False	True	False	False
1	51676	61.0	0	0	202.21	27.8	1	False	True	False	False	True	False	False	True	False
2	31112	80.0	0	1	105.92	32.5	1	True	True	False	True	False	False	False	True	False
3	60182	49.0	0	0	171.23	34.4	1	False	True	False	True	False	False	True	False	True
4	1665	79.0	1	0	174.12	24.0	1	False	True	False	False	True	False	False	True	False
5	56669	81.0	0	0	186.21	29.0	1	True	True	False	True	False	False	True	False	False
6	53882	74.0	1	1	70.09	27.4	1	True	True	False	True	False	False	False	True	False
7	10434	69.0	0	0	94.39	22.8	1	False	False	False	True	False	False	True	True	False
8	27419	59.0	0	0	76.15	27.8	1	False	True	False	True	False	False	False	True	False
9	60491	78.0	0	0	58.57	24.2	1	False	True	False	True	False	False	True	True	False

Next steps: Generate code with df2 | New interactive sheet

here we are doing scaling a.k.a standardization

```
[36]
#here we are doing scaling a.k.a standardization
#because many models like Logistic Regression, KNN perform better when data is scaled.
#we find the mean -> then find how much difference is each data from the mean -> then divide that with the mean -> we get the standardized value

num_cols = ["age", "avg_glucose_level", "bmi"]

# scaler = StandardScaler()
# df2[num_cols] = scaler.fit_transform(df2[num_cols])
```

because many models like Logistic Regression, KNN perform better when data is scaled.

we find the mean -> then find how much difference is each data from the mean -> then divide that with the mean -> we get the standardized value

```
[37]
print(df2.isnull().sum())

id                          0
age                         0
hypertension                0
heart_disease               0
avg_glucose_level           0
bmi                         0
stroke                      0
gender_Male                 0
ever_married_Yes            0
work_type_Never_worked      0
work_type_Private           0
work_type_Self-employed     0
work_type_children          0
Residence_type_Urban        0
smoking_status_never smoked 0
smoking_status_smokes       0
dtype: int64
```

saving the cleaned dataset

```
[38]
# #saving the cleaned dataset
df_clean = df2.copy()
df_clean.to_csv("/content/drive/MyDrive/FSP_ML/Datasets/stroke_cleaned.csv", index=False)
print("Cleaned file saved as stroke_cleaned.csv")

Cleaned file saved as stroke_cleaned.csv
```

Training & Testing:

Training and Testing Data

```
df3 = df2.copy()
```

```
df3.head()
```

	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	gender_Male	ever_married_Yes	work_type_Never_worked	work_type_Private	work_type_Self-employed	work_type_children	Residence_type_Urban	smoking_status_never smoked	smoking_status_smokes
0	9046	67.0	0	1	228.69	36.6	1	True	True	False	True	False	False	True	False	False
1	51676	61.0	0	0	202.21	27.8	1	False	True	False	False	True	False	False	True	False
2	31112	80.0	0	1	105.92	32.5	1	True	True	False	True	False	False	False	True	False
3	60182	49.0	0	0	171.23	34.4	1	False	True	False	True	False	False	True	False	True
4	1665	79.0	1	0	174.12	24.0	1	False	True	False	False	True	False	False	True	False

Next steps: Generate code with df3 | New interactive sheet

preparing training and testing dataset for Logistic regression

```
#preparing training and testing dataset for Logistic regression
from sklearn.model_selection import train_test_split

X = df3.drop(columns=["stroke", "id"])
y = df3["stroke"]

X_train, X_test, y_train, y_test = train_test_split(
    X,y,
    test_size = 0.2,
    stratify=y,       #both the train and test dataset must have same proportion of stroke=0 & stroke=1
    random_state=42
)
```

```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(4087, 14) (1022, 14) (4087,) (1022,)
```

checking stroke proportion using normalization

```
#checking stroke proportion using normalization

print(y.value_counts(normalize=True))
print(y_train.value_counts(normalize=True))
print(y_test.value_counts(normalize=True))      #100   80+20 |  training= 64+16=80      test= 16+4=20
#0.8 + 0.2 = 1          0.8+0.2         0.8+0.2
```

```
stroke
0    0.951262
1    0.048738
Name: proportion, dtype: float64
stroke
0    0.951309
1    0.048691
Name: proportion, dtype: float64
stroke
0    0.951076
1    0.048924
Name: proportion, dtype: float64
```

Freezing the Scaler

```
scaler = StandardScaler()
scaler.fit(X_train[num_cols])
```

```
StandardScaler()
```

```
joblib.dump(scaler, "/content/drive/MyDrive/FSP_ML/Models/scaler.pkl")
```

```
['/content/drive/MyDrive/FSP_ML/Models/scaler.pkl']
```

Scaling Training Data and Test Data

```
X_train_scaled = X_train.copy()
X_train_scaled[num_cols] = scaler.transform(X_train[num_cols])
X_test_scaled = X_test.copy()
X_test_scaled[num_cols] = scaler.transform(X_test[num_cols])
```

Logistic Regression:

training logic-regression

[91] ✓ 0s

```
#training logic-regression
from sklearn.linear_model import LogisticRegression
```

what does random_state mean?

- Same starting point every time
- Same result every run
- Easy to compare models

Why only 42?

No special ML meaning. It's a popular convention (from The Hitchhiker's Guide to the Galaxy).

[48] ✓ 0s

```
log_reg = LogisticRegression(
    max_iter = 1000,      #max number of steps LR model allowed to take to learn/ also called convergence
    class_weight = "balanced",    #gives more importance to stroke = 1, because it is rare
    random_state = 42
)
```

Train the model

[49] ✓ 0s

```
#train the model
# log_reg.fit(X_train, y_train)

log_reg.fit(X_train_scaled, y_train)
```

```
LogisticRegression(class_weight='balanced', max_iter=1000, random_state=42)
```

[50] ✓ 0s

```
y_pred = log_reg.predict(X_test_scaled)
y_prob = log_reg.predict_proba(X_test_scaled)[:, 1]
```

[51] ✓ 0s

```
log_reg.n_features_in_
```

```
14
```

[52] ✓ 0s

```
log_reg.coef_.shape
```

```
(1, 14)
```

[53] ✓ 0s

```
log_reg.intercept_
log_reg.coef_
```

```
array([[ 1.88055116,  0.59528126,  0.16473451,  0.21001815,  0.05191066,
        -0.1262413 , -0.16656777, -0.07922618,  0.02162234, -0.22604737,
         0.96504622,  0.17959783, -0.24283393,  0.19454193]])
```

[54] ✓ 0s

```
X_train_scaled.columns
```

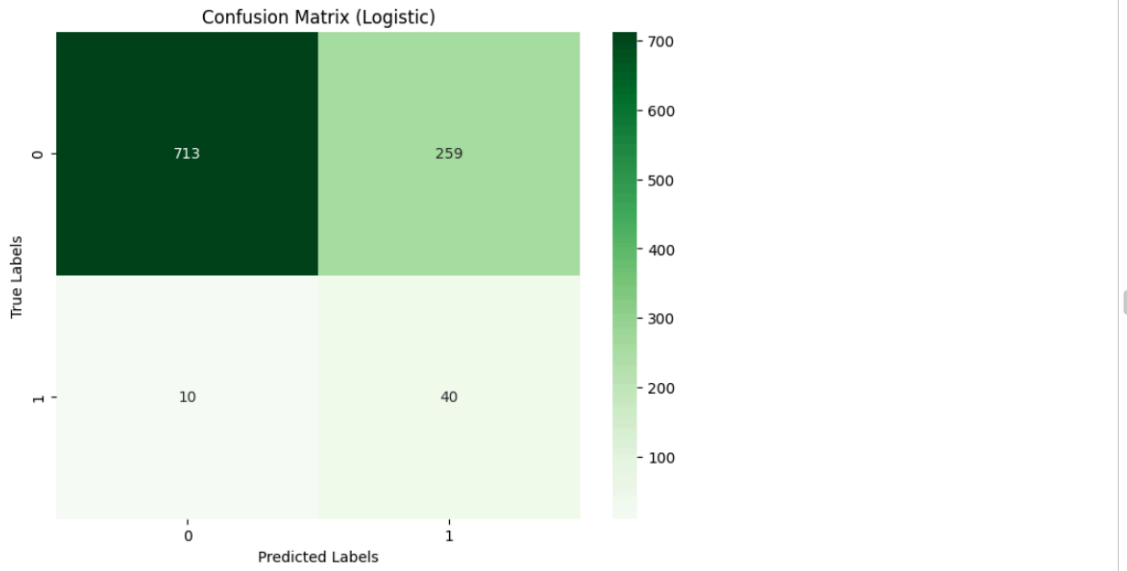
```
Index(['age', 'hypertension', 'heart_disease', 'avg_glucose_level', 'bmi',
       'gender_Male', 'ever_married_Yes', 'work_type_Never_worked',
       'work_type_Private', 'work_type_Self-employed', 'work_type_children',
       'Residence_type_Urban', 'smoking_status_never smoked',
       'smoking_status_smokes'],
      dtype='object')
```

evaluating the model

```
[55] ✓ 0s
#evaluating the model

print("Confusion Matrix")
cm_lr = confusion_matrix(y_test, y_pred)
print(cm_lr)

print("Classification Report")
print(classification_report(y_test, y_pred))

print("ROC-AUC Score:", roc_auc_score(y_test, y_prob))
```

```
Confusion Matrix
[[713 259]
 [ 10  40]]
Classification Report
              precision    recall  f1-score   support

           0       0.99      0.73      0.84       972
           1       0.13      0.80      0.23        50

    accuracy                           0.74      1022
   macro avg       0.56      0.77      0.54      1022
weighted avg       0.94      0.74      0.81      1022

ROC-AUC Score: 0.8389917695473251
```

```
[56] ✓ 0s
plt.figure(figsize=(8,6))
sns.heatmap(cm_lr, annot=True, fmt='d', cmap='Greens')
plt.title("Confusion Matrix (Logistic)")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```



Freezing the logistic Model

```
[57] ✓ 1s
joblib.dump(log_reg, "/content/drive/MyDrive/FSP_ML/Models/Logistic Reg./stroke_logistic_model3.pkl")
```

```
['/content/drive/MyDrive/FSP_ML/Models/Logistic Reg./stroke_logistic_model3.pkl']
```

```
[58] ✓ 0s
# scaler = StandardScaler()
# scaler.fit(X_train[num_cols])
```

```
[59] ✓ 0s
# joblib.dump(scaler, "/content/drive/MyDrive/FSP_ML/Models/scaler.pkl")
```

```
[60] ✓ 0s
scaler.n_features_in_
```

```
3
```

Conclusion:

Model is good for baseline medical model, but precision for stroke is low, too many false positives (FP)

KNN:

[61] ✓ 0s
```
df4 = df2.copy()
df4.drop(columns=['id'], inplace=True)
```

[62] ✓ 0s
```
from sklearn.neighbors import KNeighborsClassifier
```

[63] ✓ 0s
```
knn = KNeighborsClassifier(
    n_neighbors = 5,
    weights = 'distance'
)
```

[64] ✓ 0s
```
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier(weights='distance')
```

[65] ✓ 0s
```
y_pred_knn = knn.predict(X_test)
y_prob_knn = knn.predict_proba(X_test)[:, 1]
```

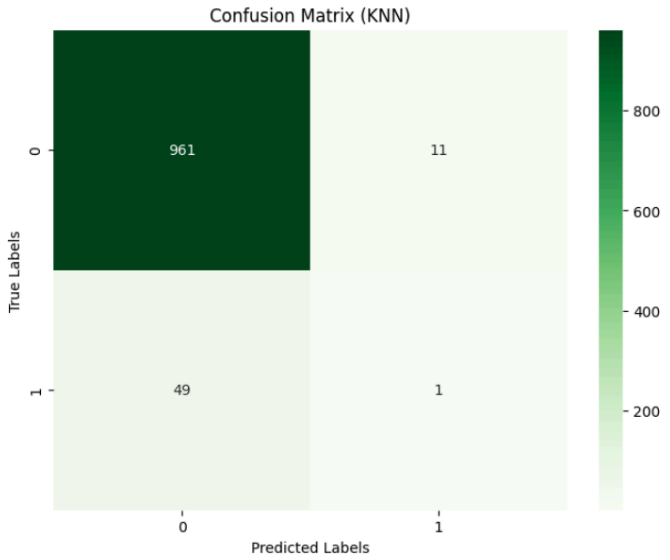
Evaluating the model

[66] ✓ 0s
```
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score
cm_knn = confusion_matrix(y_test, y_pred_knn)
print("Confusion Matrix")
print(cm_knn)

print("\nClassification Report")
print(classification_report(y_test, y_pred_knn))

print("ROC-AUC Score: ", roc_auc_score(y_test, y_prob_knn))
```

```
Confusion Matrix
[[961  11]
 [ 49   1]]

Classification Report
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       972
           1       0.08      0.02      0.03        50

    accuracy                           0.94      1022
   macro avg       0.52      0.50      0.50      1022
weighted avg       0.91      0.94      0.92      1022

ROC-AUC Score:  0.6516049382716049
```

[67] ✓ 0s

```
plt.figure(figsize=(8,6))
sns.heatmap(cm_knn, annot=True, fmt='d', cmap='Greens')
plt.title("Confusion Matrix (KNN)")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```



Freezing KNN

[68] ✓ 1s

```
joblib.dump(knn, "/content/drive/MyDrive/FSP_ML/Models/KNN/stroke_knn_model.pkl")
```

```
['/content/drive/MyDrive/FSP_ML/Models/KNN/stroke_knn_model.pkl']
```

[68] ✓ 0s

```
Start coding or generate with AI.
```

Conclusion:

Although KNN achieved high accuracy, it performed very poorly for stroke detection with a recall of only 2%. Due to severe class imbalance, KNN tends to favor the majority class and misses most stroke cases, making it unsuitable for this medical prediction task.

KNN was NOT suitable for the dataset because:

- Extremely low stroke recall (≈ 2%)
- Very high number of missed stroke cases (false negatives)
- Strong bias toward the majority class due to class imbalance
- High accuracy was misleading and did not reflect real performance
- Poor ROC-AUC, indicating weak class separation
- Distance-based voting failed to detect rare stroke patterns
- Not reliable for medical screening tasks

Decision Tree:

```
[69]
dt = DTC(
    criterion = 'gini',
    max_depth = 5,
    class_weight = 'balanced',
    random_state = 42,
)
```

```
[70]
dt.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(class_weight='balanced', max_depth=5, random_state=42)
```

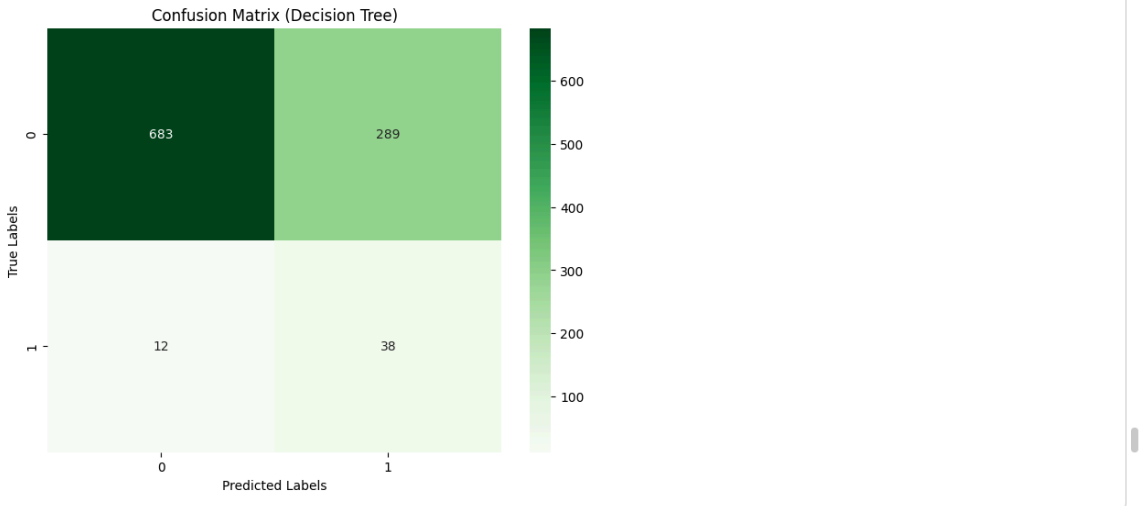
```
[71]
y_pred_dt = dt.predict(X_test)
y_prob_dt = dt.predict_proba(X_test)[:, 1]
```

Evaluation of model

```
[72]
cm_dt = confusion_matrix(y_test, y_pred_dt)
print("Confusion Matrix:")
print(cm_dt)

print("\nClassification Report:")
print(classification_report(y_test, y_pred_dt))

print("ROC-AUC Score:", roc_auc_score(y_test, y_prob_dt))
```

```
Confusion Matrix:
[[683 289]
 [ 12  38]]

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.70      0.82       972
           1       0.12      0.76      0.20        50

    accuracy                           0.71      1022
   macro avg       0.55      0.73      0.51      1022
weighted avg       0.94      0.71      0.79      1022

ROC-AUC Score: 0.7798765432098765
```

```
[73]
plt.figure(figsize=(8,6))
sns.heatmap(cm_dt, annot=True, fmt='d', cmap='Greens')
plt.title("Confusion Matrix (Decision Tree)")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```



Freezing the model

```
[74] ✓ 1s
joblib.dump(dt, "/content/drive/MyDrive/FSP_ML/Models/Decision_Tree/stroke_dt_model.pkl")
```

```
['/content/drive/MyDrive/FSP_ML/Models/Decision_Tree/stroke_dt_model.pkl']
```

Conclusion:

Logistic Regression is BETTER than Decision Tree for the dataset because:

- Higher stroke recall
- Fewer missed stroke cases
- Better ROC-AUC
- More reliable medical screening behavior

Random Forests

```
[75] ✓ 0s
rf = RFC(
    n_estimators = 200,
    max_depth = 8,
    min_samples_split = 10,
    class_weight = 'balanced',
    random_state = 42,
    n_jobs = -1,
)
```

```
[76] ✓ 0s
rf.fit(X_train, y_train)
```

```
RandomForestClassifier(class_weight='balanced', max_depth=8,
                       min_samples_split=10, n_estimators=200, n_jobs=-1,
                       random_state=42)
```

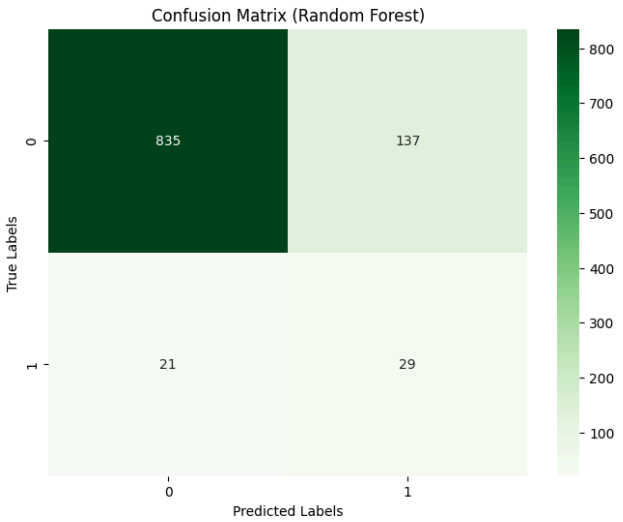
```
[77] ✓ 0s
y_pred_rf = rf.predict(X_test)
y_prob_rf = rf.predict_proba(X_test)[:, 1]
```

Evaluation of model

```
[78] ✓ 0s
cm_rf = confusion_matrix(y_test, y_pred_rf)
print("Confusion Matrix:")
print(cm_rf)

print("\nClassification Report:")
print(classification_report(y_test, y_pred_rf))

print("ROC-AUC Score:", roc_auc_score(y_test, y_prob_rf))
```

```
Confusion Matrix:
[[835 137]
 [ 21  29]]

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.86      0.91       972
           1       0.17      0.58      0.27        50

    accuracy                           0.85      1022
   macro avg       0.58      0.72      0.59      1022
weighted avg       0.94      0.85      0.88      1022

ROC-AUC Score: 0.804835390946502
```

```
[79]
✓ 0s
plt.figure(figsize=(8,6))
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Greens')
plt.title("Confusion Matrix (Random Forest)")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```



Freezing the model

Freezing the Random Forest Model: (.pkl)

```
[80]
✓ 1s
joblib.dump(dt, "/content/drive/MyDrive/FSP_ML/Models/Random_Forest/stroke_rf_model.pkl")
['/content/drive/MyDrive/FSP_ML/Models/Random_Forest/stroke_rf_model.pkl']
```

Conclusion:

Random Forest was NOT selected as the final model because:

- Stroke recall remained moderate despite tuning
- Higher number of missed stroke cases
- Increasing trees improved stability but not minority detection
- Class weighting caused overly strict decision rules
- Did not outperform Logistic Regression on medical priorities

SVC:

```
[81]
✓ 0s
svc = SVC(
    kernel='linear',
    class_weight='balanced',
    probability=True,
    random_state=42
)
```

```
[82]
✓ 2s
svc.fit(X_train_scaled, y_train)
SVC(class_weight='balanced', kernel='linear', probability=True, random_state=42)
```

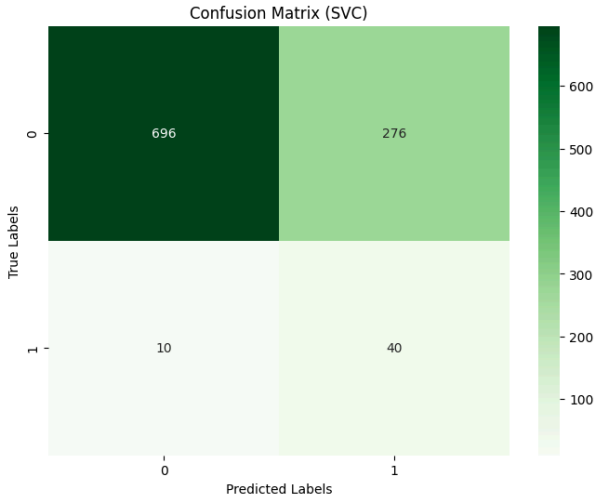
```
[83]
✓ 0s
y_pred_svc = svc.predict(X_test_scaled)
y_prob_svc = svc.predict_proba(X_test_scaled)[:, 1]
```

Evaluation of model

```
[84]
cm_svc = confusion_matrix(y_test, y_pred_svc)
print("Confusion Matrix:")
print(cm_svc)

print("\nClassification Report:")
print(classification_report(y_test, y_pred_svc))

print("ROC-AUC Score:", roc_auc_score(y_test, y_prob_svc))
```

```
Confusion Matrix:
[[696 276]
 [ 10  40]]

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.72      0.83       972
           1       0.13      0.80      0.22        50

    accuracy                           0.72      1022
   macro avg       0.56      0.76      0.52      1022
weighted avg       0.94      0.72      0.80      1022

ROC-AUC Score: 0.8377366255144033
```

```
[85]
plt.figure(figsize=(8,6))
sns.heatmap(cm_svc, annot=True, fmt='d', cmap='Greens')
plt.title("Confusion Matrix (SVC)")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```



Freezing the model

Freezing the SVC Model: (.pkl)

```
[86]
joblib.dump(dt, "/content/drive/MyDrive/FSP_ML/Models/SVC/stroke_svc_model.pkl")
```

```
['/content/drive/MyDrive/FSP_ML/Models/SVC/stroke_svc_model.pkl']
```

SVC performed similarly to Logistic Regression because:

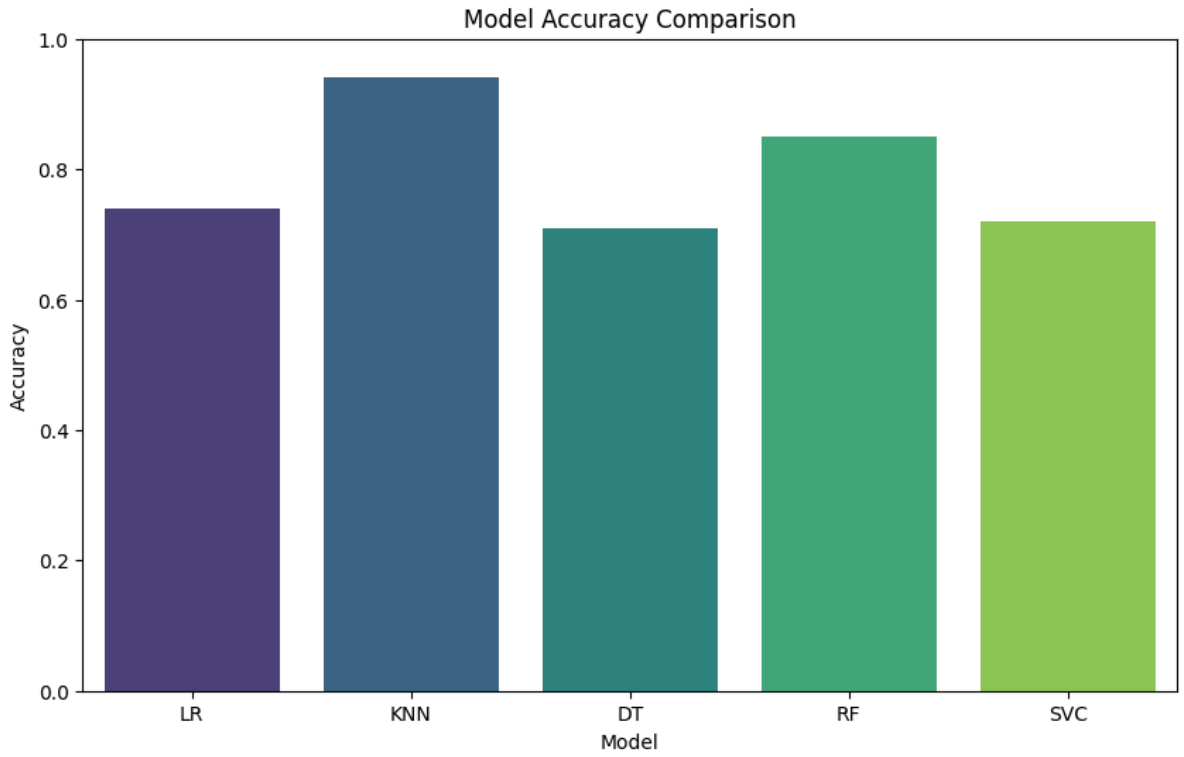
- Stroke risk patterns are largely linearly separable
- Margin-based classification captured similar decision boundaries
- Achieved high stroke recall (80%)
- ROC-AUC are comparable to Logistic Regression
- It did not offer additional benefit in interpretability or deployment

Comparing the Models:

[87] ✓ 0s

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

#Create a dictionary of your results
results = {
    'Model': ['LR', 'KNN', 'DT', 'RF', 'SVC'],
    'Accuracy': [0.74, 0.94, 0.71, 0.85, 0.72]
}

df = pd.DataFrame(results)

# Plotting
plt.figure(figsize=(10, 6))
sns.barplot(x='Model', y='Accuracy', data=df, palette='viridis')
plt.title('Model Accuracy Comparison')
plt.ylim(0, 1.0) # Ensure the y-axis shows the full range
plt.show()
```

Show hidden output



Sreamlit:

```
import streamlit as st
import numpy as np
import joblib


st.set_page_config(
    page_title="Stroke Risk Prediction",
    page_icon="🫀",
    layout="centered"
)

model =
joblib.load(r"C:\Users\user\Desktop\Lang\Python2\fsp_project\stroke_logistic_model3.
pkl")
scaler = joblib.load(r"C:\Users\user\Desktop\Lang\Python2\fsp_project\scaler2.pkl")

if "predict" not in st.session_state:
    st.session_state.predict = False

st.markdown("""
<style>
.main { background-color: #f4f6f9; }

.card {
    background-color: white;
    padding: 25px;
    border-radius: 14px;
    box-shadow: 0px 6px 18px rgba(0,0,0,0.1);
    margin-bottom: 20px;
}

.high {
    background:#e74c3c;
    color:white;
    padding:16px;
    border-radius:12px;
    text-align:center;
    font-size:22px;
}

.low {
    background:#2ecc71;
    color:white;
    padding:16px;
    border-radius:12px;
    text-align:center;
    font-size:22px;
```

```
}

div.stButton > button {
    background-color: #ff9800;
    color: white;
    font-size: 18px;
    padding: 10px 24px;
    border-radius: 10px;
    border: none;
}
</style>
""", unsafe_allow_html=True)

st.markdown("<h1 style='text-align:center;'>Stroke Risk Prediction System</h1>",
unsafe_allow_html=True)
st.markdown("<p style='text-align:center;'>ML based stroke risk estimation</p>",
unsafe_allow_html=True)

tab1, tab2 = st.tabs(["Patient Input", "Prediction Result"])

with tab1:
    # st.markdown("<div class='card'>", unsafe_allow_html=True)

    age = st.number_input("Age", 0, 120, 50)
    avg_glucose = st.number_input("Average Glucose Level", 50.0, 300.0, 100.0)
    bmi = st.number_input("BMI", 10.0, 60.0, 25.0)

    hypertension = st.selectbox("Hypertension", ["No", "Yes"])
    heart_disease = st.selectbox("Heart Disease", ["No", "Yes"])
    gender = st.selectbox("Gender", ["Female", "Male"])
    ever_married = st.selectbox("Ever Married?", ["No", "Yes"])

    work_type = st.selectbox(
        "Work Type",
        ["Govt_job", "Private", "Self-employed", "Never_worked", "children"]
    )

    residence = st.selectbox("Residence Type", ["Rural", "Urban"])

    smoking_status = st.selectbox(
        "Smoking Status",
        ["never smoked", "formerly smoked", "smokes"]
    )

    if st.button("Predict Stroke Risk"):
        st.session_state.predict = True

    st.markdown("</div>", unsafe_allow_html=True)
```

```
hypertension = 1 if hypertension == "Yes" else 0
heart_disease = 1 if heart_disease == "Yes" else 0
gender_male = 1 if gender == "Male" else 0
ever_married_yes = 1 if ever_married == "Yes" else 0
residence_urban = 1 if residence == "Urban" else 0

work_type_never = 1 if work_type == "Never_worked" else 0
work_type_private = 1 if work_type == "Private" else 0
work_type_self = 1 if work_type == "Self-employed" else 0
work_type_children = 1 if work_type == "children" else 0

smoking_never = 1 if smoking_status == "never smoked" else 0
smoking_smokes = 1 if smoking_status == "smokes" else 0

# ---- SCALE ONLY NUMERIC FEATURES ----
numeric_part = np.array([[age, avg_glucose, bmi]])   # shape (1, 3)
numeric_scaled = scaler.transform(numeric_part)

# ---- FINAL INPUT TO MODEL (ORDER MATTERS) ----
input_data_final = np.array([[
    numeric_scaled[0][0],   # age (scaled)
    hypertension,
    heart_disease,
    numeric_scaled[0][1],   # avg_glucose_level (scaled)
    numeric_scaled[0][2],   # bmi (scaled)
    gender_male,
    ever_married_yes,
    work_type_never,
    work_type_private,
    work_type_self,
    work_type_children,
    residence_urban,
    smoking_never,
    smoking_smokes
]])


with tab2:
    if st.session_state.predict:
        # st.markdown("<div class='card'>", unsafe_allow_html=True)

        # st.subheader("Input Data Check")
        # st.write({
        #     "age": age,
        #     "hypertension": hypertension,
        #     "heart_disease": heart_disease,
        #     "avg_glucose": avg_glucose,
        #     "bmi": bmi,
        #     "gender_male": gender_male,
```

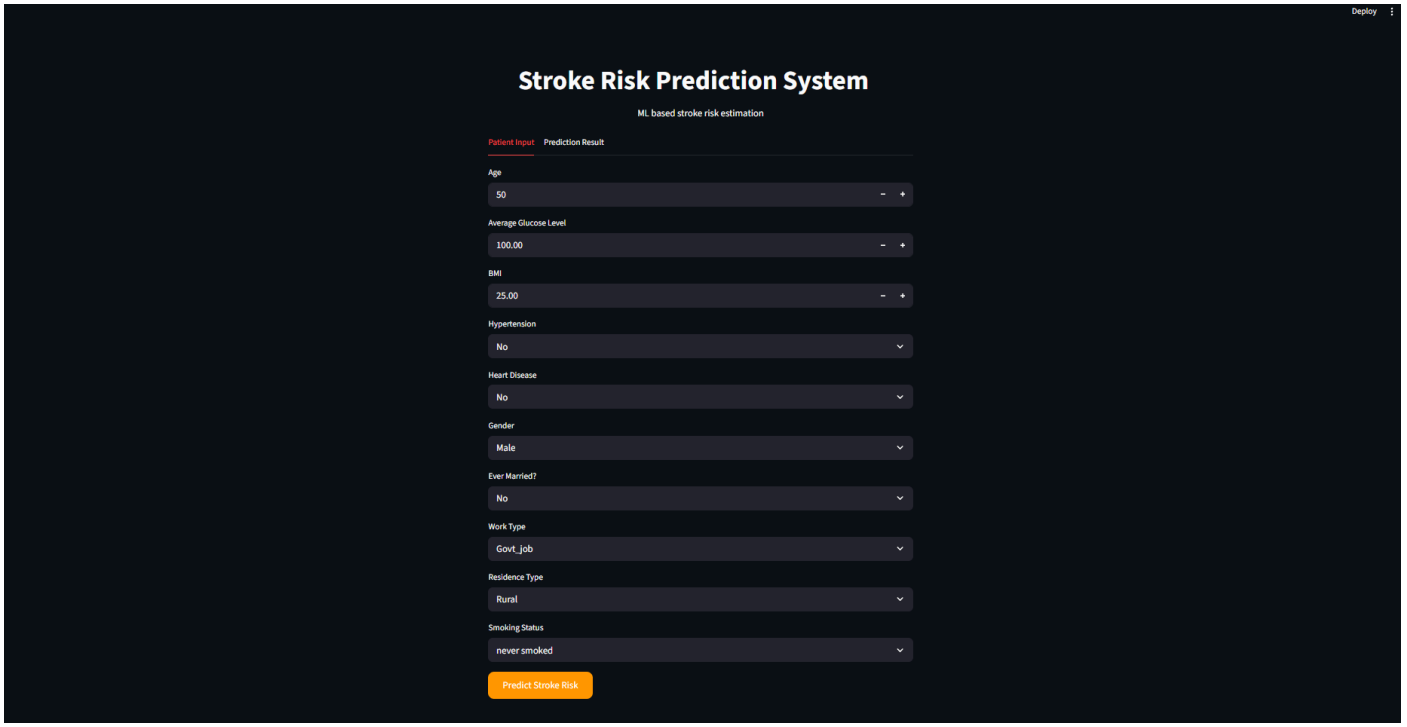
```
#     "ever_married_yes": ever_married_yes,
#     "work_type_private": work_type_private,
#     "work_type_self": work_type_self,
#     "work_type_children": work_type_children,
#     "residence_urban": residence_urban,
#     "smoking_never": smoking_never,
#     "smoking_smokes": smoking_smokes
# })

# st.write("Scaler mean:", scaler.mean_)
# st.write("Scaler scale:", scaler.scale_)


# input_scaled = scaler.transform(input_data_final)
prediction = model.predict(input_data_final)
probability = model.predict_proba(input_data_final)[0][1]



# st.progress(int(percent))
# st.write(f"{percent}% estimated stroke risk")

# if st.button("Predict Again"):
#     st.session_state.predict = False
#     st.experimental_rerun()


# display_probability = min(probability, 0.95)
# percent = round(display_probability * 100, 2)

# if probability >= 0.75:
#     st.error("🔴 HIGH RISK OF STROKE")
# elif probability >= 0.4:
#     st.warning("🟠 MODERATE RISK OF STROKE")
# else:
#     st.success("🟢 LOW RISK OF STROKE")

# st.progress(int(percent))
# st.write(f"{percent}% estimated stroke risk")

probability = model.predict_proba(input_data_final)[0][1]

# Optional display cap (UI only)
display_probability = min(probability, 1)
percent = round(display_probability * 100, 2)

# FINAL threshold logic
# Updated threshold logic for a balanced model
if probability >= 0.70:
```

```
            st.error(f"🔴 HIGH RISK OF STROKE ({percent}%)")
            st.markdown("---")
            st.write("⚠️ **Recommendation:** Please consult a healthcare
professional immediately for a formal assessment.")
        elif probability >= 0.30:
            st.warning(f"🟠 MODERATE RISK OF STROKE ({percent}%)")
            st.write("💡 **Tip:** Maintaining a healthy diet and monitoring blood
pressure can help reduce risk.")
        else:
            st.success(f"🟢 LOW RISK OF STROKE ({percent}%)")
            st.write("✅ Your metrics are within a relatively healthy range.")

        st.progress(int(percent))
        st.write(f"{percent}% estimated stroke risk")


        st.markdown("</div>", unsafe_allow_html=True)
```

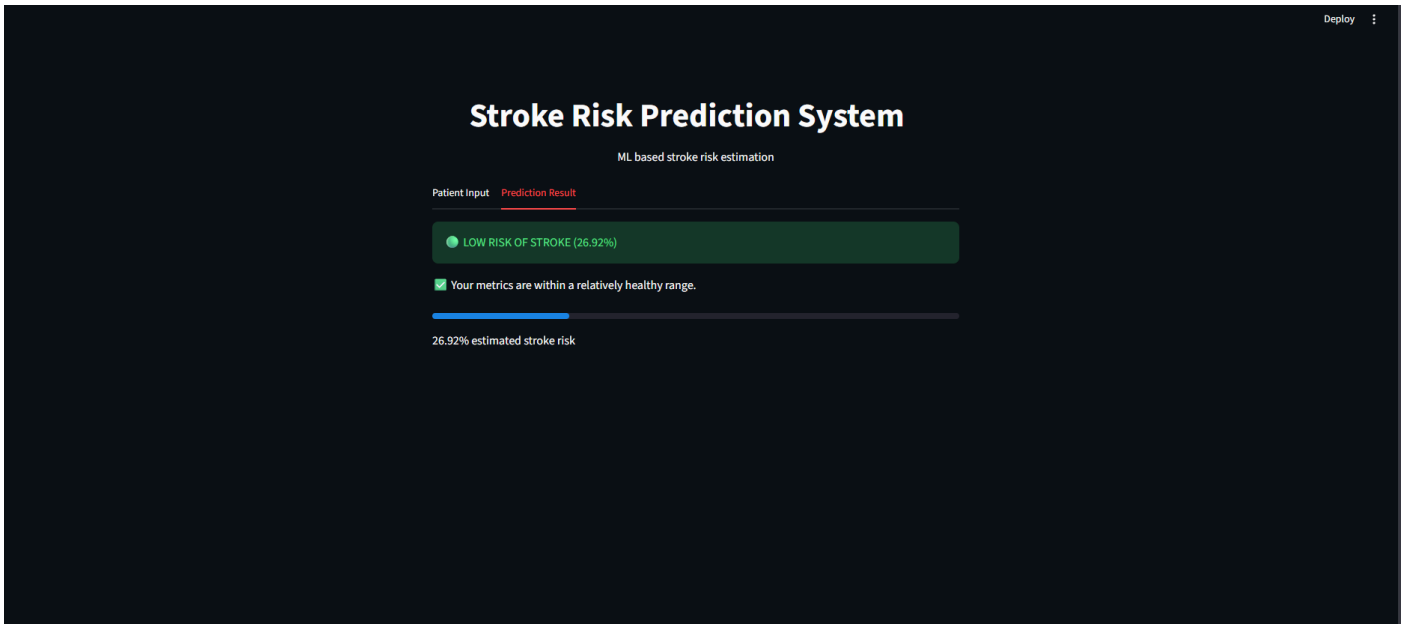
Output

Example 1:

Details Input Tab:

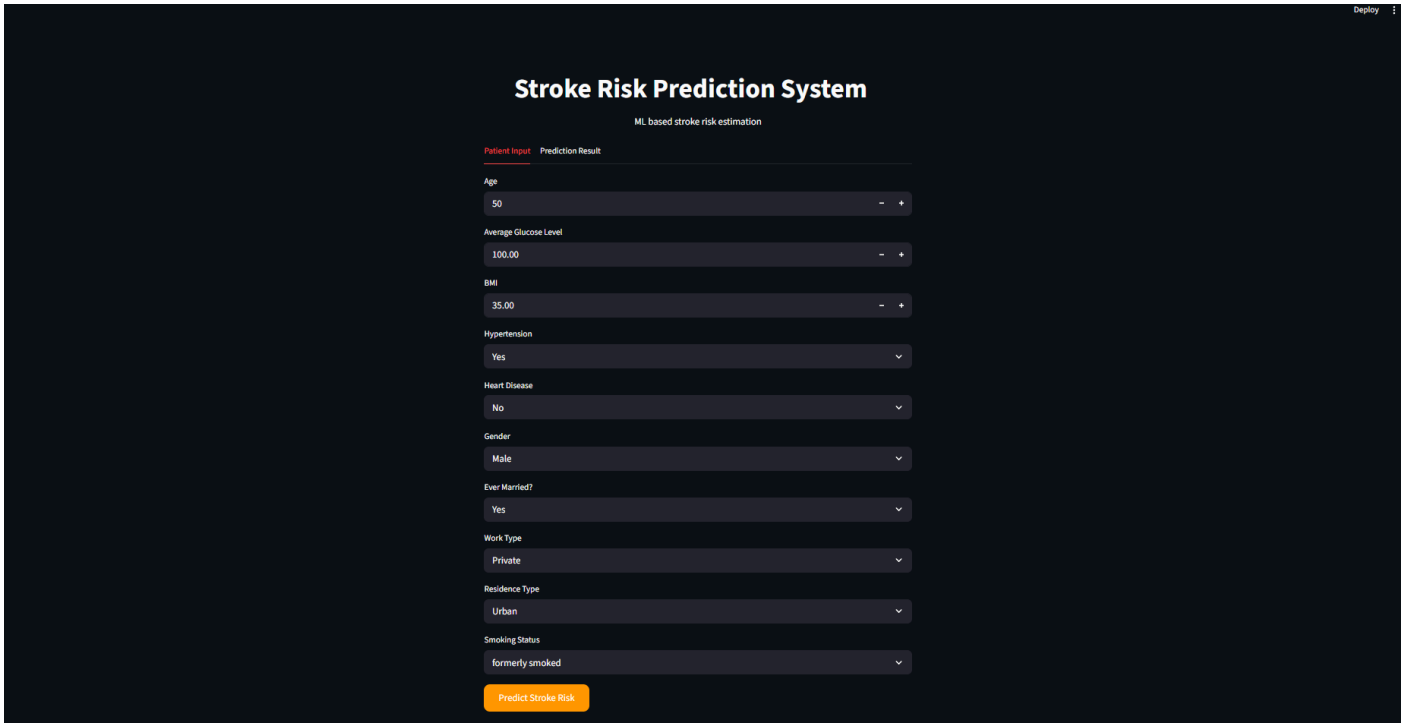


Prediction Tab:

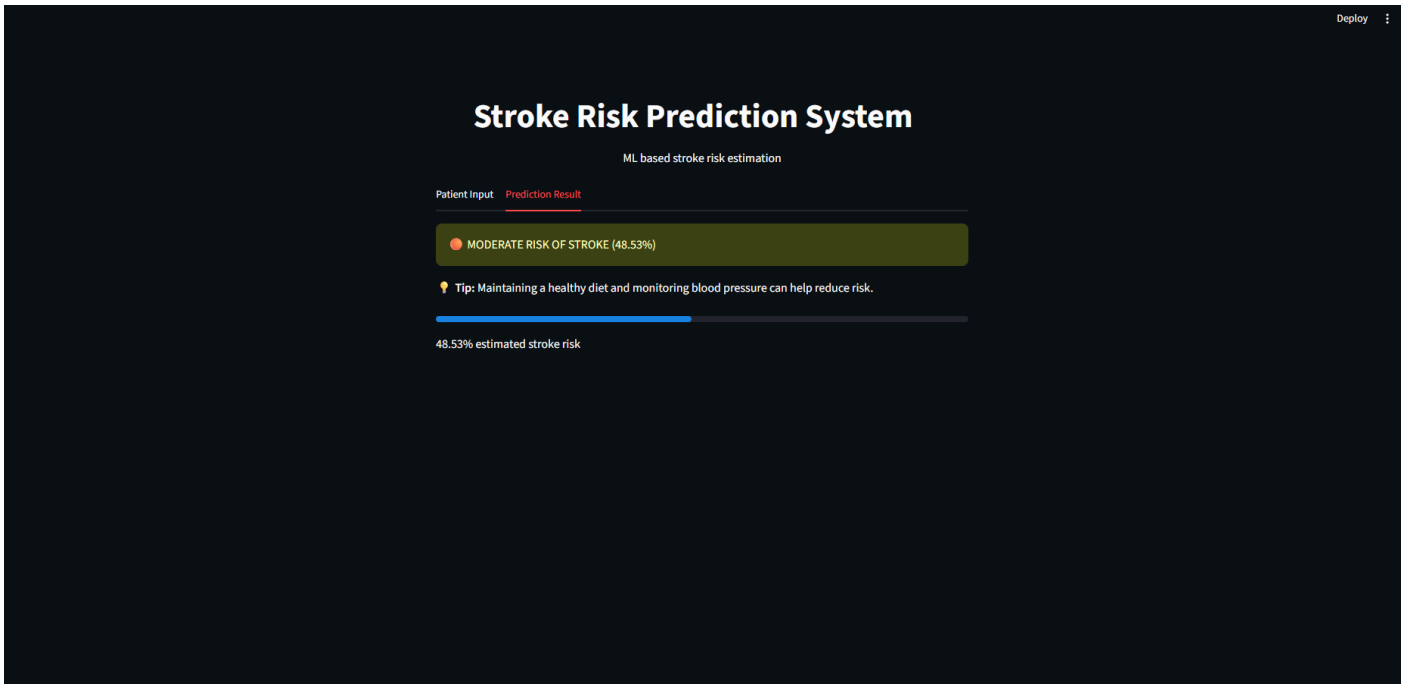


Example 2:

Details Input Tab:

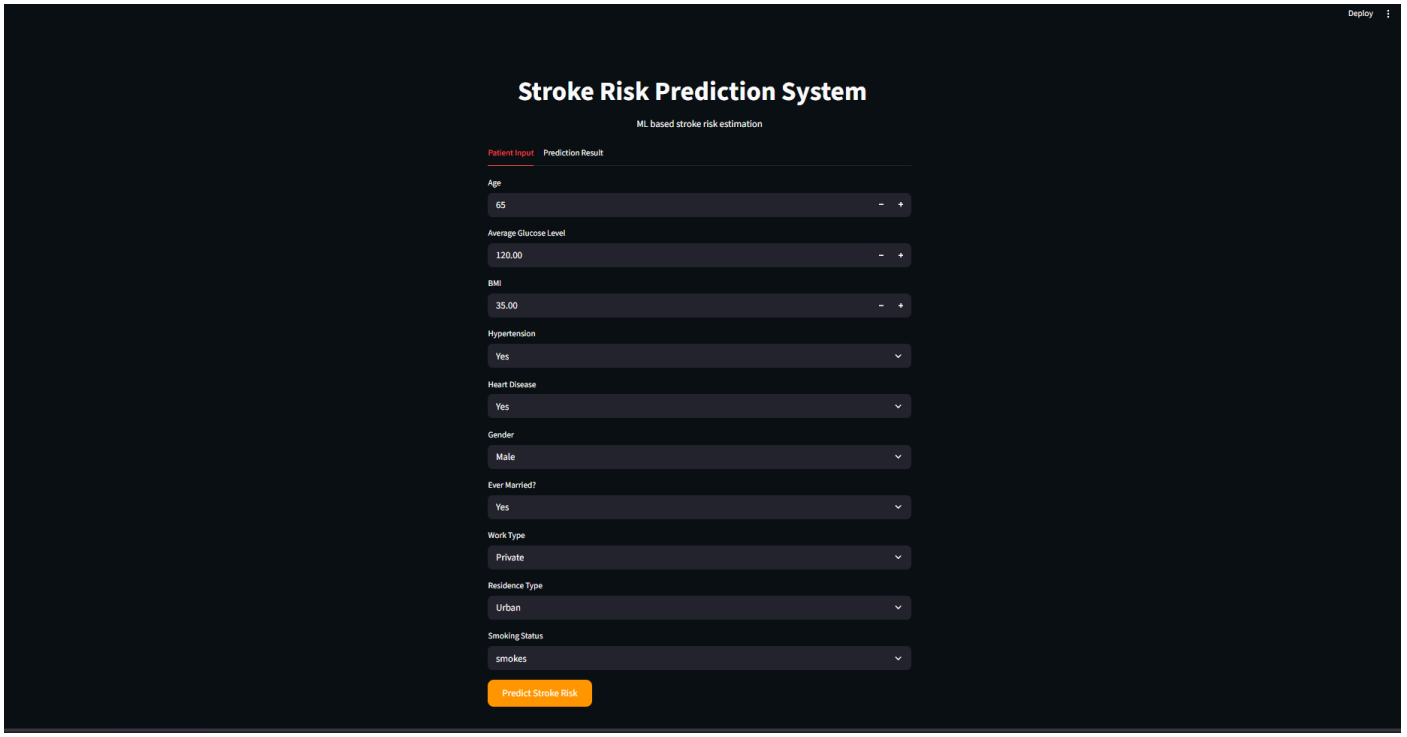


Prediction Tab:

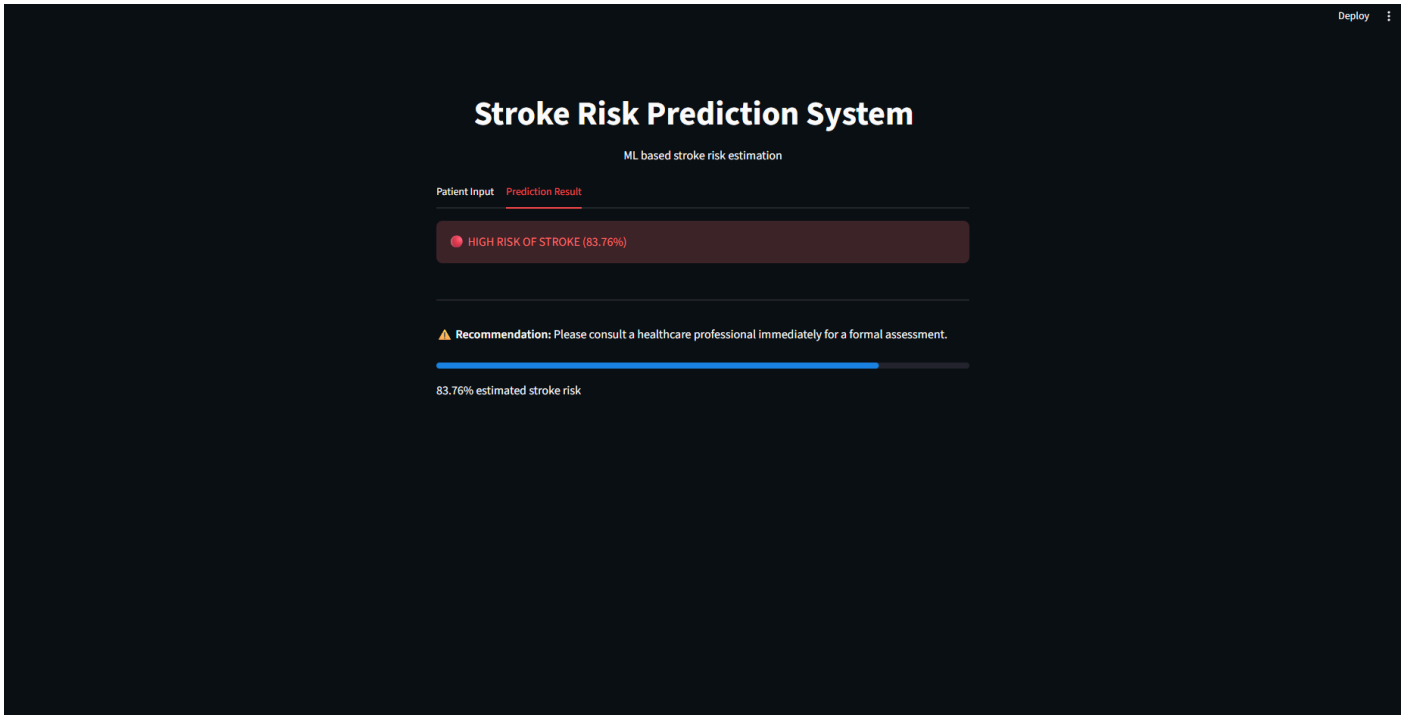


Example 3:

Details Input Tab:



Prediction Tab:



Future Scope of Improvements

Although the current Heart Stroke Prediction System demonstrates strong performance and clinical relevance, several enhancements can be explored to further improve its accuracy, usability, and real-world applicability.

1. Advanced Handling of Class Imbalance

The dataset is highly imbalanced, with stroke cases being rare. Future work can incorporate more sophisticated resampling techniques to further improve recall without excessively increasing false positives.

2. Hyperparameter Optimization

In this project, models were trained using manually selected or standard hyperparameters. Future improvements may include automated hyperparameter tuning techniques such as **Grid Search**, **Random Search**, or **Bayesian Optimization** to achieve optimal model configurations and improved predictive performance.

3. Inclusion of Additional Clinical Features

The predictive power of the system can be enhanced by integrating additional medical attributes such as:

- Blood pressure readings over time
- Cholesterol levels
- Family history of stroke
- Physical activity indicators
  These features could provide a more holistic assessment of stroke risk.

4. Use of Advanced Models

Future versions of the system may explore more advanced machine learning and deep learning models, such as:

- **XGBoost / LightGBM** for improved handling of tabular data
- **Neural Networks** for capturing complex non-linear relationships
- **Hybrid ensemble models** combining statistical and tree-based approaches

5. Threshold Optimization Based on Clinical Risk

Instead of using fixed probability thresholds, dynamic threshold optimization based on **clinical risk tolerance** can be implemented. This would allow healthcare professionals to adjust sensitivity levels depending on patient demographics or hospital policies.

6. Model Explainability and Interpretability

To increase trust and adoption in healthcare environments, future work can integrate explainability tools such as **SHAP** or **LIME**. These methods can provide patient-level explanations, helping clinicians understand *why* a particular prediction was made.

7. Real-Time and Longitudinal Data Integration

The current system is based on static historical data. Future enhancements may include:

- Integration with **real-time electronic health records (EHRs)**
- Analysis of **longitudinal patient data** to monitor risk progression over time

8. Deployment and Clinical Integration

While the project currently uses a Streamlit-based interface, future development could involve:

- Deployment as a **web-based or cloud-hosted application**
- Integration with hospital information systems
- Role-based dashboards for doctors, nurses, and administrators

9. External Validation on Diverse Datasets

To improve generalizability, the model should be validated on datasets from different regions, age groups, and healthcare systems. This would help ensure robustness and reduce demographic bias.

CERTIFICATE

This is to certify that Mr. Pritam Roy of Asansol Engineering College, Registration No. -231080110608, has successfully completed a project on Heart Stroke Prediction System Status using Machine Learning with Python under the guidance of Prof. Arnab Chakraborty.

Prof. Arnab Chakraborty

CERTIFICATE

This is to certify that
Mr. Supritam Mukhopadhyay
of Asansol Engineering College,
Registration No. -231080110263, has successfully completed a project on Heart Stroke Prediction System Status using Machine Learning with Python under the guidance of Prof. Arnab Chakraborty.

Prof. Arnab Chakraborty

CERTIFICATE

This is to certify that
Mr. Rhitinkar Bhowmick
Registration No. -231080110463
of Asansol Engineering College,
R, has successfully completed a project on
Heart Stroke Prediction System using
Machine Learning with Python under the
guidance of Prof. Arnab Chakraborty.

Prof. Arnab Chakraborty

CERTIFICATE

This is to certify that Mr. Srikanta Maji of Asansol Engineering College, Registration No. - 231080110250 , has successfully completed a project on Heart Stroke Prediction System using Machine Learning with Python under the guidance of Prof. Arnab Chakraborty.

Prof. Arnab Chakraborty