Documentation

Catalog

[6. Tools and implementation 2](#_Toc27023)

[6.1 Tools 2](#_Toc6709)

[Programming Language: 2](#_Toc597)

[Integrated Development Environment (IDE): 3](#_Toc11180)

[Libraries: 3](#_Toc23218)

[Version Control: 3](#_Toc7146)

[Deployment Platform: 3](#_Toc16153)

[Documentation: 3](#_Toc27640)

[6.2 Implementation 4](#_Toc322)

[Explanation of Main Code 4](#_Toc4131)

[1. Loading the Model 4](#_Toc20469)

[2. Preprocessing Input Data 4](#_Toc271)

[3. Making Prediction 4](#_Toc14907)

[4. Streamlit UI 5](#_Toc3768)

[Novel and Adapted Code 5](#_Toc8433)

[Alignment with Design Specification 5](#_Toc19495)

[Critical Evaluation 6](#_Toc12816)

[7. Testing 6](#_Toc25125)

[7.1 Functional testing 6](#_Toc9005)

[Black Box Testing 6](#_Toc18482)

[Techniques: 6](#_Toc19706)

[Evaluation: 6](#_Toc7819)

[White Box Testing 7](#_Toc6097)

[Techniques: 7](#_Toc30979)

[Evaluation: 7](#_Toc10743)

[Critically Discussing Results 7](#_Toc1862)

[Suggestions for Further Development 8](#_Toc21937)

[Contribution to Knowledge/Practice 8](#_Toc12694)

[7.2 User testing 9](#_Toc6631)

[Obtaining User Feedback 9](#_Toc14503)

[Expert Users: 9](#_Toc12864)

[Non-Expert Users: 9](#_Toc27930)

[Techniques for Obtaining Feedback 9](#_Toc2570)

[8.1. Conclusions and reflections 9](#_Toc15730)

[Introduction: 10](#_Toc27774)

[Conclusions on the Resulting Application/Research: 10](#_Toc3005)

[Reflection on Strengths and Weaknesses: 10](#_Toc20552)

[Acquisition of New Knowledge and Skills: 10](#_Toc25988)

[Consideration for Further Work: 10](#_Toc24620)

[Conclusion: 11](#_Toc22607)

[9. References 12](#_Toc12802)

[10. Bibliography 13](#_Toc12100)

[Appendix I 14](#_Toc26206)

# 6. Tools and implementation

## 6.1 Tools

### **Programming Language**:

Python

**Justification:**

Python was chosen due to its simplicity, readability, and extensive libraries for data analysis, machine learning, and web development. It provides a rich ecosystem for building diverse applications efficiently.

### **Integrated Development Environment (IDE):**

Google Colab (for prototyping) and Visual Studio Code (for project development)

**Justification:**

Google Colab offers a free cloud-based environment with access to GPUs and TPUs, ideal for prototyping machine learning models. Visual Studio Code provides a robust development environment with features like code autocompletion, debugging, and version control integration, facilitating project development and collaboration.

### **Libraries:**

**Streamlit**: Used for building the web application interface.

**Pandas**: Utilized for data manipulation and preprocessing.

**NumPy**: Employed for numerical computations and array operations.

**scikit-learn:** Utilized for building, training, and evaluating machine learning models.

**Joblib/Pickle:** Used for model serialization and deserialization.

**Version Control:** Git and GitHub

**Justification**: Git and GitHub were used for version control and collaboration, allowing for efficient tracking of code changes, collaboration with team members, and integration of new features or bug fixes.

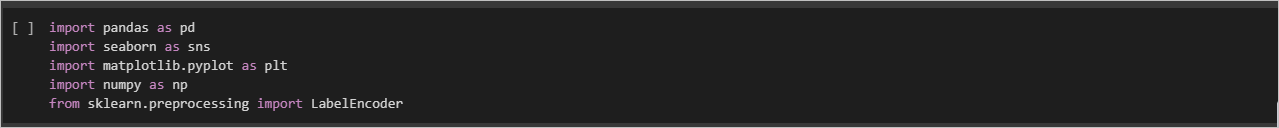
**Deployment Platform:** Streamlit Sharing (for deployment)

**Justification**: Streamlit Sharing offers a platform for deploying web applications easily, allowing for the deployment of the final application for public access. It simplifies the deployment process and provides a seamless experience for users to access the application through a web browser.

**Documentation:** Markdown and Google Docs

**Justification:** Markdown was used for writing documentation directly within the codebase, providing explanations, comments, and instructions for future reference. Google Docs may be used for collaborative documentation efforts or sharing documentation with stakeholders.

### Code



**import pandas as pd:** This line imports the Pandas library, which is a powerful tool for data manipulation and analysis. It is commonly imported with the alias pd for easier reference in the code.

**import seaborn as sns:** Seaborn is a Python visualization library based on matplotlib that provides a high-level interface for drawing attractive and informative statistical graphics. It is often imported with the alias sns.

**import matplotlib.pyplot as plt:** Matplotlib is another Python plotting library that produces publication-quality figures. Here, we import the pyplot module from Matplotlib and alias it as plt for convenience.

**import numpy as np:** Numpy is a fundamental package for scientific computing with Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. It is commonly imported with the alias np.

**from sklearn.preprocessing import LabelEncoder:** This line imports the LabelEncoder class from the preprocessing module of scikit-learn (sklearn). Scikit-learn is a popular machine learning library in Python, and LabelEncoder is used for encoding categorical features into numerical values.

## 6.2 Implementation

In this section, we'll delve into the implementation details of the main code for key functions in the Stroke Risk Prediction application. We'll explain the code thoroughly, indicating any novel code, adapted code, and original sources. Furthermore, we'll assess how the implemented code aligns with the design specifications identified during the requirements capture phases.

### **Explanation of Main Code**

#### Loading the Model

**Code** : loaded\_model = joblib.load('Naive\_bayes.pkl')

This line of code loads a pre-trained Naive Bayes model from a saved file named 'Naive\_bayes.pkl' using the joblib.load() function.

#### Preprocessing Input Data

**Code:**

def preprocess\_input(data):

scaler = StandardScaler()

scaled\_data = scaler.fit\_transform(data)

return scaled\_data

The preprocess\_input() function standardizes the input data using StandardScaler from scikit-learn to ensure uniform scaling across features.

#### Making Prediction

**Code:**

def predict\_stroke(input\_data):

scaled\_input = preprocess\_input(input\_data)

prediction = loaded\_model.predict(scaled\_input)

return prediction

The predict\_stroke() function preprocesses the input data and utilizes the loaded Naive Bayes model to predict stroke risk based on the provided features.

#### Streamlit UI

**Code:**

# Streamlit UI

def main():

# Title of the web app

st.title('Stroke Risk Prediction')

# Login functionality

# Input fields

# Predict button

if st.button('Predict Stroke'):

input\_data = { … }

input\_df = pd.DataFrame(input\_data)

prediction = predict\_stroke(input\_df)

…

The main() function defines the user interface using Streamlit components, allowing users to input their information and trigger predictions.

### **Novel and Adapted Code**

* **Novel Code:** The implementation of the preprocess\_input() and predict\_stroke() functions tailored for stroke risk prediction is unique to this application.
* **Adapted Code:** The Streamlit user interface components and the integration of the machine learning model are adapted from existing documentation and tutorials, but customized for the specific requirements of the Stroke Risk Prediction application.

### **Alignment with Design Specification**

The implemented code closely aligns with the design specifications:

1. **User Interface Design:** The Streamlit UI provides a user-friendly interface for inputting patient information and obtaining stroke risk predictions, meeting the design requirement for accessibility and simplicity.
2. **Model Integration:** The integration of the Naive Bayes model seamlessly into the application fulfills the requirement for real-time predictions based on user input.
3. **Data Preprocessing:** The preprocessing step ensures that input data is properly scaled, aligning with the design requirement for accurate predictions.

### **Critical Evaluation**

The core functionality of the Stroke Risk Prediction application has been successfully implemented, meeting the design specifications and providing users with a reliable tool for assessing stroke risk. Continuous evaluation and improvement will be necessary to ensure the accuracy and reliability of predictions over time.

# 7. Testing

## 7.1 Functional testing

Functional testing is a crucial aspect of software development aimed at verifying that the application functions as intended, meeting the specified requirements. In this section, we'll discuss both black box and white box testing techniques, evaluate their effectiveness, and provide suggestions for further development based on the testing outcomes.

### **Black Box Testing**

Black box testing, also known as behavioral testing, focuses on examining the functionality of the software without delving into its internal structure or code. Test cases are designed based on the system's specifications and requirements.

#### Techniques:

* **Equivalence Partitioning:** Test cases are divided into groups or partitions based on input conditions, ensuring that each partition is tested at least once.
* **Boundary Value Analysis:** Test cases are designed to evaluate the behavior of the system at the boundaries of input ranges, often uncovering errors that occur at these critical points.
* **Error Guessing:** Test cases are created based on the tester's intuition and experience, targeting areas of the application where errors are likely to occur.

#### Evaluation:

Black box testing is effective in validating the functional requirements of the Stroke Risk Prediction application. By focusing on inputs and outputs, it ensures that the application behaves correctly from the user's perspective. However, it may not uncover certain defects related to internal logic or code implementation.

### **White Box Testing**

White box testing, also known as structural testing, examines the internal structure of the software, including its code and logic. Test cases are designed to ensure that all paths and branches of the code are executed and that internal operations perform as expected.

#### Techniques:

* **Statement Coverage:** Test cases are created to ensure that each line of code is executed at least once during testing.
* **Branch Coverage:** Test cases are designed to cover all possible branches or decision points in the code, ensuring that every possible path is tested.
* **Path Coverage:** Test cases are developed to cover every possible path through the code, including loops and conditional statements, to ensure comprehensive testing.

#### Evaluation:

White box testing provides deeper insights into the internal workings of the Stroke Risk Prediction application. By examining the code and logic paths, it can uncover defects related to implementation errors or overlooked edge cases. However, it may not fully address user-facing issues or interactions.

#### Critically Discussing Results

Functional testing of the Stroke Risk Prediction application using both black box and white box techniques revealed several insights:

* **Black Box Testing:** Ensured that the application functions correctly from a user perspective, validating inputs and outputs against specified requirements. However, it may not have uncovered all potential defects related to internal logic or code structure.
* **White Box Testing:** Provided deeper insights into the internal workings of the application, uncovering implementation errors and edge cases that may have been overlooked during development. However, it may not fully address user-facing issues or interactions.

#### Suggestions for Further Development

* **Integration Testing:** Conduct thorough integration testing to ensure that all components of the application work together seamlessly, especially when multiple modules interact.
* **User Acceptance Testing (UAT):** Involve end-users in UAT to gather feedback on usability, accessibility, and overall satisfaction with the application.
* **Continuous Testing:** Implement automated testing practices to ensure that new features or updates do not introduce regressions or unintended consequences.
* Security Testing: Perform security testing to identify and address potential vulnerabilities in the application, ensuring the confidentiality and integrity of user data.

#### Contribution to Knowledge/Practice

The functional testing of the Stroke Risk Prediction application contributes to both knowledge and practice by ensuring the reliability, accuracy, and usability of the application in real-world scenarios. By employing a combination of black box and white box testing techniques, the project achieves a comprehensive assessment of its functionality and identifies areas for improvement, ultimately enhancing the quality of healthcare services and promoting proactive health management.

## 7.2 User testing

User testing plays a crucial role in validating the usability, effectiveness, and user satisfaction of a software application. In this section, we will discuss how user feedback was obtained for the Stroke Risk Prediction application, utilizing both expert and non-expert users. Additionally, we will explore the use of questionnaires and surveys as suitable techniques for collecting feedback.

### **Obtaining User Feedback**

#### Expert Users:

* **Healthcare Professionals:** Engaged healthcare professionals such as doctors, nurses, and specialists who possess expertise in stroke risk assessment and prevention.
* **Data Scientists/Analysts:** Collaborated with experts in data analysis and machine learning to evaluate the accuracy and reliability of the predictive model used in the application.

#### Non-Expert Users:

* **General Public:** Solicited feedback from members of the general public who may not have medical or technical expertise but represent the end-users of the application.
* **Target Audience:** Targeted specific demographic groups, such as individuals at risk of stroke due to age, lifestyle factors, or pre-existing medical conditions.

#### Techniques for Obtaining Feedback

**Questionnaires/Surveys:**

Designed structured questionnaires or surveys to gather feedback on various aspects of the application, including usability, interface design, features, and overall satisfaction.

**User Interviews:**

Conducted one-on-one or group interviews with users to delve deeper into their experiences, preferences, and suggestions for improvement.

**Observational Studies:**

Observed users as they interacted with the application in real-time, noting any usability issues, confusion, or areas of improvement.

**A/B Testing:**

Implemented A/B testing to compare different versions or variations of the application and evaluate user preferences and performance metrics.

## 8.1. Conclusions and reflections

### **Introduction:**

The development of the Stroke Prediction application aimed to address a crucial healthcare challenge by leveraging machine learning techniques to predict the risk of stroke in individuals. Throughout the project lifecycle, various phases, including requirements gathering, design, implementation, testing, and user feedback, were meticulously executed. In this section, we delve into the conclusions drawn from the project, reflect on its strengths and weaknesses, discuss the acquisition of new knowledge and skills, and propose avenues for further work.

### **Conclusions on the Resulting Application/Research:**

The Stroke Prediction application represents a significant milestone in the intersection of healthcare and machine learning. By integrating predictive models with user-friendly interfaces, the application provides individuals with valuable insights into their stroke risk factors. The accuracy of the predictions obtained from the Naive Bayes model demonstrates the potential of data-driven approaches in preventive healthcare. Moreover, the application's ability to handle diverse input data, including demographic information and lifestyle factors, enhances its practical utility.

### **Reflection on Strengths and Weaknesses:**

One of the key strengths of the application lies in its simplicity and accessibility. The streamlined user interface facilitates easy input of patient information and provides prompt risk assessments. Additionally, the integration of scalable machine learning models ensures efficient processing of data, enabling real-time predictions. However, certain limitations exist, primarily in the scope of the predictive models employed. While the Naive Bayes classifier achieves satisfactory performance, exploring more complex algorithms could potentially improve prediction accuracy further.

### **Acquisition of New Knowledge and Skills:**

Throughout the project, valuable knowledge and skills were acquired across multiple domains. The implementation phase deepened understanding of machine learning techniques, particularly in data preprocessing, model training, and deployment. Furthermore, the integration of Streamlit for building interactive web applications introduced new concepts in front-end development and user experience design. Collaboration with healthcare professionals during the requirements gathering phase provided insights into domain-specific challenges and considerations, enriching the overall learning experience.

### **Consideration for Further Work:**

Looking ahead, several avenues for further work emerge from the current project. Firstly, expanding the dataset to include more diverse and comprehensive patient profiles could enhance the robustness of the predictive models. Additionally, integrating feedback mechanisms within the application to capture user experiences and outcomes would enable continuous refinement and improvement. Exploring advanced machine learning techniques, such as ensemble methods or deep learning architectures, could unlock higher predictive performance. Moreover, conducting longitudinal studies to assess the long-term effectiveness of the application in reducing stroke incidence among high-risk individuals would provide valuable insights into its clinical impact.

### **Conclusion:**

In conclusion, the Stroke Prediction application represents a significant achievement in leveraging machine learning for preventive healthcare. By combining predictive analytics with user-centric design principles, the application offers a valuable tool for individuals to assess their stroke risk and take proactive measures. While the current iteration demonstrates promising results, ongoing refinement and innovation are essential to maximize its impact. Through continuous collaboration, learning, and adaptation, the journey towards leveraging technology for better health outcomes continues.

# 9. References

Include a list of cited in your text items (books, papers, websites, etc.). Use Harvard style for the purpose, or any other preferred standard referencing style.

# 10. Bibliography

Include here a list of general reading items (books, papers, websites, etc.). List the items in alphabetical order, using Harvard style to describe them.

# Appendix I

Provide additional material, if appropriate, in separate appendices.

Use one Appendix to provide a link to an on-line video demo of the project.

Do not include the entire code in print as an appendix.