**INDEX :**

**1. ABSTRACT**

**2. System Requirements**

**-2.1 Hardware requirements**

**-2.2 Software requirements**

**-2.3 Python and Flask versions**

**3. Setup and Installation**

**- 3.1 Setting up the Python environment**

**- 3.2 Installing Flask and other dependencies**

**-3.3 Running the Flask application locally**

**4. Machine Learning Model**

**-4.1 Description of the dataset**

**-4.2 Data preprocessing steps**

**-4.3 Model selection and training**

**-4.4 Model evaluation and validation**

**5. Flask Application Structure**

**-5.1 Overview of the Flask app structure**

**-5.2 Explanation of app routes and views**

**-5.3 Integration of the machine learning model**

**6. API Endpoints**

**-6.1 Detailed description of API endpoints**

**-6.2 Request and response formats**

**-6.3 Example API calls**

**7. User Interface**

**-7.1 Designing the web interface**

**-7.2 Connecting the interface with Flask**

**-7.3 User interaction flow**

8. Conclusion

**-8.1 Summary of the project**

**-8.2 Future work and improvements**

**9. References**

**-9.1 Citing datasets, libraries, and tools used**

**-9.2 Acknowledgments**

**1. ABSTRACT**

Stroke is a significant global health concern, ranking as the second leading cause of death worldwide according to the World Health Organization (WHO), accounting for approximately 11% of total deaths. To address this issue, this project utilizes a dataset to develop a predictive model aimed at identifying patients at risk of experiencing a stroke based on various input parameters.

The dataset contains information about patients, including their gender, age, presence of hypertension and heart disease, marital status, work type, residence type, average glucose level, body mass index (BMI), smoking status, and stroke occurrence. Each row in the dataset represents a unique patient profile.

The predictive model is built using machine learning techniques, which analyze the relationship between the input parameters and the occurrence of stroke. Key features such as age, hypertension, heart disease, BMI, and smoking status are examined to determine their impact on stroke risk.

Through the deployment of this predictive model, healthcare professionals can potentially identify individuals who are at a higher risk of stroke and intervene with appropriate preventive measures or treatments. By leveraging data-driven insights, this project aims to contribute to the early detection and prevention of strokes, ultimately improving patient outcomes and reducing the burden of stroke-related morbidity and mortality.

**2. System Requirements**

**2.1 Hardware Requirements:**

**- \*\*Processor\*\*:** A modern multi-core processor (e.g., Intel Core i5 or equivalent) for efficient computation.

**- \*\*Memory (RAM)\*\*:** Minimum 8GB RAM to ensure smooth execution of machine learning algorithms and Flask application.

**- \*\*Storage\*\*:** Sufficient disk space to store the dataset, codebase, and any generated model files.

**2.2 Software Requirements:**

**Operating System\*\*: - \*\*** Any modern operating system supported by Python and Flask (e.g., Windows, macOS, Linux).

**- \*\*Python\*\*:** Version 3.6 or higher is recommended for compatibility with the required libraries and frameworks.

- \*\*Integrated Development Environment (IDE)\*\*:

**- Jupyter Notebook:** For data exploration, analysis, and model development.

**- Visual Studio Code:** For writing and managing Flask application code.

**2.3** **Python and Flask Versions**:

- \*\*Python\*\*:

- Version 3.11 or higher.

- Required Python libraries: pandas, numpy, scikit-learn, matplotlib, seaborn, Flask.

- \*\*Flask\*\*:

- Version 2.0.1 or higher for creating the web application.

- Additional Flask extensions may be used for enhanced functionality (e.g., Flask-RESTful for creating APIs).

This section outlines the hardware and software requirements for setting up the environment to run the stroke prediction application. Ensure that your system meets these requirements to effectively develop and deploy the application.

**3 Setup and Installation:**

**3.1 Setting up the Python Environment:**

**1. \*\*Python Installation\*\*:**

- Download and install Python from the official website: [Python Official Website](https://www.python.org/).

- Follow the installation instructions for your operating system.

- Ensure Python is added to the system PATH during installation.

**2. \*\*Creating a Virtual Environment (Optional)\*\*:**

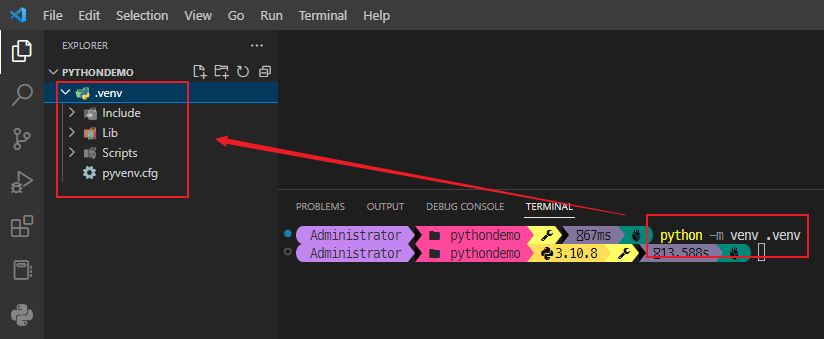
- It is recommended to create a virtual environment to isolate dependencies for the Flask application.

- Open a command prompt or terminal.

- Navigate to your project directory.

- Run the following command to create a virtual environment named `venv`:

‘’’python -m venv .venv```

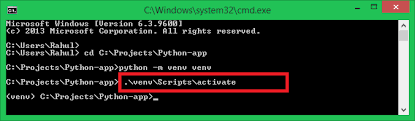


**3. \*\*Activating the Virtual Environment\*\*:**

- Activate the virtual environment by running the appropriate command based on your operating system:

- On Windows:

‘’’venv\Scripts\activate’’’



- On macOS and Linux:

```

source venv/bin/activate

```

**### Installing Flask and Other Dependencies:**

**1. \*\*Installing Required Libraries\*\*:**

- Open a command prompt or terminal.

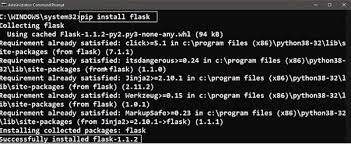
- Navigate to your project directory.

- Activate the virtual environment if you created one.

- Run the following command to install Flask and other required dependencies:

```

pip install Flask scikit-learn==1.2.1 pandas numpy matplotlib seaborn

``` 

**### Running the Flask Application Locally:**

**1. \*\*Starting the Flask Application\*\*:**

- Navigate to the directory containing your Flask application code.

- Ensure all necessary files and directories are in place, including templates and static files if applicable.

- Activate the virtual environment if you created one.

- Run the following command to start the Flask development server:

```

flask run

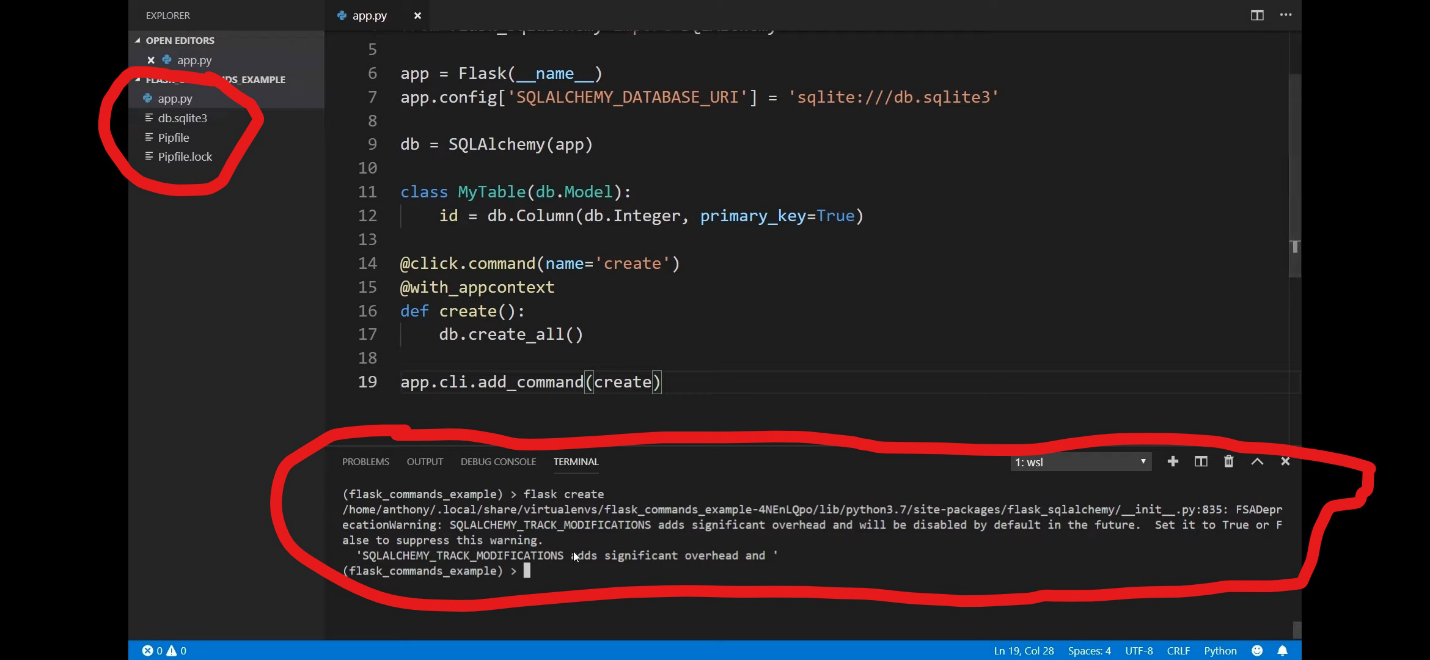
```

**2. \*\*Accessing the Application\*\*:**

- Once the Flask application is running, open a web browser.

- Enter the following URL in the address bar to access the application locally: `http://127.0.0.1:5000/`.

- You should see the home page or landing page of your Flask application.



**3. \*\*Testing the Application\*\*:**

- Test various functionalities of the Flask application to ensure everything is working as expected.

- Submit sample requests if your application includes APIs.

- Verify that the application responds correctly and handles errors gracefully.

**### Additional Notes:**

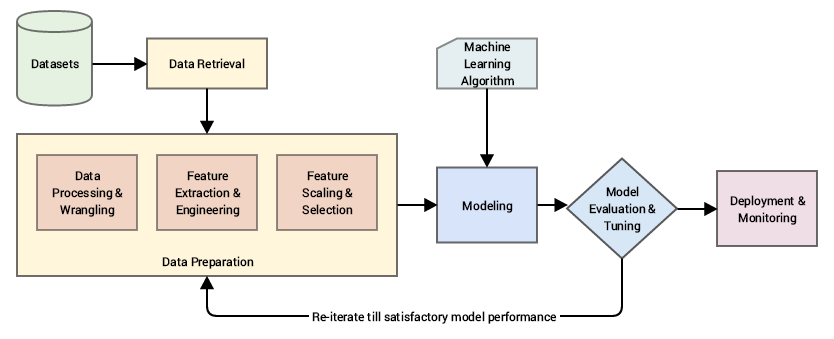
- Ensure that any necessary configuration settings (e.g., database configuration, environment variables) are properly set before running the Flask application.

- Always deactivate the virtual environment when you're finished working on your project by running the command `deactivate`.

---

This section 3.provides detailed instructions for setting up the Python environment, installing Flask and other dependencies, and running the Flask application locally. Follow these steps to ensure a smooth setup and installation process for your Flask project.

**4. Machine Learning Model**



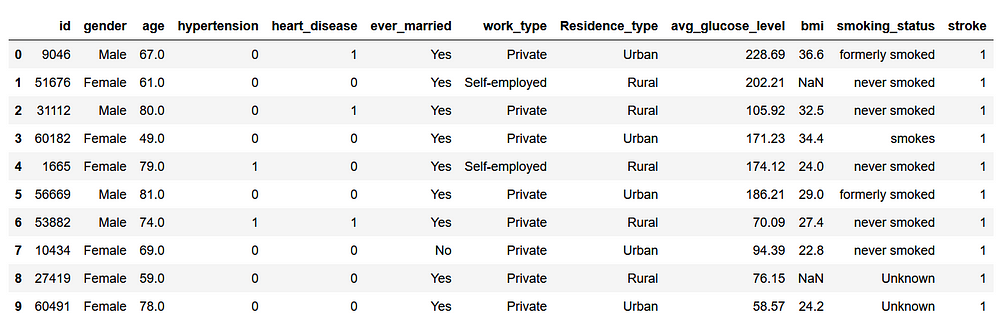
**### Description of the Dataset:**

- The dataset used for this project contains information about patients and various health-related attributes.

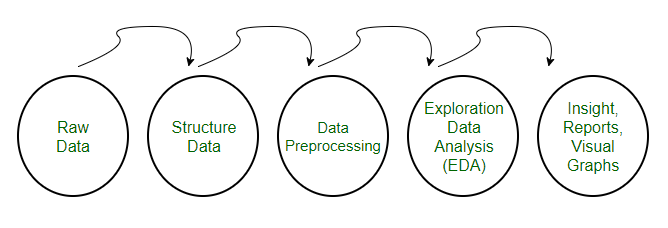
**Description of the Dataset:**

* The dataset contains information about patients, including:
  + Gender
  + Age
  + Hypertension status
  + Heart disease status
  + Marital status
  + Work type
  + Residence type
  + Average glucose level
  + Body mass index (BMI)
  + Smoking status
  + Stroke occurrence (target variable)
* Each row in the dataset represents a unique patient profile.

- Each row represents a unique patient profile, with the target variable being whether the patient had a stroke (1) or not (0).

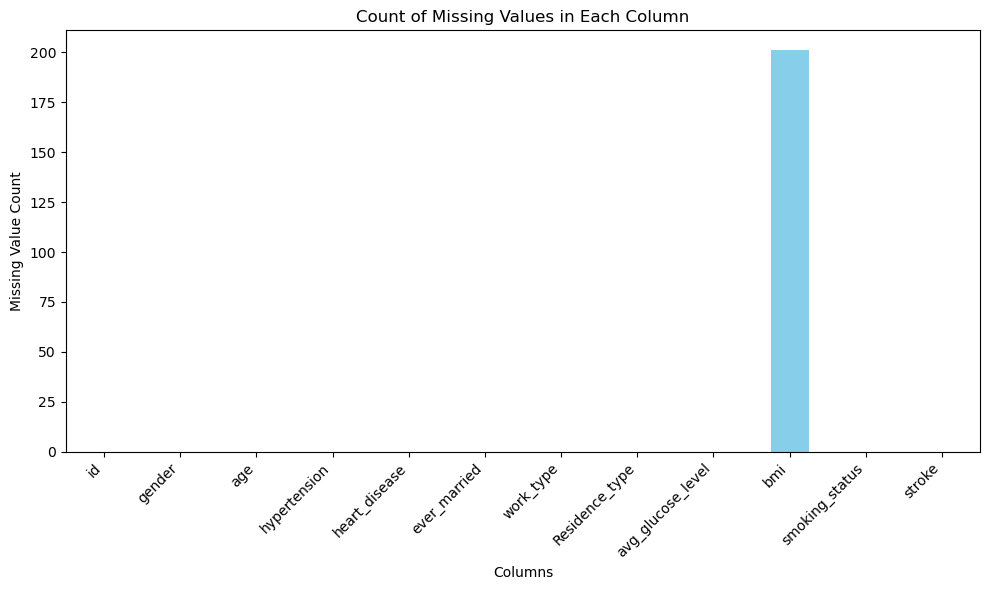


**### Data Preprocessing Steps:**

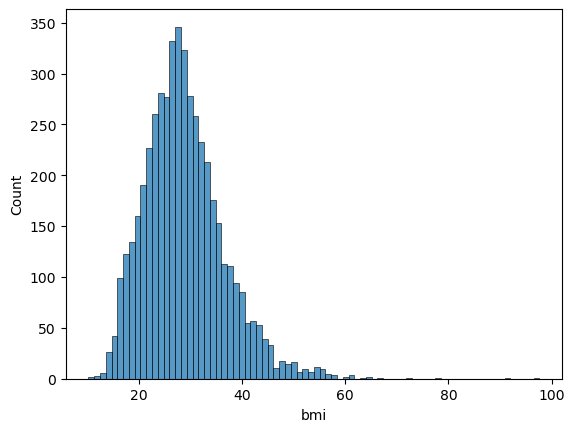


**1. \*\*Data Cleaning\*\*:**

- Handle missing values, if any, by imputation or removal.



* 1. DATA DISTRUBUSTION



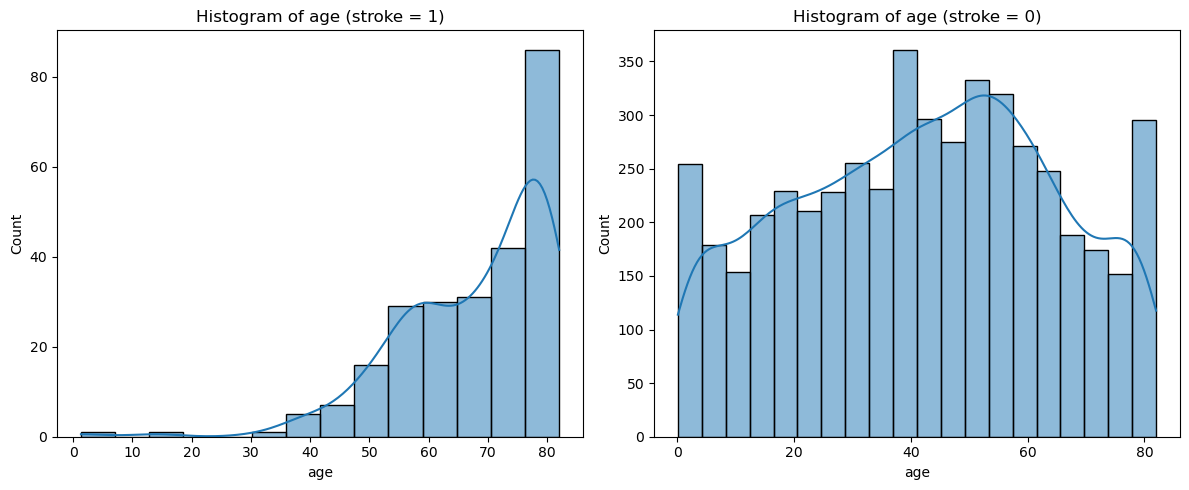
- Remove any duplicate entries from the dataset.

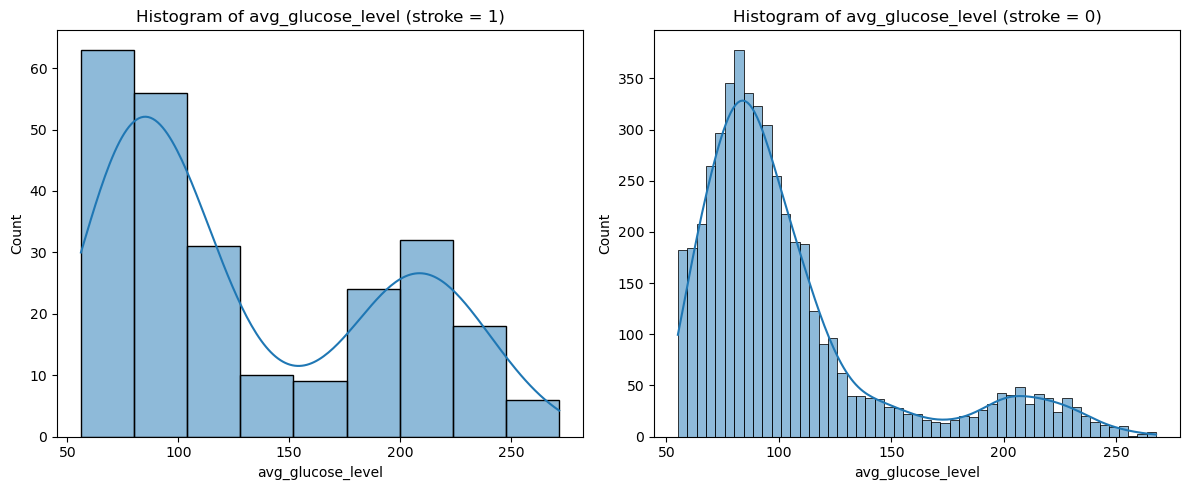
- Impute of bmi the null with mean

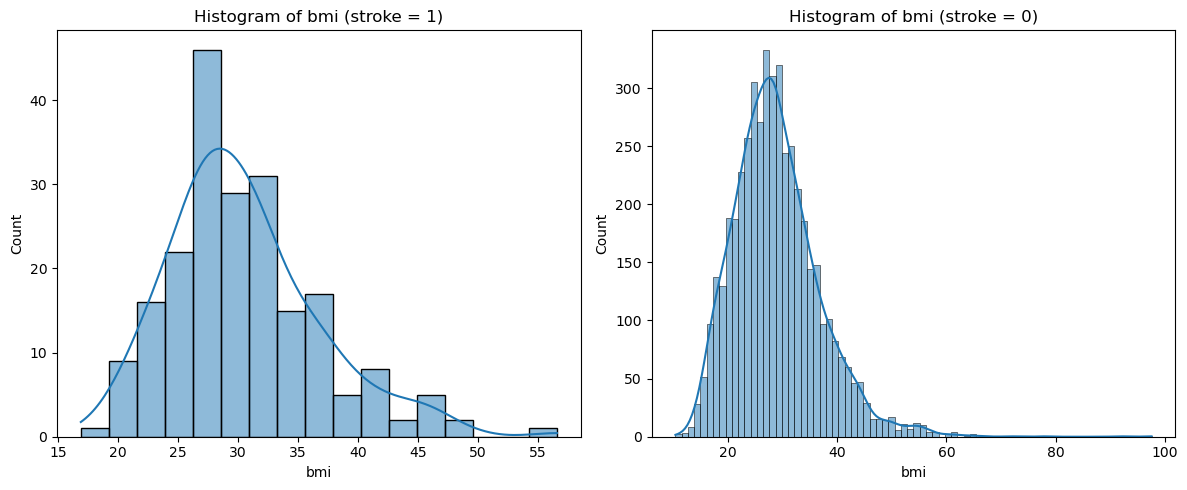
**2. \*\*Exploratory Data Analysis (EDA)\*\*:**

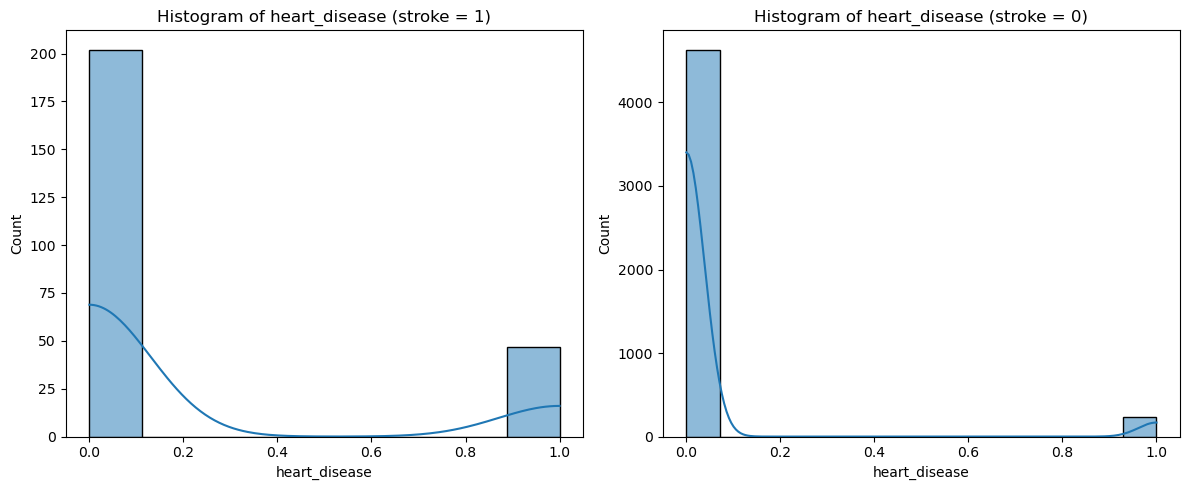
- Explore the distribution of each feature.

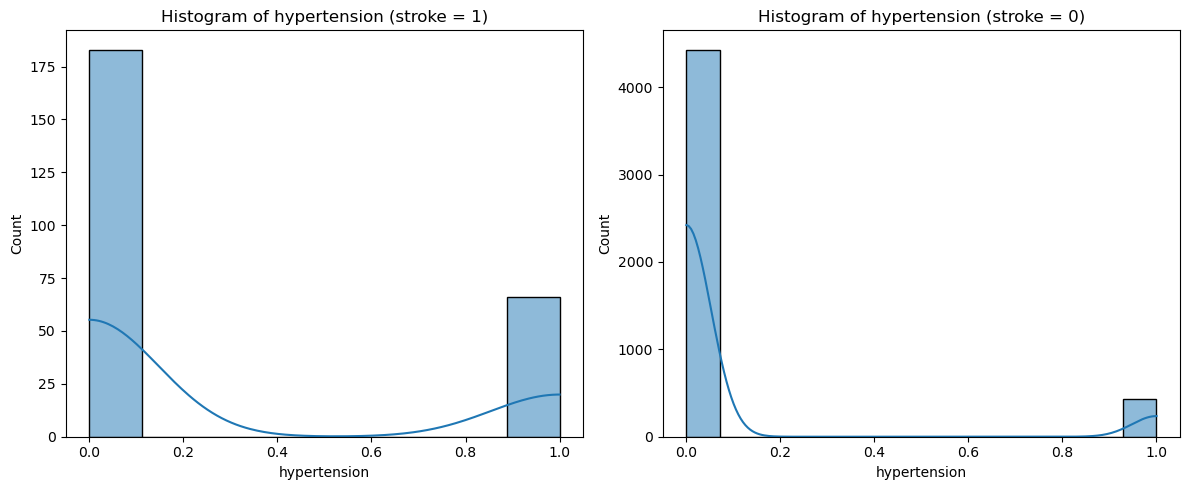
**Numerical features**

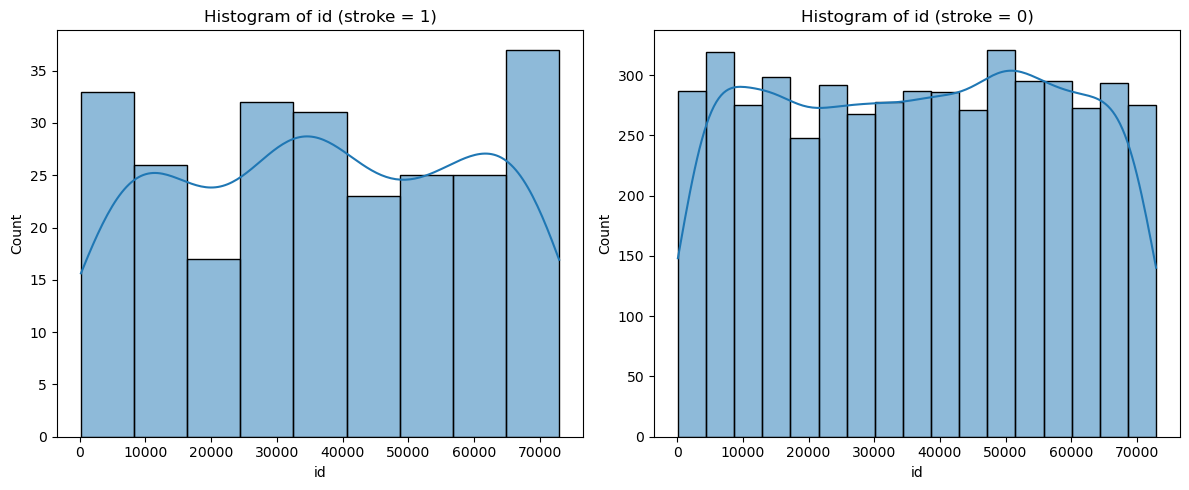




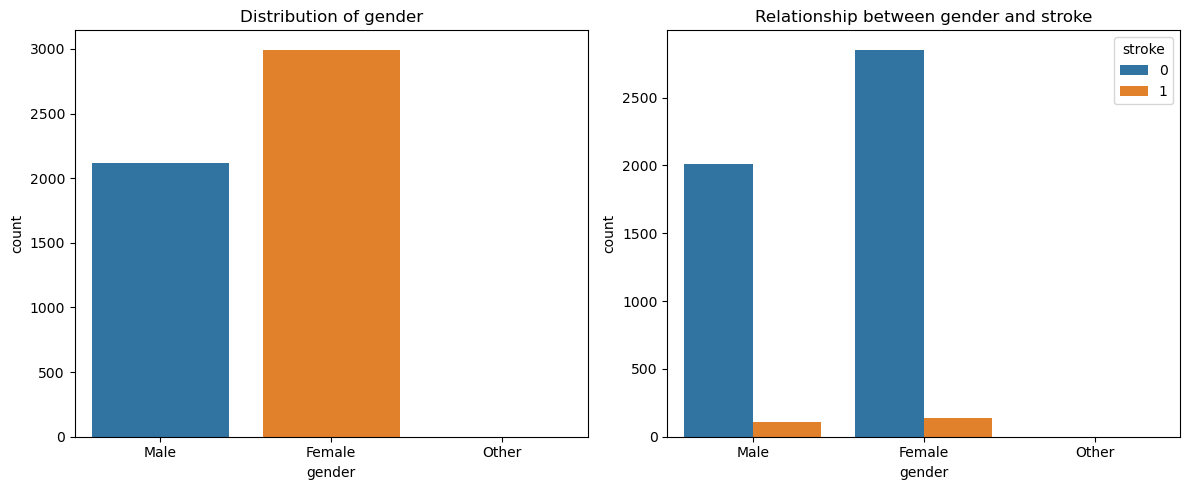


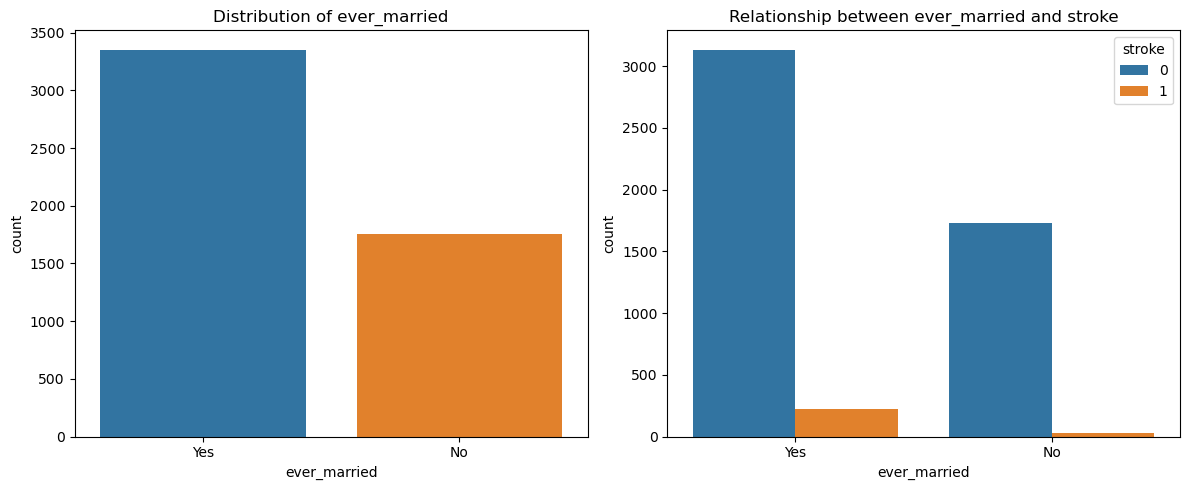


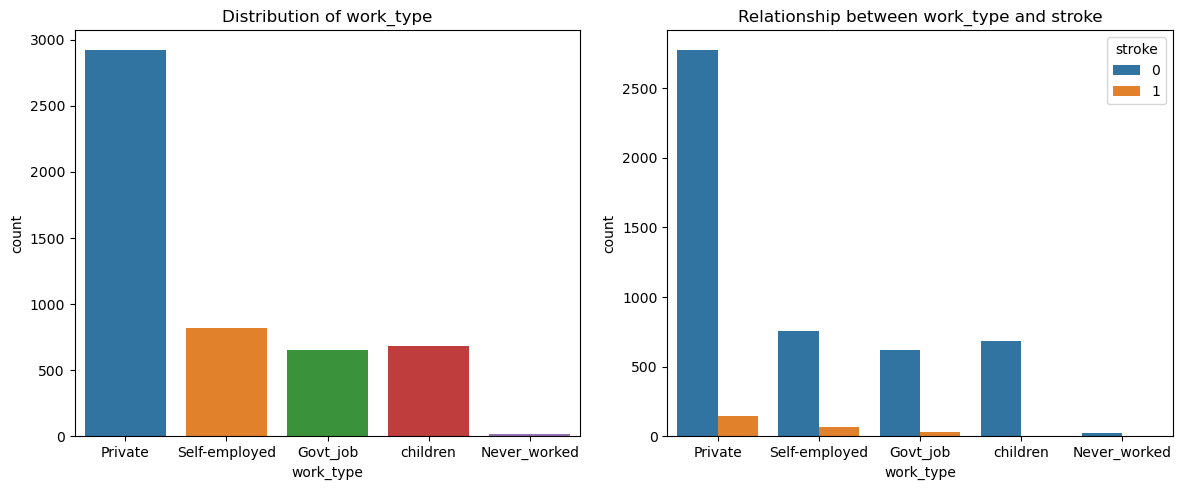




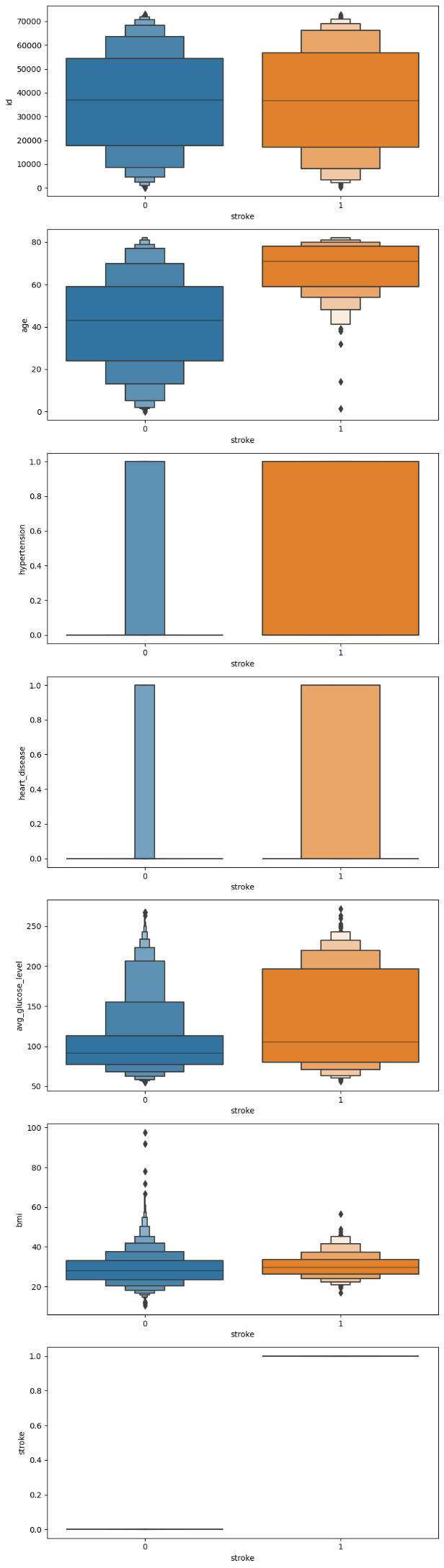
**Categorical features**

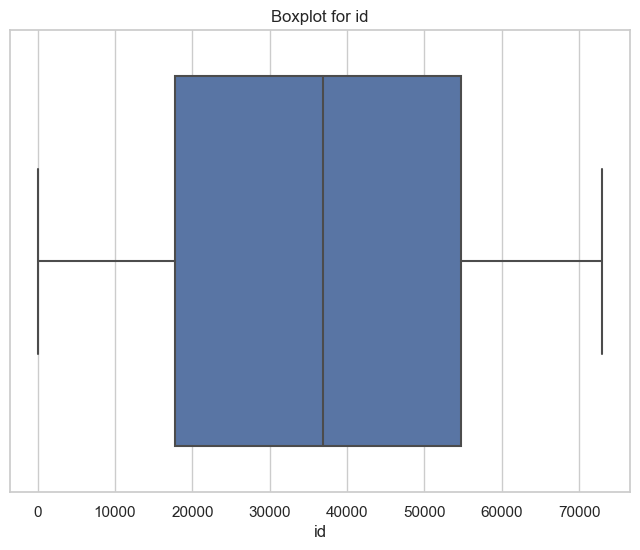


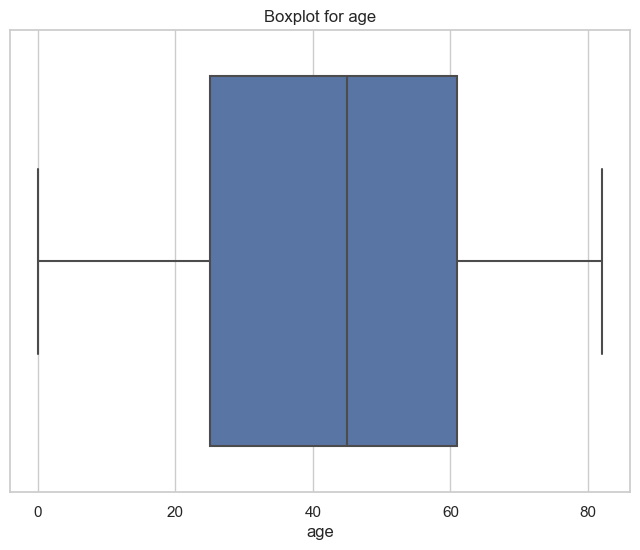


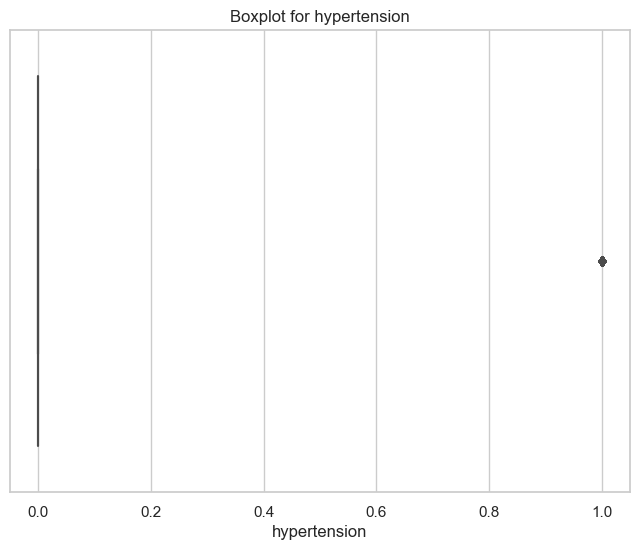


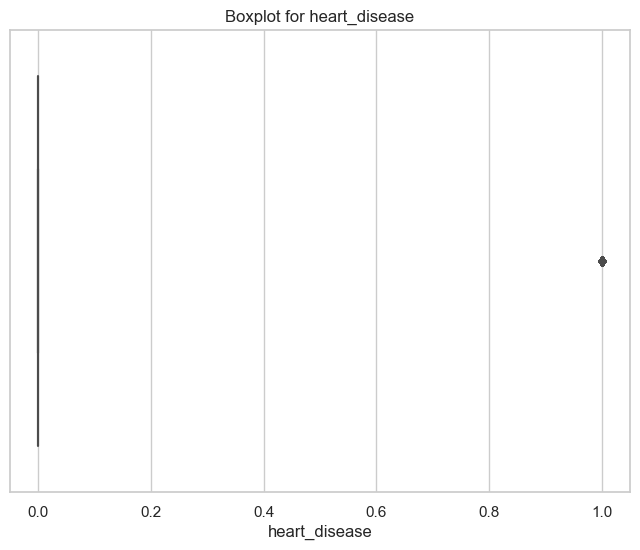
**- Identify any outliers or anomalies in the data.**

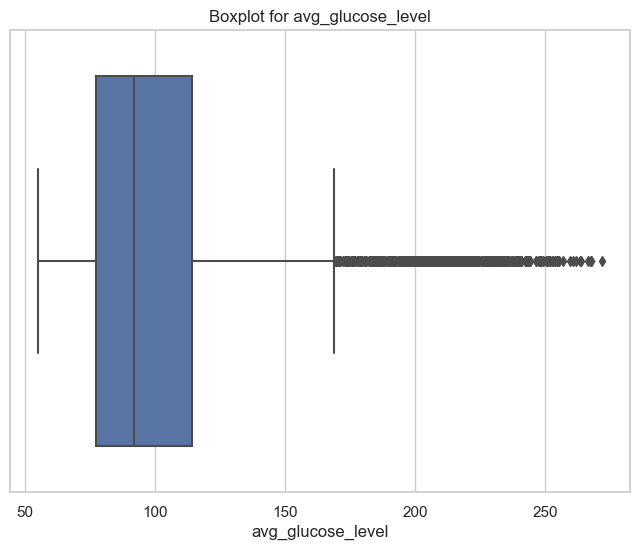


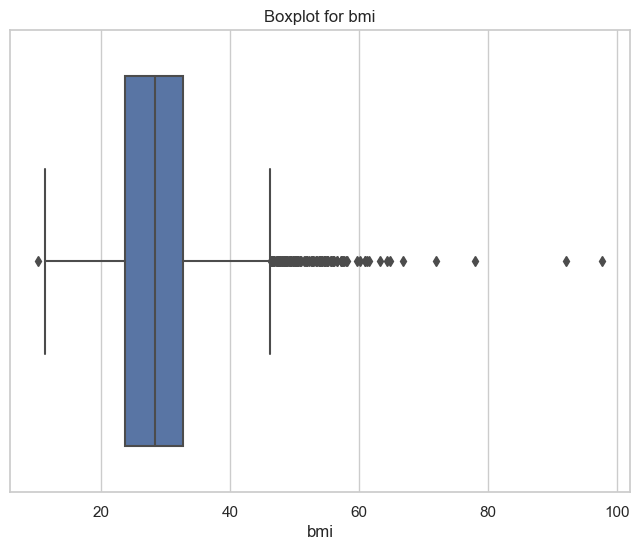


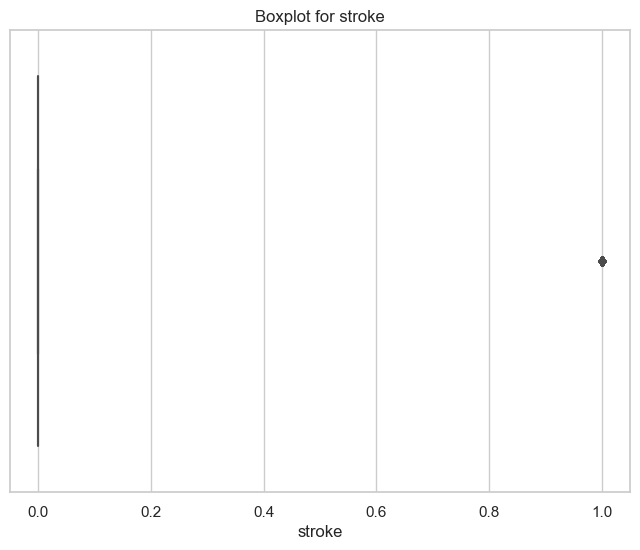




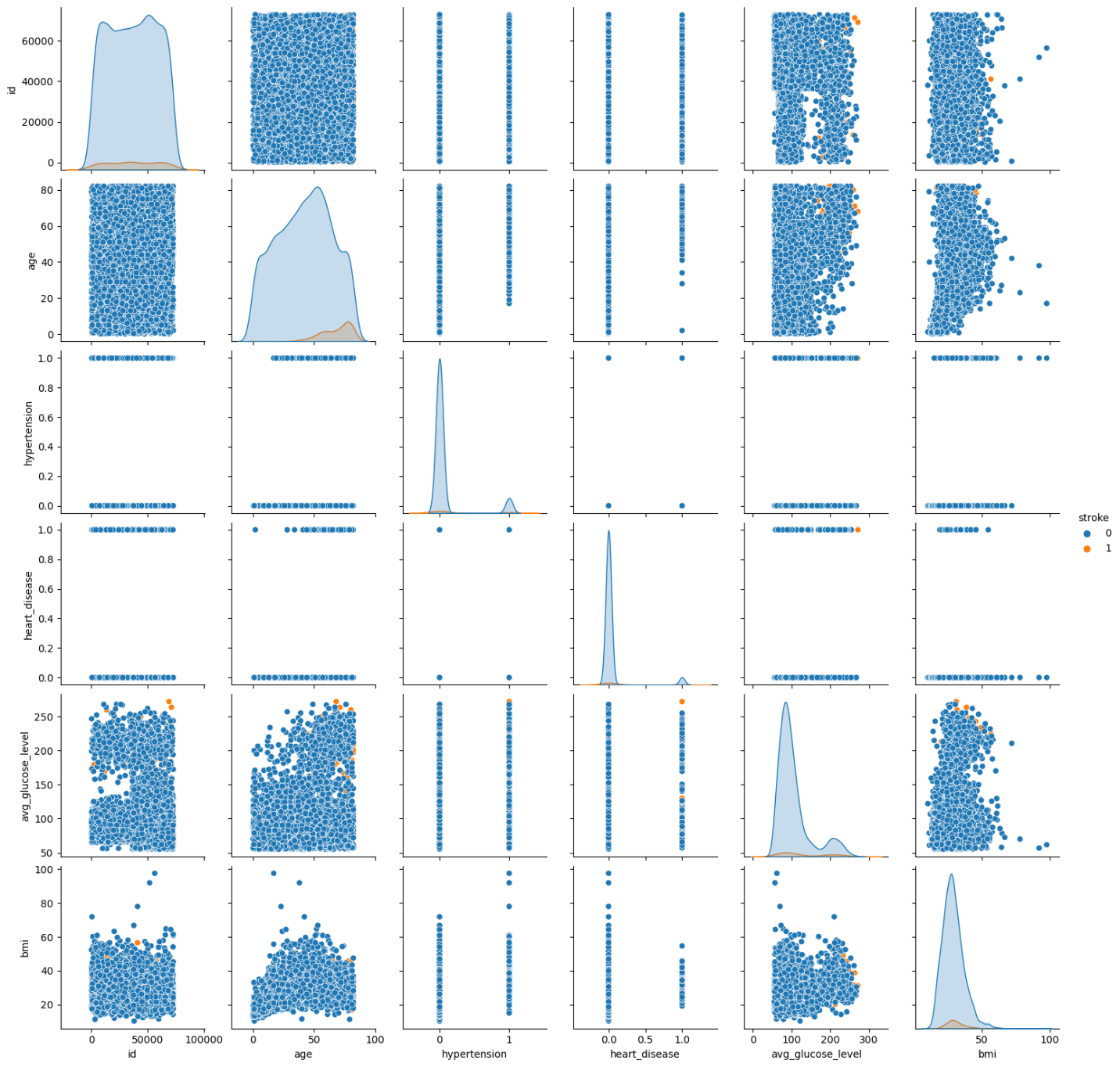








**- Analyze correlations between features and the target variable.**



**### Exploratory Data Analysis (EDA):**

**1. \*\*Data Visualization\*\*:**

- Generate histograms, box plots, and scatter plots to visualize the distribution and relationships between variables.

- Use seaborn and matplotlib libraries to create insightful visualizations.

**2. \*\*Statistical Analysis\*\*:**

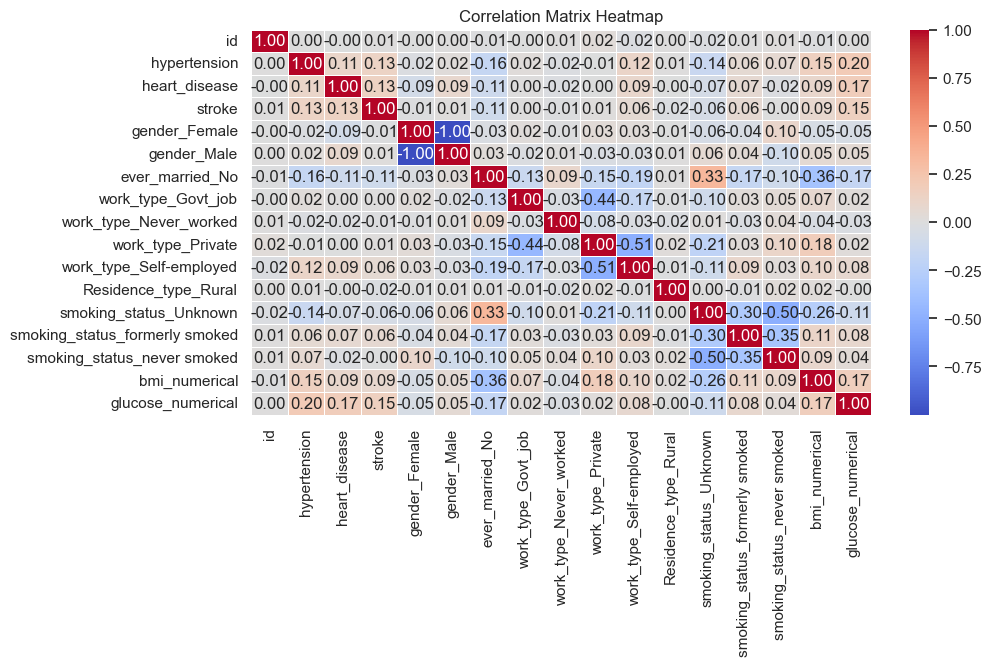
- Compute summary statistics such as mean, median, and standard deviation for numerical features.

- Conduct hypothesis testing or correlation analysis to uncover significant relationships in the data.

### **Model Selection and Training:**

**1. \*\*Feature Engineering\*\*:**

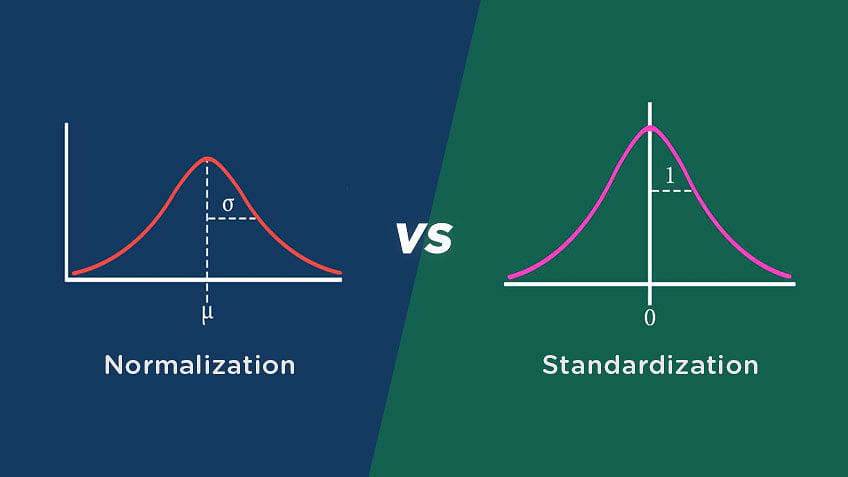
- Select relevant features for model training.



- Encode categorical variables using one-hot encoding or label encoding.



- Scale numerical features using techniques like min-max scaling or standardization.



**2. \*\*Model Selection\*\*:**

- Choose appropriate machine learning algorithms for classification tasks.

### Selecting the right model for binary classification involves considering factors such as the nature of your data, the characteristics of your problem, and the performance requirements. Here are some popular models for binary classification along with guidelines on when to use them:

1. \*\*Logistic Regression:\*\*

- Use when the relationship between features and the binary outcome is assumed to be linear.

- Suitable for problems where interpretability is important.

- Good for baseline performance.

2. \*\*Decision Trees:\*\*

- Useful when the relationship between features and the target variable is nonlinear and complex.

- Automatically handles feature interactions.

- Prone to overfitting, so consider using ensemble methods like Random Forests or Gradient Boosting.

3. \*\*Random Forest:\*\*

- Ensemble method that builds multiple decision trees and combines their predictions.

- Robust and less prone to overfitting compared to individual decision trees.

- Handles both numerical and categorical features well.

4. \*\*Gradient Boosting (e.g., XGBoost, LightGBM, CatBoost):\*\*

- Ensemble method that builds trees sequentially, each correcting the errors of the previous one.

- Often provides high predictive accuracy.

- Tends to perform well even with default hyperparameters.

5. \*\*Support Vector Machines (SVM):\*\*

- Suitable for both linear and nonlinear relationships between features and target variable.

- Effective in high-dimensional spaces.

- Works well for small to medium-sized datasets.

6. \*\*Neural Networks:\*\*

- Effective for complex, high-dimensional data.

- Requires large amounts of data for training and may be computationally intensive.

- Can capture intricate patterns but may lack interpretability.

7. \*\*K-Nearest Neighbors (KNN):\*\*

- Simple and intuitive algorithm.

- Effective when the decision boundary is irregular.

- Sensitive to outliers and may be computationally expensive for large datasets.

8. \*\*Naive Bayes:\*\*

- Assumes independence between features, making it less suitable for highly correlated features.

- Efficient and computationally inexpensive.

- Good choice for text classification or when the independence assumption holds.

When selecting a model, it's important to consider the trade-off between interpretability, predictive accuracy, and computational complexity. Additionally, it's advisable to experiment with multiple models and evaluate their performance using metrics like accuracy, precision, recall, F1 score, and area under the ROC curve (AUC-ROC) to get a comprehensive understanding of how well they perform on your specific problem. Cross-validation can also help assess the generalization performance of the models.

- Consider models such as decision trees, random forests, svm and gradient boosting.

**3. \*\*Model Training\*\*:**

- Split the dataset into training and testing sets.



- Train the selected models using the training data.

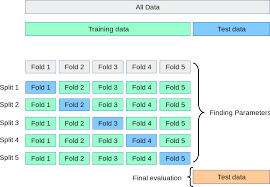
**### Model Evaluation and Validation:**

**1. \*\*Evaluation Metrics\*\*:**

- Evaluate model performance using metrics such as accuracy, precision, recall, F1-score, and area under the ROC curve (AUC-ROC).

**2. \*\*Cross-Validation\*\*:**

- Perform k-fold cross-validation to assess the robustness of the models.



**3. \*\*Model Comparison\*\*:**

- Compare the performance of different models to select the best-performing one.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Fold 1**  **Accuracy** | **Fold 2**  **Accuracy** | **Fold 3**  **Accuracy** | **Fold 4**  **Accuracy** | **Fold 5**  **Accuracy** | **Average Accuracy** |
| **Decision**  **Tree** | 0.9364 | 0.9325 | 0.9354 | 0.9423 | 0.9325 | 0.9358 |
| **Random**  **Forest** | 0.9423 | 0.9384 | 0.9393 | 0.9491 | 0.9423 | 0.9423 |
| **Gradient**  **Boosting** | 0.9511 | 0.9462 | 0.9501 | 0.9491 | 0.9481 | 0.9489 |
| **SVM** | 0.9511 | 0.9511 | 0.9511 | 0.9511 | 0.9521 | 0.9513 |

**4. \*\*Hyperparameter Tuning\*\*:**

- Fine-tune hyperparameters of the chosen model to optimize performance further.

---

This section provides a comprehensive overview of the machine learning model, including dataset description, data preprocessing steps, exploratory data analysis (EDA), model selection and training, and model evaluation and validation techniques.

**document outlining the API endpoints for the stroke prediction application:**

---

**## 6. API Endpoints**

**### Detailed Description of API Endpoints:**

1. \*\*Predict Stroke Endpoint\*\*:

- \*\*URL\*\*: `/predict`

- \*\*Method\*\*: POST

- \*\*Description\*\*: Endpoint to predict stroke occurrence based on input parameters.

- \*\*Parameters\*\*:

- `gender`: Gender of the patient (string: "Male", "Female", "Other").

- `age`: Age of the patient (integer).

- `hypertension`: Hypertension status of the patient (integer: 0 or 1).

- `heart\_disease`: Heart disease status of the patient (integer: 0 or 1).

- `ever\_married`: Marital status of the patient (string: "No", "Yes").

- `work\_type`: Work type of the patient (string: "children", "Govt\_job", "Never\_worked", "Private", "Self-employed").

- `residence\_type`: Residence type of the patient (string: "Rural", "Urban").

- `avg\_glucose\_level`: Average glucose level in blood of the patient (float).

- `bmi`: Body mass index of the patient (float).

- `smoking\_status`: Smoking status of the patient (string: "formerly smoked", "never smoked", "smokes", "Unknown").

- \*\*Response\*\*:

- `prediction`: Predicted stroke occurrence (integer: 0 or 1).

### Request and Response Formats:

- \*\*Request Format\*\*:

- JSON format containing input parameters as key-value pairs.

```

{

"gender": "Male",

"age": 65,

"hypertension": 1,

"heart\_disease": 0,

"ever\_married": "Yes",

"work\_type": "Private",

"residence\_type": "Urban",

"avg\_glucose\_level": 120.56,

"bmi": 28.3,

"smoking\_status": "never smoked"

}

```

- \*\*Response Format\*\*:

- JSON format containing the predicted stroke occurrence.

```

{

"prediction": 1

}

```

### Example API Calls:

1. \*\*Example API Call - Predict Stroke\*\*:

- \*\*URL\*\*: `http://localhost:5000/predict`

- \*\*Method\*\*: POST

- \*\*Request Body\*\*:

```

{

"gender": "Female",

"age": 45,

"hypertension": 0,

"heart\_disease": 0,

"ever\_married": "No",

"work\_type": "Private",

"residence\_type": "Rural",

"avg\_glucose\_level": 85.2,

"bmi": 24.7,

"smoking\_status": "never smoked"

}

```

- \*\*Response\*\*:

```

{

"prediction": 0

}

```

- \*\*Description\*\*: Predicts stroke occurrence for a 45-year-old unmarried female living in a rural area with no hypertension, heart disease, and a healthy lifestyle.

This document provides a detailed description of the API endpoints for the stroke prediction application, including request and response formats and example API calls. Follow these specifications to interact with the API effectively and obtain predictions for stroke occurrence based on input parameters.

**Here's the document outlining the user interface for the stroke prediction application:**

---

**## 7. User Interface**

### Designing the Web Interface:

1. \*\*Homepage\*\*:

- Display an introduction to the stroke prediction application.

- Provide options for users to input their information for stroke prediction.

2. \*\*Input Form\*\*:

- Design a form with input fields for gender, age, hypertension status, heart disease status, marital status, work type, residence type, average glucose level, body mass index (BMI), and smoking status.

- Include appropriate labels and placeholders for each input field.

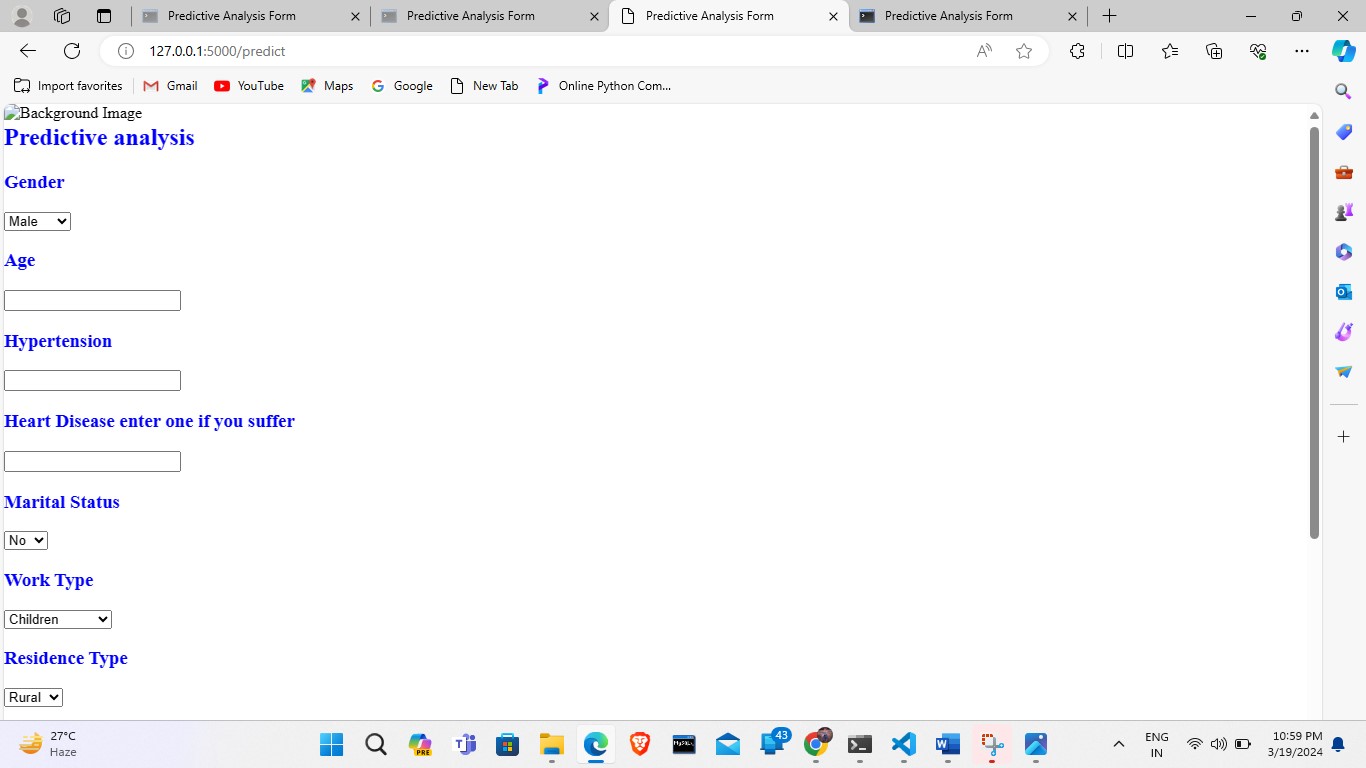
- Add a submit button for users to submit their information.

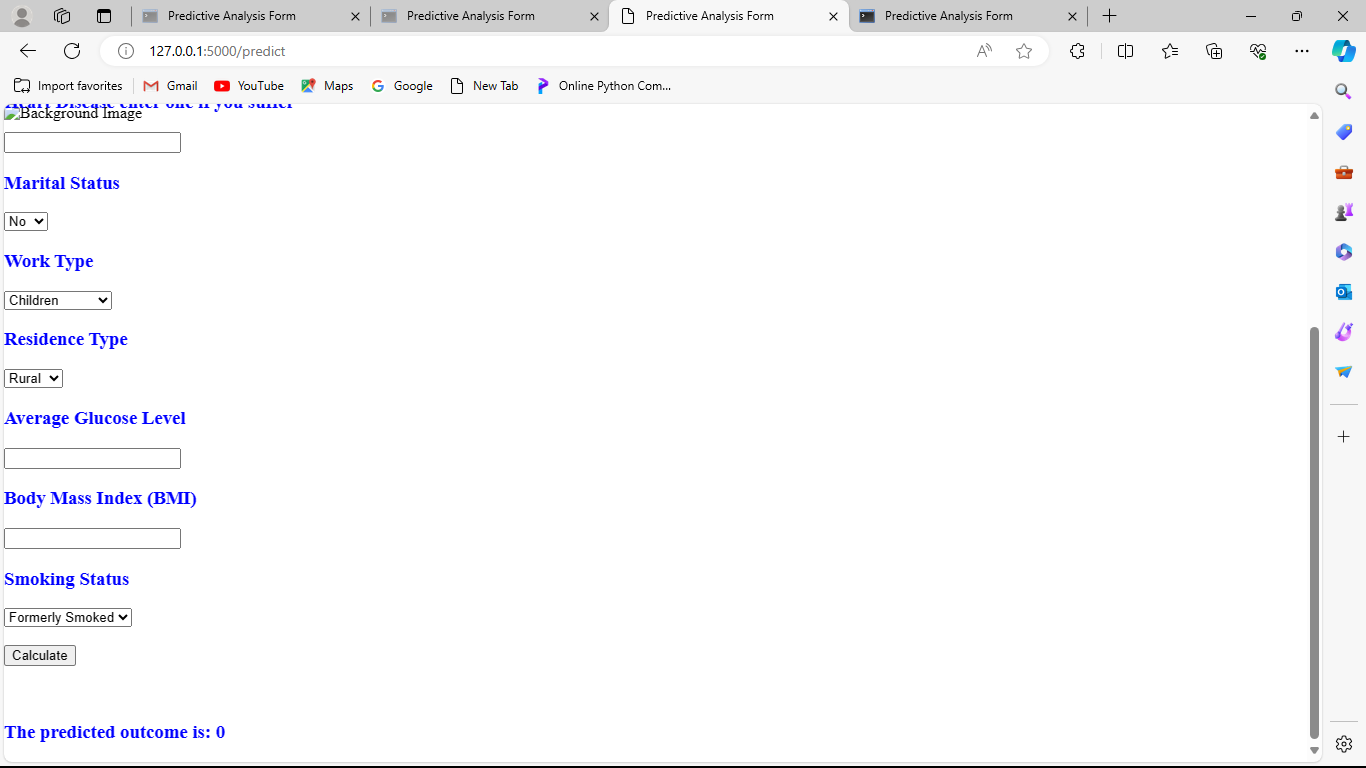
3. \*\*Result Display\*\*:

- Display the predicted stroke occurrence based on the user's input.

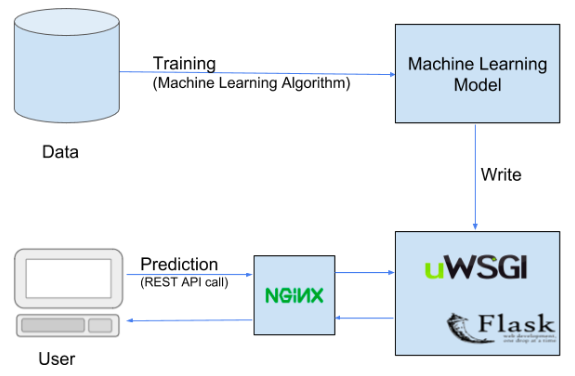
- Provide clear and concise messaging indicating whether the user is at risk of stroke or not.

- Optionally, provide additional information or resources for stroke prevention and management.





**### Connecting the Interface with Flask:**



1. \*\*HTML Templates\*\*:

- Create HTML templates for the homepage, input form, and result display using Flask's template engine (Jinja2).

- Include placeholders for dynamic content that will be populated by Flask.

2. \*\*Routes\*\*:

- Define Flask routes to render the HTML templates and handle form submissions.

- Map each route to the corresponding HTML template.

3. \*\*Form Submission\*\*:

- Handle form submissions in Flask routes by retrieving input data from the request.

- Process the input data, make predictions using the machine learning model, and return the result to the user.

**### User Interaction Flow:**

1. \*\*Homepage\*\*:

- Users land on the homepage, which provides an overview of the stroke prediction application and instructions on how to use it.

2. \*\*Input Form\*\*:

- Users fill out the input form with their personal information, including gender, age, medical history, lifestyle factors, etc.

3. \*\*Form Submission\*\*:

- Upon submitting the form, the input data is sent to the Flask backend for processing.

- The Flask application uses the machine learning model to predict the likelihood of stroke based on the user's input.

4. \*\*Result Display\*\*:

- The predicted stroke occurrence is displayed to the user on the result page.

- Users receive clear feedback indicating whether they are at risk of stroke or not.

**### Additional Notes:**

- Ensure the web interface is intuitive and user-friendly, with clear instructions and error messages where applicable.

- Validate user input on the frontend and backend to prevent errors and ensure data integrity.

- Implement responsive design principles to ensure the interface is accessible and functional across different devices and screen sizes.

---

This document provides guidelines for designing the user interface, connecting it with Flask, and defining the user interaction flow for the stroke prediction application. Follow these specifications to create an effective and user-friendly interface for users to interact with the application.

**Here's the conclusion section for the stroke prediction application:**

---

**## 8. Conclusion**

**### Summary of the Project:**

The stroke prediction application aims to leverage machine learning techniques to assess an individual's risk of experiencing a stroke based on various demographic, medical, and lifestyle factors. The project involves data analysis, model development, web application deployment, and user interface design.

The project begins with data exploration and preprocessing, where the dataset containing patient information is cleaned, encoded, and prepared for model training. Machine learning algorithms are then trained on the preprocessed data to predict stroke occurrence. The trained model is integrated into a Flask web application, allowing users to input their information and receive personalized stroke risk predictions.

**### Future Work and Improvements:**

**1. \*\*Model Enhancement\*\*:**

- Explore advanced machine learning algorithms and techniques to improve the accuracy and reliability of stroke predictions.

- Incorporate additional features or data sources to capture more comprehensive patient profiles and enhance model performance.

**2. \*\*User Interface Enhancements\*\*:**

- Implement user feedback mechanisms to enhance user experience and engagement with the application.

- Integrate visualization tools to provide interactive insights into the factors influencing stroke risk and the importance of preventive measures.

**3. \*\*Scalability and Deployment\*\*:**

- Optimize the deployment process to ensure scalability and reliability of the application, enabling seamless access for a larger user base.

- Explore cloud deployment options and containerization techniques for efficient management and scalability of the application infrastructure.

**4. \*\*Community Engagement\*\*:**

- Collaborate with healthcare professionals, researchers, and stakeholders to validate and refine the predictive model and application functionalities.

- Engage with the community to raise awareness about stroke prevention and encourage proactive healthcare practices.

**5. \*\*Data Privacy and Security\*\*:**

- Implement robust data privacy measures to protect sensitive patient information and ensure compliance with regulatory requirements (e.g., GDPR, HIPAA).

- Strengthen security measures to safeguard the application against potential cyber threats and unauthorized access.

**### Conclusion:**

The stroke prediction application represents a significant step towards leveraging data-driven approaches to enhance healthcare outcomes and empower individuals to make informed decisions about their health. By combining machine learning, web development, and user interface design, the project demonstrates the potential of technology to address complex healthcare challenges and improve patient well-being.

---

This conclusion section summarizes the key aspects of the stroke prediction project and outlines potential avenues for future work and improvements. It highlights the project's significance in leveraging technology to advance healthcare outcomes and empower individuals to take proactive steps towards stroke prevention and management.

Here's the section for references and acknowledgments:

---

**## References**

### Datasets:

- World Health Organization (WHO) Stroke Dataset: [Dataset Source](<Provide link to the dataset if available>)

### Libraries and Tools:

- Python Programming Language: [Python Official Website](https://www.python.org/)

- Flask Web Framework: [Flask Documentation](https://flask.palletsprojects.com)

- scikit-learn Machine Learning Library: [scikit-learn Documentation](https://scikit-learn.org/stable/)

- Pandas Data Analysis Library: [Pandas Documentation](https://pandas.pydata.org/)

- NumPy Scientific Computing Library: [NumPy Documentation](https://numpy.org/)

- Matplotlib Data Visualization Library: [Matplotlib Documentation](https://matplotlib.org/)

- Seaborn Statistical Data Visualization Library: [Seaborn Documentation](<https://seaborn.pydata.org/>)

Github : https://github.com/vijay33391/Heroku-Demo

### Acknowledgments:

- We would like to express our gratitude to the World Health Organization (WHO) for providing the stroke dataset used in this project.

- Special thanks to the developers and contributors of the Python programming language, Flask web framework, scikit-learn machine learning library, and other open-source tools and libraries used in this project.

- We also extend our appreciation to the research community for their valuable insights and contributions to the field of stroke prediction and prevention.

---

This section provides references to datasets, libraries, and tools used in the project, as well as acknowledgments to organizations, developers, and contributors who have contributed to the project's success.