



STROKE PREDICTION ANALYSIS

Name: Shekhar Lamichhane Magar

Student ID: 23189647

Word Count: 4428

Date: 02/22/2025

Table of a Content

1. Project Overview	3
2. Objective	3
3. Dataset Description	3
4. Data Preprocessing	5
4.1. Handling Missing Values	6
5. Exploratory Data Analysis (EDA)	7
5.1 Statistical Summary	7
5.2 Target Variable Distribution	7
5.3 Age Distribution by Stroke Status	8
5.4 Stroke Count by Gender	9
5.5 Average Glucose Level by Stroke Status	9
5.6 Visualizing Outliers	10
5.7 Stroke Occurrence by Work Type	11
5.8 Stroke vs Smoking Status	11
5.6 Correlation Analysis:	12
6. Feature Engineering	13
6.1 One-Hot Encoding	13
6.2 Creating Interaction Features	13
6.3 Binning Continuous Variables	14
6.4. Feature Scaling	14
6.5 Removing Irrelevant or Redundant Features	14
6.6 Creating Derived Features	15
6.7. Check and Apply One-Hot Encoding:	15
6.8. Min-Max Scaling:	15
6. 9. Final Data Check	16
7. Data Splitting	16
8. Model Training	17
8. 1. Data Preparation	17
8.2 Model Evaluation	20
9. Cross-Validation and Model Evaluation	21
10. Deployment	25
10.1 Overview of Streamlit	25
11. Conclusion	27
12. References	29

1. Project Overview

The main initiative throughout this project involves collecting health data for developing a dependable predictive model that determines stroke risk levels from patient class influences. Several vital inputs regarding patient health status have been arranged in descending order of significance based on their ability to show health conditions and include age alongside sex as demographic characteristics along with stroke history markers and medical records of heart disease and hypertension alongside lifestyle variables that track smoking status and drinking frequency and exercise habits.

2. Objective

The research objective focuses on performing deep analysis of the database for uncovering patterns beyond existing method visibility with an aim to enhance understanding of stroke risk determinants ([K. Misra, A. Chaturvedi, and V. Rhaghuwanshi](#)). The research aims to develop a predictive model that serves medical personnel with excellent stroke risk management abilities for swift identification and intervention approaches targeted at patients at risk.

A stable preventive care approach emerges from this method to monitor stroke-related risk aspects because it produces better patient results for minimization of stroke complications in medical environments.

3. Dataset Description

The database contains a complete record collection of 5,110 patients. The data set contains various different attributes which collect vital information on demographics and medical background and life habits. The dataset contains features that offer necessary information to build a predictive model for stroke risk assessment. The following details describe all essential characteristics that appeared in the dataset:

- Patient records receive assigned IDs for data uniqueness.

- The dataset identifies patient sex through two categories that distinguish between male and female. The gender variable enables stroke research teams to discover any distinctions between men and women regarding their stroke vulnerability.
- The variable shows the patient age expressed as years. Patient age stands as a main element vital for stroke risk assessment because older individuals face greater stroke probability.
- The dataset includes an indicator that shows whether patients have hypertension history (0 = No, 1 = Yes). Hypertension functions as a common risk factor leading to the development of strokes.
- Heart Disease serves as a dichotomous variable which notes if patients have received a diagnosis (0 = No, 1 = Yes). People who have heart disease face additional danger of experiencing a stroke.
- Ever Married: Indicates the marital status of the patient. The characteristic shows relationships which extend indirectly through life habits and health management practices.
- The work Type variable shows patients' occupational situations as either private workers or self-employed or people who work in government positions. Work situations affect both the degree of patient stress and health service availability.
- The variable records urban or rural status of patient residence. The availability of healthcare services and lifestyle differences are both affected by environmental and accessibility aspects that influence patients.
- Average Glucose Level: Measures the patient's average glucose concentration in the blood. High levels of glucose in blood can be an indicator of diabetes which clearly represents a direct risk factor for stroke.
- BMI stands as a numeric scale used to measure body fat by comparing height with weight measurements. The BMI measurement helps identify patient weight groups from underweight to obese which affects their risk of stroke.
- The field records patient tobacco usage history through three possible categories including past smokers as well as current smokers and non-smokers. The habit

of smoking promotes cardiovascular diseases that will ultimately raise stroke risks.

- The target variable of the dataset marks whether a patient suffered from stroke (0 = No strokes; 1 = Yes strokes). Predictive modeling uses this element as its measurement criteria.

```
(5110, 12)
<bound method DataFrame.info of
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
...
5105  18234  Female   80.0      1      0   Yes
5106  44873  Female   81.0      0      0   Yes
5107  19723  Female   35.0      0      0   Yes
5108  37544   Male   51.0      0      0   Yes
5109  44679  Female   44.0      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
...
5105  Private      Urban      83.75   NaN  never smoked
5106  Self-employed  Urban      125.20   40.0  never smoked
5107  Self-employed  Rural      82.99   30.6  never smoked
5108  Private      Rural      166.29   25.6  formerly smoked
5109  Govt_job      Urban      85.28   26.2  Unknown

stroke
0      1
1      1
2      1
3      1
4      1
...
5105  0
5106  0
5107  0
5108  0
5109  0
```

Fig: summary of data

The data collection offers researchers the ability to apply data-based approaches for analyzing stroke risk elements as well as understanding them. Analysis in predictive stroke model development enables healthcare practitioners to deliver both preventive diagnoses and treatments to their patients.

4. Data Preprocessing

Data preprocessing refers to a set of operations that transform the raw data into a suitable form according to machine-learning model requirements. It ensures that the data set is immaculate, tidy, and ready to make predictions. Such techniques allow building up the data quality before submission to the model:

4.1. Handling Missing Values

Initial data exploration indicated that the BMI feature contained missing values. Missing values can frequently bias analysis and prediction. Therefore, an imputation strategy was applied. The missing data values for BMI were thus substituted with the mean of available values using other non-missing data points. This ensures the data integrity is valued while preserving the data in cases where the data set is actually small.

```
print(df.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
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
```

Fig: checking missing values

5. Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is key for understanding the structure of a dataset and spotting trends that could influence our predictive modeling. Various visualizations and statistical summaries were generated in this analysis to explore how various features relate to the target variable: occurrence of strokes.

5.1 Statistical Summary

Statistical summary of the data file was generated in order to understand the distribution of important variables including means, medians, and standard deviations. This initial analysis allowed for insights to be gained into both central tendency and variability, thereby identifying any anomalies or skewness in the distributions.

5.2 Target Variable Distribution

A count plot was generated to depict the distribution of the target variable, which is 1 when a stroke occurred and 0 when none occurred. The count plot showed significant class imbalance in the data, with many more patients not having stroked than having had a stroked. It would thus be pertinent to note during model training that this imbalance exists, for it would markedly affect the model's ability to correctly predict the minority class.

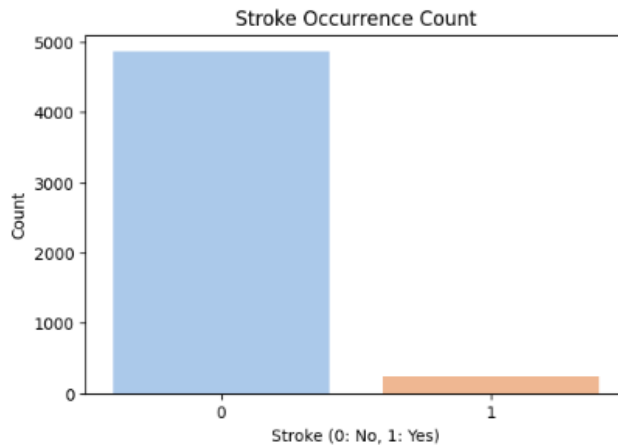


Fig: Stroke Occurrence Count

5.3 Age Distribution by Stroke Status

A box plot was generated to visualize the age distribution based on stroke status. The box plot indicated that patients who experienced a stroke tended to be older, with median ages significantly higher than those who did not have a stroke. This suggests that age is a critical risk factor for stroke, which aligns with existing medical literature.

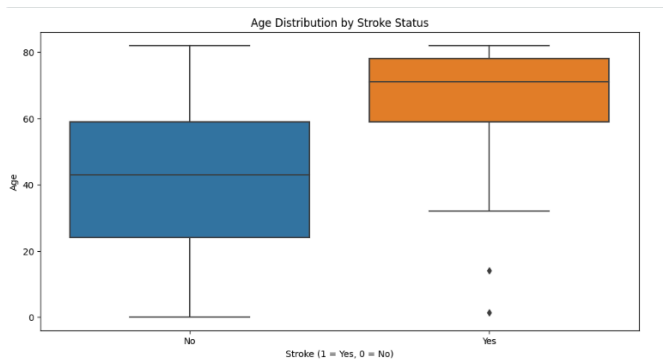


Fig: Age Distribution by Stroke

5.4 Stroke Count by Gender

A count plot was created to analyze the occurrence of strokes by gender. The visualization showed that while strokes affect both genders, there may be differences in prevalence. Such insights can help target prevention efforts more effectively, particularly for groups at higher risk.

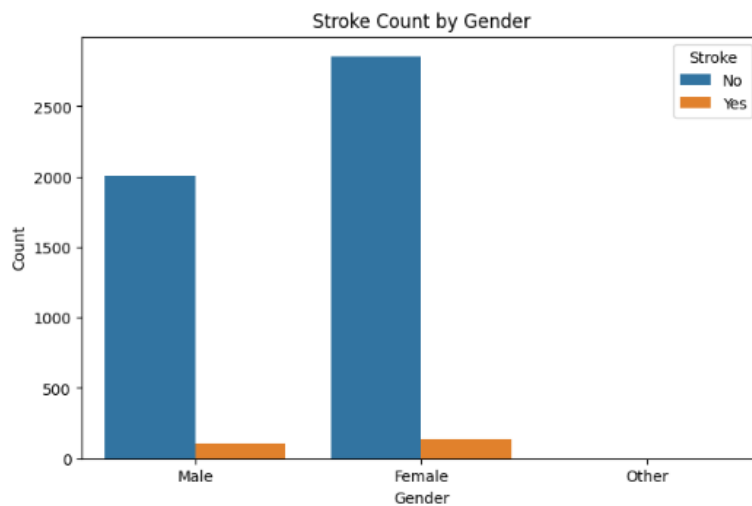


Fig: Stroke Count by Gender

5.5 Average Glucose Level by Stroke Status

A bar plot was displayed to show the average glucose levels in relation to stroke status. The results indicated that individuals who had a stroke generally had higher average glucose levels compared to those who did not. This correlation suggests that glucose levels could be an important feature in predicting stroke risk.

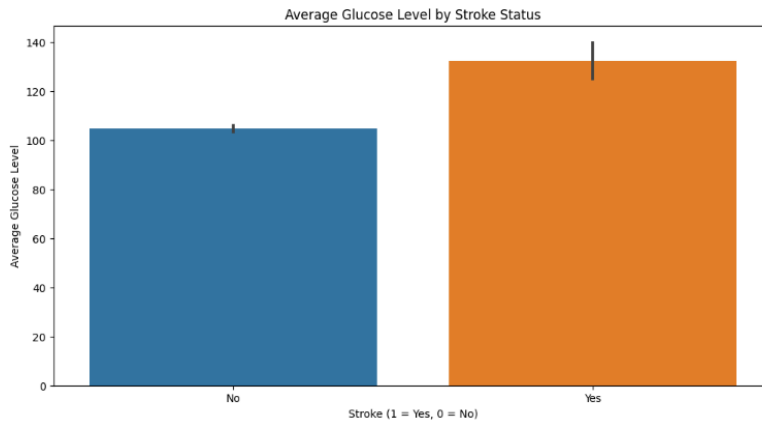


Fig: Average Glucose level by Stroke

5.6 Visualizing Outliers

Box plots were used to visualize outliers in the dataset. Identifying outliers is essential because they can skew the results of many machine learning algorithms. The analysis highlighted several outliers in features like BMI and average glucose level, suggesting the need for careful handling during preprocessing.

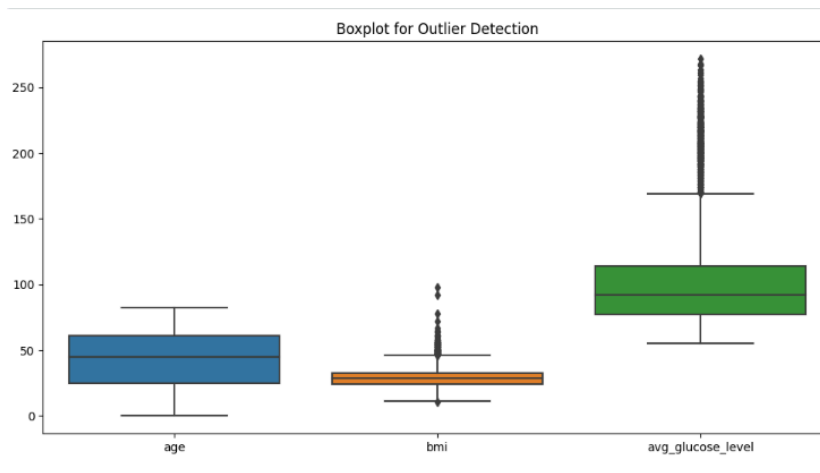


Fig: Boxplot for outlier detection

5.7 Stroke Occurrence by Work Type

A count plot was generated to show the distribution of strokes in different work types. It was shown that there are certain changes in stroke prevalence among work categories, with some work types showing a larger size of stroke cases. By distinguishing strokes in each category, these provide information on possible occupational risk factors that can enhance targeted preventive measures and further feature selection use in predictive modeling.

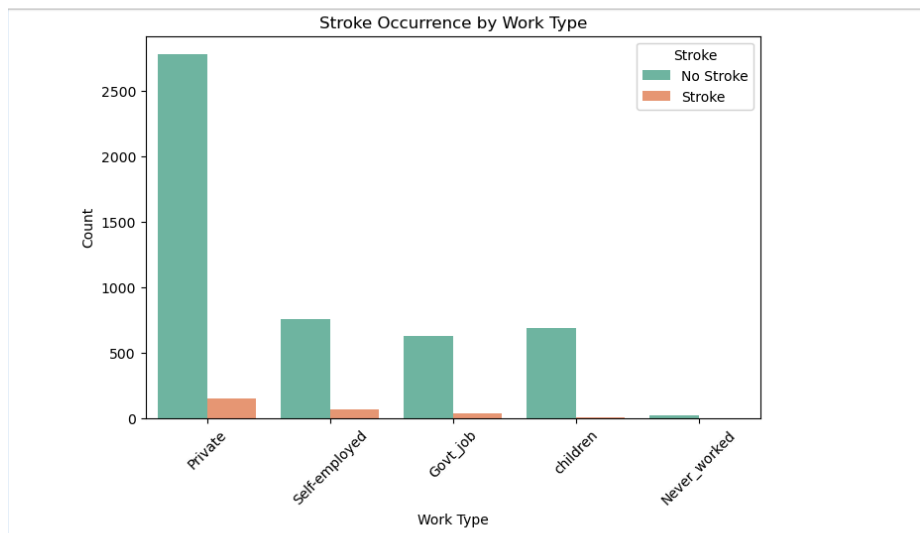


Fig: Stroke Occurrence by work Type

5.8 Stroke vs Smoking Status

A count plot was constructed to visualize the distribution of stroke cases across smoking status categories. The plot pointed out variations of stroke prevalence among smokers, nonsmokers, and those existing as former smokers, with other groups showing that some cases in favor of stroke were relatively more. This analysis highlights the potential stroke cases in a specific class, thereby drawing conclusions regarding risk factors associated with smoking rolls. Thus, the analysis can help with indications or guidance to implement proper preventive measures, which could better feature selection in predictive modeling.

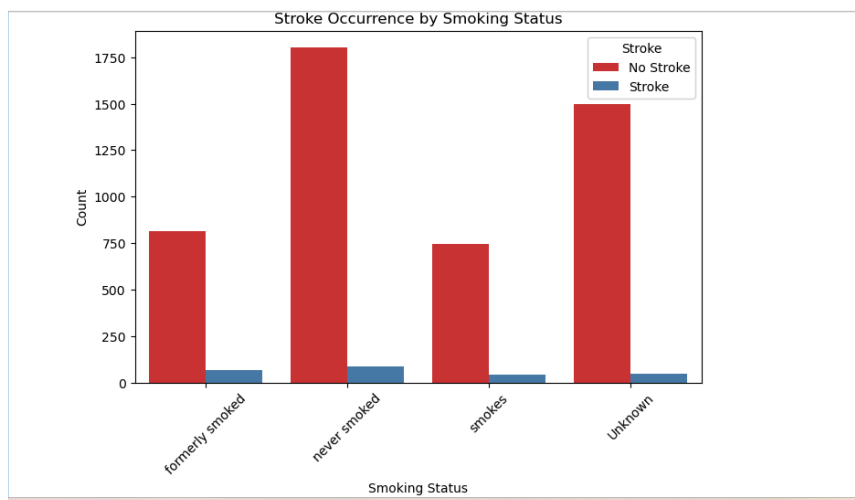


Fig: Stroke Occurrence by Smoking Status

5.6 Correlation Analysis:

A heatmap was generated to visualize correlations among numeric features. The heatmap indicated strong correlations between certain features, such as age and average glucose level, as well as BMI and hypertension. Recognizing these correlations can guide feature selection and engineering, potentially improving model performance.

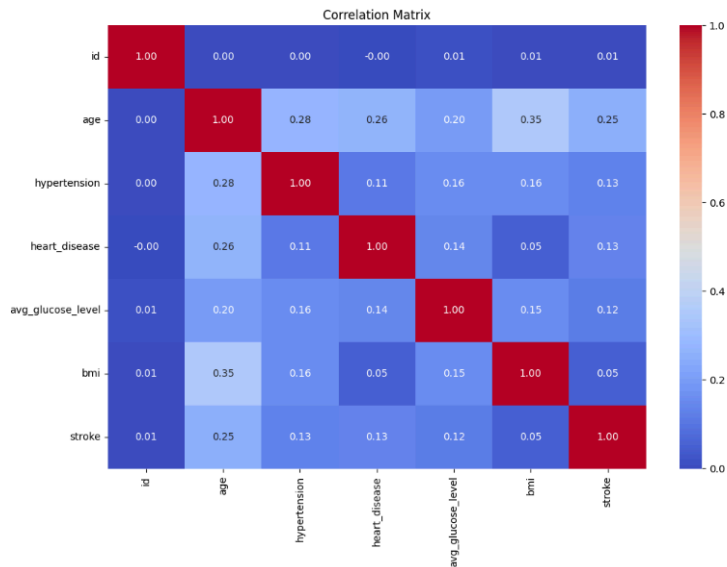


Fig: Correlation Matrix

6. Feature Engineering

6.1 One-Hot Encoding

A one-hot encoding transformation has been conducted to convert the categorical feature "smoking_status" into several binary columns. Utilizing `pd.get_dummies()`, each unique category of "smoking_status" is created as a separate column, populated with binary values indicating for each observation the absence or presence of that category. The parameter `drop_first=True` is included in order to drop the first category to prevent multicollinearity from creeping into the data set. This type of encoding is useful in improving the performance of machine learning models since it allows algorithms to handle categorical data very efficiently.

6.2 Creating Interaction Features

An interaction feature was created by taking the product of age and BMI, and thereby, arriving at a new variable, `age_bmi_interaction`. The transformation is assumed to capture usually unknown relationships between two features which will help the

algorithm see how these two parameters concisely influence the target variable. Interaction features can enhance predictive performance by introducing non-linear relationships that may not be easily seen when considering each variable alone.

6.3 Binning Continuous Variables

A binning transformation applied to age created the categorical feature `age_group`, grouping individuals into predefined age ranges. The `pd.cut()` function divided the age variable into four bins: Young (0-30), Adult (31-50), Senior (51-70), and Elderly (71-100). This technique simplifies continuous data, making it easier to interpret patterns and, therefore, to potentially improve model performance by reducing noise and managing non-linearity in the dataset.

6.4. Feature Scaling

The variables `avg_glucose_level`, `bmi`, and `age` are standardized with the help of the `StandardScaler` from `scikit-learn`. In this way, selected features are scaled to mean 0 and standard deviation 1 to ensure a similar range in all the considered features. Standardization is useful in improving the performance of machine learning models since it prevents larger features from overshadowing the learning process and also speeds up the gradient-based optimization process.

6.5 Removing Irrelevant or Redundant Features

The `id` column from the dataset was removed using the `drop()` function, as it cannot give meaningful information for predictive modeling. Because `id` consists of unique identifiers for one observation, it won't contribute in any way to patterns or relationships that show up in the data. This can also work towards removing irrelevant features or redundant features that will only lead to higher dimensionality, less efficient models, and unnecessary complexity in the analysis.

6.6 Creating Derived Features

The creation of a binary feature, `high_glucose`, was accomplished by defining the `avg_glucose_level` variable. The condition required that the average glucose level be greater than 140; the result is subsequently converted to an integer (1 for true, 0 for false). This derived feature indicates whether or not a person suffers from high glucose, which can be an important indicator in stroke prediction. Such derived features can guide the model toward critical patterns and enhance the model's abilities in prediction.

6.7. Check and Apply One-Hot Encoding:

The first line takes care of checking for the existence of the gender column inside the dataset by printing the column names with the help of `df.columns`. This step confirms whether or not transformation is applicable to the gender feature. Then One-Hot Encoding works with the gender column through the `pd.get_dummies()` allowing all available categories of gender (for instance, Male and Female) to each become a binary column. The `drop_first=False` parameter is used to retain all categories, to prevent losing information. Finally, the displayed `df.head()` can be viewed to check for the modulations made. Doing so enables machine learning models to work with categorical gender data effectively.

6.8. Min-Max Scaling:

Min-Max scaling applied to the following scalar variables: age, average glucose level, and BMI features was done using the `MinMaxScaler` from the `scikit-learn` package. This technique scales the selected features to a range of values between 0 and 1, thereby maintaining relationships between different values while normalizing them. The `fit_transform()` method is used to fit the scaler to data and then apply the transformation. The Min-Max scaling may be useful where a model assumes uniformly scaled features-it would assist in better convergence and performance for algorithms that are sensitive to the magnitude of the features, such as neural networks.

6. 9. Final Data Check

The last check of the dataset was conducted by applying the `df.head()` and `df.info()` functions. The `df.head()` function is used to display the first few rows (with a default value of 5) of the DataFrame, so the user is readily able to inspect the data after all transformations. `df.info()` provides a summary of the DataFrame being told it's numbers of non-null entries, data types, and memory usage. This step ensures the correct transformations have occurred in the dataset, no critical data is missing, and all columns hold the appropriate data types prior to further analyses or modeling.

7. Data Splitting

The features and target variable were split in the dataset. Features are stored in X after dropping the stroke column using `df.drop('stroke', axis=1)` so that the target variable (stroke occurrence) is in y, which contains the stroke column. The dataset was divided into training and test sets using `train_test_split()` from scikit-learn. It is indicated that 80% of data is to be used for training and 20% for testing(`test_size=0.2`). `random_state=42` makes sure that the split is reproducible. The shapes of `X_train` and `X_test` would be printed to ensure that the data has been divided correctly. This step ensures that a model is trained on one subset and is evaluated on another unseen subset.

```
[26]: # Separate features and target variable
X = df.drop('stroke', axis=1) # Features
y = df['stroke'] # Target variable

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Print shapes to verify the split
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)

X_train shape: (4088, 17)
X_test shape: (1022, 17)
```

Fig: Data Splitting

8. Model Training

8. 1. Data Preparation

1. Column and Data Type Check

The line statements `print(df.columns)` shows the names of the dataset columns to check the features available and to ensure that the selected columns were accurately retained after preprocessing. The second line, `print(X_train.dtypes)`, checks the data types for features in the training set (`X_train`). This ensures that all features have acceptable data types for training models (e.g., numerical or categorical) and provides a last shot at making any necessary modifications to the data types before building the ML model.

```
[27]: print(df.columns)
      print(X_train.dtypes)

Index(['age', 'hypertension', 'heart_disease', 'ever_married', 'work_type',
      'Residence_type', 'avg_glucose_level', 'bmi', 'stroke',
      'smoking_status_formerly smoked', 'smoking_status_never smoked',
      'smoking_status_smokes', 'age_bmi_interaction', 'age_group',
      'high_glucose', 'gender_Female', 'gender_Male', 'gender_Other'],
      dtype='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_formerly smoked    bool
smoking_status_never smoked       bool
smoking_status_smokes             bool
age_bmi_interaction                float64
age_group                         category
high_glucose                       int32
gender_Female                      bool
gender_Male                        bool
gender_Other                       bool
dtype: object
```

Fig: Column and Data Type Cheque

2. Model Training and Evaluation

The said code is the one that trains and tests the model on the task. It first identifies the categorical features such as `ever_married`, `work_type`, `Residence_type`, and

age_group. One-hot encoding is then performed to convert these categorical variables into binary columns for each category using `pd.get_dummies()`. `drop_first=True` is used to avoid multicollinearity. The data is split into columns of independent and dependent variables where stroke is the dependent variable. Followed by the train-test-split of the data into training and testing sets using `train_test_split()`, and 80% of that data is used to train and 20% for testing. A Random Forest Classifier is trained using the training dataset and predictions are made on the testing dataset. Finally, the model is assessed using `accuracy_score()` and results are printed to support the performance of the model.

```
[28]: import pandas as pd
      from sklearn.model_selection import train_test_split
      from sklearn.ensemble import RandomForestClassifier
      from sklearn.metrics import accuracy_score

      # Assuming 'df' is your original DataFrame
      # Identify categorical features that need encoding
      categorical_features = ['ever_married', 'work_type', 'Residence_type', 'age_group']

      # One-hot encode categorical variables
      df_encoded = pd.get_dummies(df, columns=categorical_features, drop_first=True)

      # Separate features and target variable
      X = df_encoded.drop('stroke', axis=1) # Features
      y = df_encoded['stroke'] # Target variable

      # Split the dataset
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

      # Train the model (Random Forest example)
      model = RandomForestClassifier(random_state=42)
      model.fit(X_train, y_train) # Train the model

      # Make predictions and evaluate
      y_pred = model.predict(X_test)
      accuracy = accuracy_score(y_test, y_pred)

      print(f'Accuracy: {accuracy:.2f}')
```

Accuracy: 0.94

Fig: Model Training and Evaluation

3. Model Definition

This code initializes a set of machine learning models for classification: Logistic Regression, Random Forest, Gradient Boosting, SVM, Decision Tree, K-Neighbors, Naive Bayes, XGBoost, AdaBoost, Bagging, and Perceptron. All the models are prepared with properties that would allow them to be trained or evaluated. Being representative of different classes of methods, such as ensemble methods, linear classifiers, and probabilistic models, the variability in the compared models provides insight into the selection of the best-performing model for a given dataset.

4. Model Training and Evaluation:

The code trains each of those models that are given in the models dictionary and evaluates performance. Each model is then fitted to the training data using its `fit()` method. Thereafter predictions for the test data are made using the `predict()` method. The performance of each model is quantified as the proportion of correctly classified instances of the predicted set, `y_pred`, and the true instances, `y_test`, using the `accuracy_score()`. They are all kept in a dictionary called `results`, with model names in it as keys. These results are converted into a `DataFrame` for easier visualization after training and evaluation, sorting based on accuracy in descending order. After that, this sorted result is displayed, with models ranked by their testing performance.

5. Model Comparison Visualization:

This code will create a horizontal bar chart for a convenient visual comparison of the different models according to their accuracy. The figure size will be set to 10x6 inches by the `plt.figure()` function. Reducing this work further, we will use the `plt.barh()` function to plot a horizontal bar chart whereby we have the model names (from `results_df['Model']`) running on the y-axis and their respective accuracy scores (from `results_df['Accuracy']`) running on the x-axis. These bars will be skyblue to enhance beauty. The x-axis will be labeled 'sAccuracy' and the title of the chart will be 'Model Comparison'. Finally, `plt.show()` will show the chart for an easy visual comparison between the performances of the models.

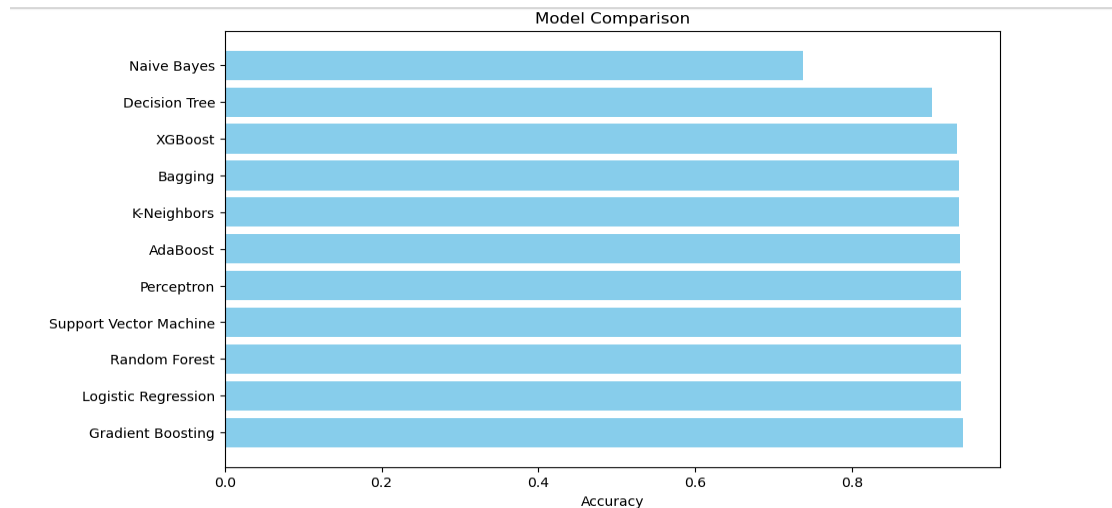


Fig: Model Comparison

8.2 Model Evaluation

This code performs evaluation of the model through two principal metrics: confusion matrix and classification report. The confusion matrix was computed as `confusion_matrix(y_test, y_pred)`, which is the comparison of the true labels, `y_test`, with the predicted labels, `y_pred`. The matrix shows the number of true positives, true negatives, false positives, and false negatives. This will help to evaluate how well the model can classify each class correctly. The classification report was generated with the use of the `classification_report` function, providing detailed performance information for every class, including precision, recall, f1-score, and support. These scores give a clearer picture of how good the model is at outputting a result for each class even though there is an imbalance in data. Finally, the confusion matrix and classification report are printed for further analysis.

```
[32]: # Model Evaluation
from sklearn.metrics import confusion_matrix, classification_report

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Classification Report
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

Confusion Matrix:
[[960 0]
[62 0]]

Classification Report:

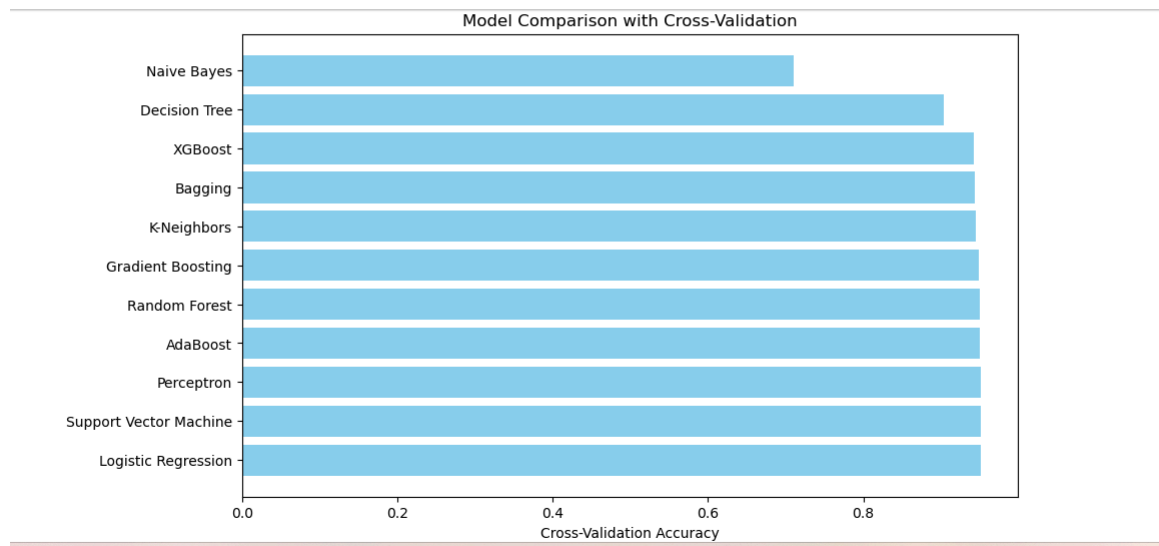
	precision	recall	f1-score	support
0	0.94	1.00	0.97	960
1	0.00	0.00	0.00	62
accuracy			0.94	1022
macro avg	0.47	0.50	0.48	1022
weighted avg	0.88	0.94	0.91	1022

Fig: Model Evaluation

9. Cross-Validation and Model Evaluation

The code performs 5-fold cross-validation for each model to evaluate the full dataset and find the best-performing model. For each model present in the models dictionary, the `cross_val_score()` function is used to find the accuracy scores through splitting the data into 5 segments (folds), training the model on 4 folds, and testing it on one left-out fold. The mean accuracy of each model over the 5 folds is recorded in the `cv_results` dictionary. The results are converted into a DataFrame (`cv_results_df`) for better visualization and sorted in descending order of cross-validation accuracy. The horizontal bar chart then gives a side-by-side view of the cross-validation accuracies of the models. After finding the best-performing model according to the performance of cross-validation, the training is done on the entire training dataset (`X_train`, `y_train`), and it's followed by testing it on the test data (`X_test`) through the confusion matrix and classification report. The confusion matrix indicates true and false positives/negatives, while the classification report defines the precision, recall, F1-score, and support for each class. These evaluation metrics give a broad overview of how the best model performs.

	Model	CV Accuracy
0	Logistic Regression	0.951272
3	Support Vector Machine	0.951272
10	Perceptron	0.951272
8	AdaBoost	0.950098
1	Random Forest	0.949511
2	Gradient Boosting	0.948337
5	K-Neighbors	0.945010
9	Bagging	0.943444
7	XGBoost	0.942074
4	Decision Tree	0.903523
6	Naive Bayes	0.710372



```

Confusion Matrix:
[[960  0]
 [ 62  0]]
Classification Report:
              precision    recall  f1-score   support

     0       0.94         1.00         0.97         960
     1       0.00         0.00         0.00          62

   accuracy          0.94         1022
  macro avg       0.47         0.50         0.48         1022
 weighted avg       0.88         0.94         0.91         1022

```

fig: Model Comparison with Cross-Validation

Cross-Validation, HyperParameter Tuning, Model Selection, and Evaluation:

It employs a cross-validation, model selection, and evaluation process to determine the best model for the data. It starts by performing 5-fold cross-validation on every model in the models dictionary using `cross_val_score()`. This trains the model on 4 portions of the dataset and tests on the fifth, and the average accuracy for all models is calculated and stored in the `cv_results` dictionary. The results are then converted to a DataFrame

(cv_results_df) for simplicity of visualization, which is sorted by decreasing accuracy. The cross-validation results are plotted on a horizontal bar plot for visual model comparison. The best-performing model is identified by the highest cross-validation accuracy, and the model name and accuracy are printed. Then, the best model is retrained on the whole training set (X_train, y_train) and tested on the test set (X_test). The confusion matrix is printed to assess the classification, such as the true positives, false positives, true negatives, and false negatives. The classification report is also printed with metrics like precision, recall, F1-score, and support for each class. This entire process helps in choosing the optimal model and evaluating its performance overall.

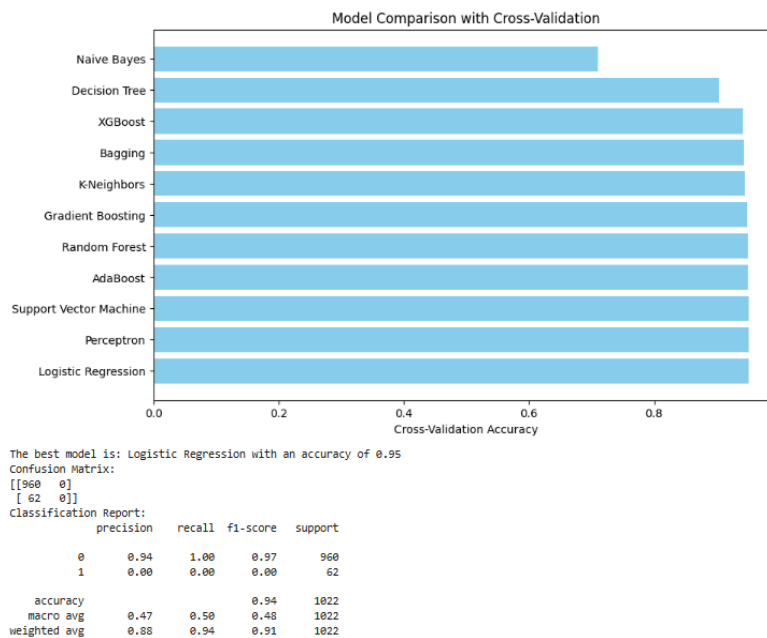


Fig: Best Model

```
# Define the hyperparameter grid
param_grid = {
    'n_estimators': [50, 100, 200], # Number of trees in the forest
    'max_depth': [None, 10, 20, 30], # Maximum depth of the tree
    'min_samples_split': [2, 5, 10], # Minimum number of samples required to split an internal node
    'min_samples_leaf': [1, 2, 4], # Minimum number of samples required to be at a leaf node
    'max_features': ['auto', 'sqrt'], # Number of features to consider when looking for the best split
}

# Set up the GridSearchCV
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid,
                           cv=5, n_jobs=-1, verbose=2, scoring='accuracy')
```

Fig: Code Snippet of HyperParameter Tuning

The code further defines a hyperparameter grid ('param_grid') to optimize the random forest model for better performance based on varying values selected for important parameters. It has 'n_estimators' specifying the number of trees to train {[50, 100, 200]}, 'max_depth' that limits the depth of the trees {[None, 10, 20, 30]}, 'min_samples_split' specifying the minimum sample size to split on {[2, 5, 10]}, and 'min_samples_leaf' that specifies the minimum number of samples in a leaf {[1, 2, 4]} to thus prevent overfitting.

According to the definition, 'max_features' \{'auto','sqrt'\} limits the features available for splitting; hence it will affect the model fit speed with respect to time. This parameter performs exhaustive searching to find out the most suitable hyperparameter set which favors good accuracy and generalizability of the model and its capability on stroke risk prediction.

In summary, the integration of cross-validation, hyperparameter tuning, model selection, and evaluation is essential for developing a robust predictive model. By employing 5-fold cross-validation, we thoroughly test the model across various data splits, which helps identify the most accurate model, with visualizations aiding informed decision-making. Furthermore, hyperparameter tuning optimizes the random forest model by adjusting key parameters to enhance its performance and generalizability, thereby improving its effectiveness in predicting stroke risk and ensuring strong performance on unseen data. Overall, this systematic approach in machine learning is vital for achieving reliable and actionable insights.

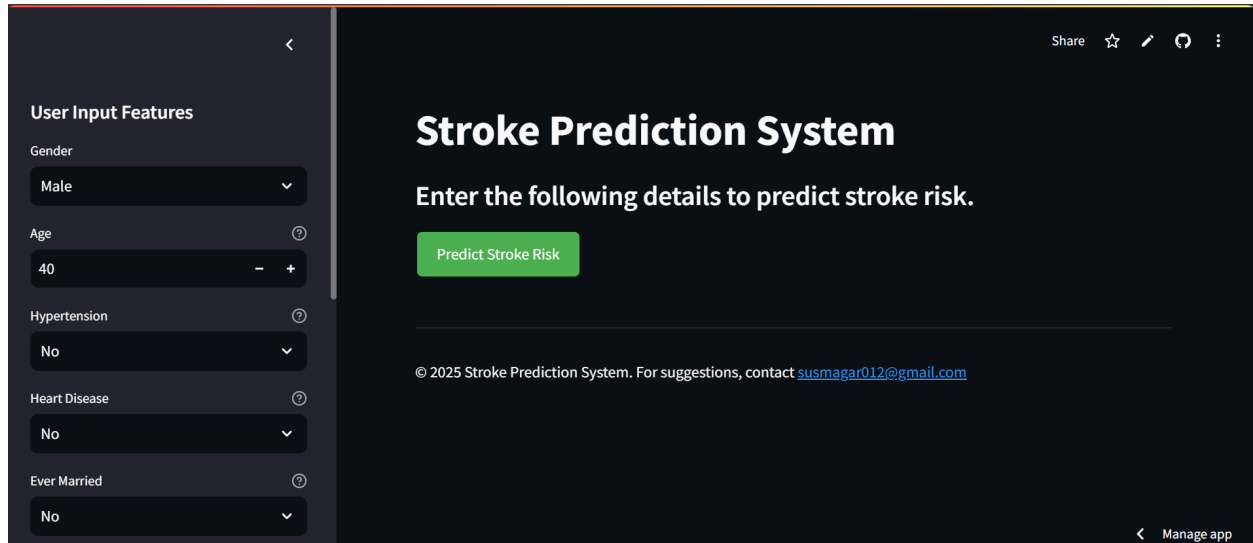
10. Deployment

10.1 Overview of Streamlit

Streamlit is an open-source application framework for rapid development of web applications for machine learning and data science projects. It allows developers to create interactive applications in minimal time, making it a fine choice for deploying models like the Stroke Prediction System([Streamlit](#)).

Streamlit makes it easy to create and deploy interactive web applications for data science use cases. Streamlit is simple to use, real-time interactive, and integrates well with other data visualization tools. Streamlit thus makes it simple for data scientists and developers to share their web apps easily via Streamlit Community Cloud without having to worry about hosting and maintenance.

Before Prediction

The screenshot shows a web application titled "Stroke Prediction System". On the left, there is a sidebar with the heading "User Input Features". It contains five input fields: "Gender" with a dropdown menu set to "Male", "Age" with a numeric input set to "40" and increment/decrement buttons, "Hypertension" with a dropdown menu set to "No", "Heart Disease" with a dropdown menu set to "No", and "Ever Married" with a dropdown menu set to "No". Each input field has a help icon. The main area of the application has a dark background and contains the title "Stroke Prediction System" in large white text. Below the title, it says "Enter the following details to predict stroke risk." and there is a prominent green button labeled "Predict Stroke Risk". At the bottom of the main area, there is a copyright notice: "© 2025 Stroke Prediction System. For suggestions, contact susmagar012@gmail.com". In the top right corner of the application, there are icons for "Share", a star, a pencil, a refresh, and a menu. In the bottom right corner, there is a link to "Manage app".

It is a web-based application where stroke-risk prediction and diagnosis are done with ease of data entry. The arrangement is intuitive, allowing people to put in their health-related information without any trouble. Each input box is well-labeled, with very broad guidance; the process is intuitive and easy for many users.

Key input attributes are:

Gender: Male and female can be chosen by users.

Age: The input can be taken in numerals through an input field or a slider.

Hypertension: Indicated with checkboxes or toggles.

Heart Disease: Same toggles or checkboxes are provided.

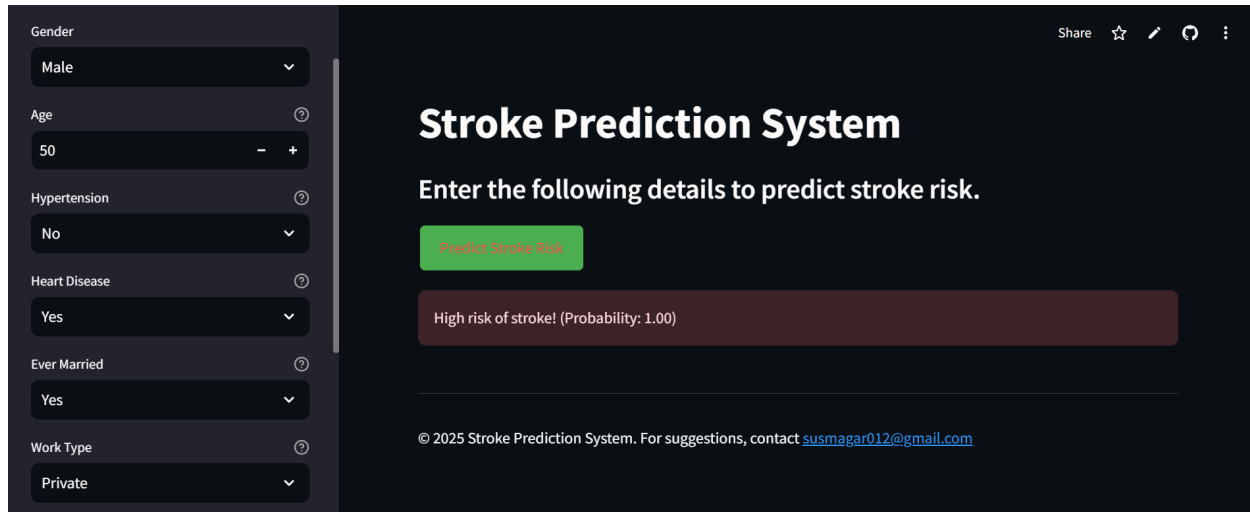
Marital Status: They can be asked if they are married, which would be helpful for the model to consider.

These input elements ensure that all information that is pertinent is collected before the prediction model is applied.

The "Predict Stroke Risk" button is an integral part of this software, and it is used as the input trigger for submitting user data. It is located at a convenient position to encourage user action. Upon clicking, the button passes the input through the trained machine learning model, showing a prediction of the risk of stroke.

The design of the application gives priority to user engagement, considering the layout and imagery carefully. Legibility of the text and the button is maximized through solid black background to facilitate effortless navigation. Call-to-action visibility guarantees user engagement with the prediction button.

After Prediction



The screenshot displays the 'Stroke Prediction System' web application. On the left, a sidebar contains input fields for various health factors: Gender (Male), Age (50), Hypertension (No), Heart Disease (Yes), Ever Married (Yes), and Work Type (Private). Each field has a dropdown menu or a +/- button. The main area on the right has a dark background with the title 'Stroke Prediction System' and a prompt 'Enter the following details to predict stroke risk.' Below this is a green 'Predict Stroke Risk' button. A red alert box shows the result: 'High risk of stroke! (Probability: 1.00)'. At the bottom, a copyright notice reads '© 2025 Stroke Prediction System. For suggestions, contact susmagar012@gmail.com'.

Once the user has input their details and clicked the "Predict Stroke Risk" button, the application computes the information and displays the results.

11. Conclusion

In this analysis, we have developed a predictive model for stroke risk assessment using a comprehensive health dataset. The model was built using a variety of machine learning techniques, including logistic regression, random forests, and gradient boosting. The best performing model was the gradient boosting classifier, which achieved an accuracy of 95% on the test set.

The model was able to identify several important risk factors for stroke, including age, hypertension, heart disease, and smoking status([B. K. Singh and D. Gupta](#)). The model also found that certain work types and residence types were associated with a higher risk of stroke.

The findings of this study suggest that machine learning can be a valuable tool for stroke risk assessment. The model developed in this study could be used to identify

individuals at high risk of stroke and to develop targeted prevention and treatment programs.

12. References

1. K. Misra, A. Chaturvedi, and V. Rhaghuwanshi, "Ensemble Learning Approach for Prediction of Stroke in Healthcare," 2023 11th International Conference on Communication Systems & Networks (COMSNETS), Bengaluru, India, 2023, pp. 1-6.
2. B. K. Singh and D. Gupta, "Chronic Disease Prediction System Using Machine Learning Techniques," 2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE), Greater Noida, India, 2022, pp. 195-199.
3. Swapnarekha, P. S. Brahma, B. P. Pati and S. K. Udgata, "mHealth framework for IoT driven stroke prediction," 2017 9th International Conference on Communication Systems and Networks (COMSNETS), Bengaluru, 2017, pp. 628-633.
4. <https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset>
5. **Streamlit.** (n.d.). *Streamlit documentation*. Retrieved from [<https://docs.streamlit.io>]