DTSC-691 DATA SCIENCE CAPSTONE PROJECT

TITLE : STROKE RISK PREDICTION

BY JESTIN JOSE

TABLE OF CONTENTS

PROJECT OVERVIEW

Project Objective and Scope

This project focuses on predicting the risk of stroke using health and lifestyle data from a publicly available dataset. By analyzing a variety of health-related factors—such as age, blood pressure, and glucose levels, the goal is to identify individuals who may be more likely to experience a stroke.

Stroke is one of the leading causes of death and long-term disability worldwide. Early identification of individuals at high risk can significantly reduce the impact of strokes through timely intervention and prevention strategies. By using predictive methods, this project aims to provide early warnings to individuals, potentially motivating them to take preventive action. At the same time, it offers healthcare professionals valuable insights to help them better monitor and support those at risk.

Key Objectives

- Support early medical decision-making through accurate and reliable stroke risk predictions.
- Identify the most important health and lifestyle factors that contribute to stroke risk.
- Predict individuals who may be at risk of stroke based on their personal health information, while doing our best to avoid missing potential cases.
- Address the imbalance in the dataset to improve the ability to correctly identify high-risk individuals.

To achieve these goals, we will train and evaluate several machine learning approaches and select the one that performs best in predicting stroke risk.

Scope of the Project

- Use a public dataset to study stroke risk.
- Clean the data and handle missing values.
- Explore key health and lifestyle factors like age, glucose levels, and smoking habits.

- Train and compare several machine learning models to find the best model.
- Deploy the best-performing model in a web application.

What the Project Will Not Cover

- Use real-time or clinical data.
- Connect with electronic health records.
- Predict stroke severity or treatment outcomes.

Data Acquisition

The dataset used in this project was obtained through Google Dataset Search and contains health and demographic information relevant to stroke risk. It includes data for over 5,000 individuals, covering features such as age, gender, hypertension, heart disease, average glucose level, BMI, smoking status, and whether the person has experienced a stroke.

Dataset Link:

https://figshare.com/articles/dataset/_b_Predicting_Stroke_Risk_Dataset_b_/28668398? file=53444630

Reason for Choosing

- Early accessible from a reputable open-source platform.
- The dataset contains relevant features useful for predicting stroke risk.
- Includes health-related and life-style related data.

The dataset adds significant value to the project by offering real-world like data that can be used to simulate early detection methods and support preventive healthcare decisions.

Exploratory Data Analysis

Before training any machine learning models, it was essential to thoroughly explore and understand the dataset.

Data Inspection: We began by loading the dataset and examining its overall structure. This included checking the number of rows and columns, identifying datatypes for each feature and detecting any missing values.

Descriptive Statistics: To better understand the distribution of the data we generated descriptive statistics. This helped us understand the spread, central tendencies, maximum and minimum values within the dataset. During the initial analysis, we observed that the maximum BMI value was 97, which is unusually high and may indicate a potential outlier or data entry error. Additionally, we noticed that the minimum age recorded was 0.08 years (approximately one month old). While this could be valid in cases involving infants, while this could be valid in rare cases involving infants, we considered such data points were not appropriate for stroke risk prediction. So we removed those records.

Class Imbalance Check: The target variable stroke was highly imbalanced, with positive stroke cases representing only a small portion of the dataset. To address this, we used CTGAN (Conditional Tabular Generative Adversarial Network) to generate synthetic data specifically for the positive stroke cases. CTGAN is a deep learning model that learns the underlying patterns of tabular data and generates realistic synthetic samples. By generating additional positive stroke cases, we were able to balance the dataset more effectively, which helps improve the model's ability to recognize and predict stroke risk in minority-class instances.

Data Visualization: To better understand factors influencing stroke risk, three plots were created:

- Generated a Box Plot to visualize the age distribution among individuals who had a stroke and those who did not. We observed that stroke cases were more common among individuals older than 60. Few outliers were found in between 0-1 and at 5-10 ages suggesting rare pediatric stroke cases that may not align with the focus of this prediction task.

- Bar Plot to identify if individuals who smoke or previously smoked had experienced stroke. Identified former smokers are more likely to get a stroke compared to current or non-smokers.

- KDE Plot to visualize average glucose levels among individuals who had a stroke and those who did not. This distribution indicated that both normal and high glucose levels are observed in stroke patients, but higher glucose levels are more prevalent than in non-stroke individuals.

- Correlation heat map to understand the relationship between numerical features and find the strongest correlated features against the target variable. Age showed the moderate positive correlation with stroke, followed by hypertension, heart disease, and average glucose level.

Data Preparation and Cleaning

Based on the findings from the exploratory data analysis phase, we applied the following data preparation and cleaning strategies:

- Individuals with age less than 1 year were removed, as these cases are likely data entry errors or may not be relevant to stroke prediction. Additionally, the age column was originally a float datatype with decimal values, we converted it into an integer datatype to better represent whole years, simplify the analysis, and avoid unrealistic fractional ages that may arise from data inconsistencies.

- We removed the id column as it is not relevant for our predictive analysis.

- We found 201 missing values, all in the BMI column. To address this, we used median imputation, replacing missing BMI values with the median BMI of the dataset. The median is more robust to outliers compared to the mean.

- BMI had a maximum value of 97 which is unusually high and likely data errors or extreme outliers. These unusually high numbers can confuse the model we're building. Instead of removing those records, we chose to limit all BMI values to a maximum of 60. This helps keep the data accurate without letting extreme values affect the results too much.

- Converted categorical features like gender, ever_married, smoking_status, and work_type into numerical format using one hot encoding for model compatibility.

- Applied Standard Scaling to numerical features (e.g., age, BMI, avg_glucose_level) to bring all values into the same range between 0 and 1 which helps models perform better.

Model Training

Training Intent

The primary objective is to predict stroke risk using classification-based machine learning algorithms. The target variable is a binary indicator of stroke occurrence, and the training features include clinical and demographic data such as age, BMI, glucose levels, smoking status, and medical history.

Training Rationale

To establish a strong baseline, we initially use logistic regression, a simple yet effective model for binary classification. To improve prediction performance, we incorporate advanced ensemble models such as Random Forest, XGBoost, AdaBoost, and Gradient Boosting, which are known for capturing nonlinear relationships and feature interactions.

We also develop a classification neural network using TensorFlow and Keras to learn complex patterns that traditional ML models may overlook. This enables us to compare performance metrics like accuracy and recall across different model types.

To further enhance personalized predictions, we build a Graph Neural Network (GNN) using PyTorch. This model identifies patients with similar feature profiles and histories, potentially improving prediction accuracy by leveraging relational data structures.

Training Process Summary

- Data Splitting: The dataset is split into training (80%) and testing (20%) subsets to evaluate model generalizability.
- Feature Preparation: Categorical variables are one-hot encoded; numerical features are scaled; missing values are imputed.
- Model Training:
    - Logistic Regression to establish baseline performance.
    - Ensemble models (Random Forest, XGBoost, AdaBoost, Gradient Boosting).
    - Neural network trained with multiple layers and epochs using TensorFlow/Keras, optimizing for accuracy and recall.
    - Graph Neural Network developed in PyTorch to model patient similarity and improve personalized prediction.

Hyperparameter Tuning

We applied Grid Search with 3-fold cross-validation for traditional ML models and used manual tuning for deep learning models. The table below shows the final hyperparameters used

Model	Key Hyperparameters
Logistic Regression	C=0.01, penalty = l1, solver='liblinear',
Random Forest	n_estimators=100, max_depth=17, min_samples_split=10
XGBoost	n_estimators=300, learning_rate=0.05, max_depth=6, scale_pos weight = 2
AdaBoost	n_estimators=100, learning_rate=[0.01,0.05,0.1],estimator_max_depth=5
Gradient Boosting	n_estimators=100, learning_rate=[0.05,0.1] max_depth=5
Neural Network	epochs=10, batch_size=32, optimizer=Adam, loss=binary_crossentropy

GNN (PyTorch)	2 GraphConv layers, hidden_dim=32, epochs=100

- **Evaluation:** Model performance is evaluated using confusion matrix, AUC-ROC curve, and classification report. Recall is prioritized as the primary metric to minimize false negatives, ensuring that potential stroke cases are less likely to be missed.
- **Final Model Selection:** Selection is based on the best trade-off between accuracy and recall on the testing set. Among the models evaluated, XGBoost model demonstrated the strongest balance of these metrics making it the most suitable choice for stroke risk prediction.

XGBoost Model Parameters

- n_estimators=300: A relatively high number of trees was chosen to allow the model to learn complex patterns without overfitting. This number was

selected after tuning to balance improved model performance with reasonable training time.

- learning_rate=0.05: A low learning rate slows down the learning process, which helps the model converge more smoothly and reduces the risk of overfitting.
- max_depth=6: This controls the maximum depth of each tree. A depth of 6 was chosen to capture sufficient complexity in the data while preventing overly complex trees that might overfit.
- scale_pos_weight=2: This parameter addresses class imbalance by assigning more weight to the minority class (stroke cases). Setting it to 2 helps the model pay more attention to stroke cases, improving recall and reducing false negatives.

Model Evaluation

Model Evaluation

To evaluate the performance of our classification models for stroke prediction, we used a range of metrics appropriate for binary classification, especially in the presence of class imbalance.

Evaluation Metrics Used

- Classification report: Summarizes precision, recall, F1-score and support for each class, providing a detailed breakdown of model performance.
- AUC-ROC Curve: Assesses the model's ability to distinguish between classes across different thresholds.
- Confusion Matrix: Provides a clear view of true positives, true negatives, false positives, and false negatives.

Rationale**:** These metrics collectively provide both quantitative and visual insights into model effectiveness. Prioritizing recall helps ensure stroke cases are detected even if it results in more false positives, aligning with the high stakes of medical diagnosis. This approach minimizes the risk of overlooking true stroke patients, which is critical for timely intervention and improved patient outcomes. Additionally, analyzing the AUC-ROC curve allows us to understand model performance across different thresholds, helping to balance sensitivity and specificity. The confusion matrix further guides model refinement by pinpointing specific types of errors, enabling targeted improvements to reduce misclassification. Together, these evaluation tools offer a comprehensive framework to validate model reliability and clinical applicability.

Each model was evaluated on the same 20% test dataset to ensure consistency and fair comparison of performance across all models.

Model Results and Discussion

To assess model performance, we used a combination of classification metrics:

- Accuracy
- Precision
- Recall
- F1-Score
- AUC-ROC

These metrics were chosen to provide a comprehensive view of model behavior, particularly given the medical nature of the problem where false negatives (missed stroke cases) are more critical than false positives.

Among the evaluation metrics, we focused on finding the best trade-off between accuracy and recall. While recall is critical to ensure that as many stroke cases as possible are identified—minimizing false negatives—accuracy remains important to maintain overall correctness and avoid excessive false positives. Balancing these two metrics helps create a model that is both sensitive to detecting strokes and

reliable in its overall predictions, which is essential for practical deployment in healthcare setting.

Results Summary

Model	Accuracy	Recall	Precision
Logistic Regression	0.82	**0.88**	0.69
Random Forest	0.88	0.80	0.85
XGBoost	**0.87**	**0.84**	**0.80**
Neural Network	0.83	0.78	0.86
AdaBoost	0.88	0.78	0.86
Gradient Boost	0.88	0.80	0.87
GNN (PyTorch)	0.83	**0.85**	0.71

These results indicate that the models can predict stroke risk with strong performance, particularly in detecting stroke cases (high recall).

- XGBoost model demonstrated the strongest overall balance of precision, recall, and F1-score. Its high recall (0.84) ensures effective identification of stroke cases, while maintaining solid precision (0.80) and overall accuracy

(0.87). This balance makes XGBoost the most suitable choice for stroke risk prediction, as it minimizes missed cases without sacrificing too much precision.

- GNN model had a good recall , and accuracy but precision was slightly lower than XGBoost .
- Logistic Regression had good recall and accuracy but it had low precision which can lead to more false alarms.

Final Interpretation

- XGBoost stands out as the most balanced and reliable model overall and would be well-suited for real-world deployment in stroke prediction.

Real-World Application

The trained model can be applied in clinical decision-support systems to flag patients at higher risk of stroke based on their demographic and health profiles. This could help doctors prioritize preventive measures or follow-up investigations.

By minimizing false negatives (via high recall), the model can play a critical role in early intervention, especially in environments where doctors must process large volumes of patient data.

Model Interpretation

To understand how the model makes predictions, we used SHAP (SHapley Additive exPlanations), a model-agnostic interpretability tool that explains the influence of each factor on the prediction.

Key Insights

- Age was the most influential factor in predicting stroke risk, indicating older individuals were more likely to be classified as high risk
- Average glucose level and BMI followed closely in importance
- Lifestyle/health conditions such as hypertension, heart disease, and marital status also increased the predicted risk.
- Work type, gender, smoking status, and residence type had smaller impacts.

The SHAP summary plot's alignment with known stroke risk factors such as age, glucose level, BMI, hypertension, and heart disease supports the model's validity and potential usefulness in clinical settings.

CODE FOR USER INTERFACE

USER INTERFACE APPLICATION CODE

Application Page

Filename : General Projects.py

```
import streamlit as st

import runpy

import os

st.title("General Projects")

project_options = ["Select Project","Stroke Risk Prediction App"]

main_choice = st.selectbox("Choose a project",project_options,index=0)

if main_choice == "Stroke Risk Prediction App":

        choice = st.selectbox("Choose an option:",
["Home","Application","Performance Metrics", "SHAP Modeling"])

        st.markdown("<br>", unsafe_allow_html=True)

        if choice == "Home":

                runpy.run_path(os.path.join("modules", "Home.py"))

        elif choice == "Application":

                runpy.run_path(os.path.join("modules",
"Application.py"))
```

```
        elif choice == "Performance Metrics":
                runpy.run_path(os.path.join("modules",
"performance_metrics.py"))
        elif choice == "SHAP Modeling":
            runpy.run_path(os.path.join("modules",
"shap_modeling.py"))
        else:
               st.error("Page not Found")
```

Filename : Home.py

```
import streamlit as st
st.header("Stroke Risk Prediction")
st.image("images/stroke-img.png",width=550)
# Project Overview
st.markdown("""
## Project Overview
```

```
The purpose of this page is to provide a clear, non-technical overview
of:

- The problem we aim to solve

- The approach we used

- The results we achieved

---

""")

# Problem

st.subheader("The Problem")

st.markdown("""

Stroke is a medical emergency that can lead to permanent disability or
death. Early detection and risk assessment are crucial.

However, many people at risk of stroke are unaware of it until
symptoms appear. This project aims to provide a quick, data-driven
stroke risk prediction app  to raise awareness and encourage proactive
health monitoring.

""")
```

```
# Approach

st.subheader(" The Approach")

st.markdown("""

We collected health-related data , trained and tested 7 different ML models and selected best performing model  to identify stroke risk patterns. The goal was to find a model that does'nt miss any stroke

stroke cases, while also being as accurate and reliable as possible. The data includes:

- Age

- Hypertension and heart disease history

- Average glucose level

- Body Mass Index (BMI)

- Smoking status

- Gender

- Marital Status , type of work and Residence Type

After careful testing we selected **XGBoost model** because it did the best job of detecting stroke risks without making too many false alarms. SHAP (SHapley Additive exPlanations) values were used to explain the model's decisions.

""")
```

```
# Results

st.subheader("Results")

st.markdown("""

The model was evaluated on test data with the following outcomes:

- **Accuracy**: 87%

- **Precision**: 80%

- **Recall**: 84%

These results indicate that the model is effective at identifying

individuals at risk.

""")
```

Filename : Application.py

```
import numpy as np

import joblib

import streamlit as st

import pandas as pd

# Load trained model
```

```
model_pipeline = joblib.load('stroke_model_sklearn.pkl')



# Collect user input

gender = st.selectbox("Gender", ['Male', 'Female', 'Other'])

age = st.slider("Age", 0, 100, 30)

hypertension = st.selectbox("Hypertension", [0, 1])

heart_disease = st.selectbox("Heart Disease", [0, 1])

ever_married = st.selectbox("Ever Married", ['Yes', 'No'])

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

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

avg_glucose_level = st.slider("Average Glucose Level", 50.0, 300.0,
100.0)

bmi = st.slider("BMI", 10.0, 60.0, 22.0)

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



user_input_df = pd.DataFrame([{

        'gender': gender,
```

```
        'age': age,
        'hypertension': hypertension,
        'heart_disease': heart_disease,
        'ever_married': ever_married,
        'work_type': work_type,
        'Residence_type': residence_type,
        'avg_glucose_level': avg_glucose_level,
        'bmi': bmi,
        'smoking_status': smoking_status
    }])

if st.button("Predict Stroke Risk"):
        prediction = model_pipeline.predict(user_input_df)[0]

        if prediction == 1:
            st.error("High Risk of Stroke")  # red colored message
        else:
            st.success("Low Risk of Stroke")  # green colored message
```

Filename : performance_metrics.py

```
import streamlit as st

from PIL import Image



st.title("Model Performance")

metrics_name = st.selectbox(

    "Select a performance metric to view:",

    ("ROC-AUC Curve", "Confusion Matrix")

)

image_paths = {

    "ROC-AUC Curve": "images/ROC-AUC.png",

    "Confusion Matrix": "images/ConfusionMatrix.png"

}

if metrics_name in image_paths:

        image = Image.open(image_paths[metrics_name])

        st.image(image,caption=f"{metrics_name}")
```

File_name : shap_modeling.py

```
import streamlit as st

from PIL import Image

import os



 #Title

st.title("SHAP Summary Plot")

image_path = "images/Shap.png"

image = Image.open(image_path)

st.image(image,width=550)



st.markdown("""

            The SHAP plot reveals that age, average glucose level, BMI, hypertension, and heart disease are the top contributors to the model’s stroke predictions.

            Higher values of these features generally increase predicted stroke risk.
```

Importantly, these findings align closely with established medical research, which identifies these factors as major stroke risk determinants.

This overlap between model insights and clinical evidence increases confidence in the model’s relevance for real-world health risk assessment

""")

Biographical Data and Resume Page

File_name : Resume.py

```
import streamlit as st
from PIL import Image, ImageDraw

def make_circle_image(image_path):
    img = Image.open(image_path).convert("RGBA")
    size = min(img.size)
    mask = Image.new('L', (size, size), 0)
    draw = ImageDraw.Draw(mask)
    draw.ellipse((0, 0, size, size), fill=255)

    output = Image.new('RGBA', (size, size), (0, 0, 0, 0))
    output.paste(img, ((size - img.width) // 2, (size - img.height) //
2))
    output.putalpha(mask)
```

```
    return output


circle_img = make_circle_image("images/MyPhoto.png")

st.image(circle_img,width=200)


st.markdown("""

# Jestin Jose


*Chicago, IL | 224-258-8087*

[jestin.joseocto896@gmail.com](mailto:jestin.joseocto896@gmail.com)

[LinkedIn](https://www.linkedin.com/in/jestin-jose-b00ba2193) |

[GitHub](https://github.com/jestinjose90)


---



## Summary

Highly motivated and detail-oriented data science enthusiast with a

strong academic background in Data Science and hands-on experience in
```

data analysis and machine learning. Proficient in Python, R, and data visualization tools. Eager to leverage my knowledge and skills to contribute to the success of a data-driven organization as a Junior Data Scientist. A quick learner with a passion for solving complex problems using data-driven insights.

Education

- **Master of Science (B.S.) in Data Science**, Eastern University, St Davids, PA
  - GPA: 3.63
- **IBM Data Science Professional Certificate** (Coursera.Inc)
- **Bachelor of Science (B.S.) in Computer Science**, Northeastern Illinois University, Chicago, IL
  - GPA: 3.83
  - Graduated with Magna Cum Laude Honors

Relevant Experience

Capstone Project: SpaceX Landing Outcome Prediction

Mar 2023 - Aug 2023

Tools: Python, NumPy, pandas, Scikit-learn, Seaborn, Folium, SQLite

- Gathered historical data on SpaceX Falcon 9 launches including mission parameters, environmental conditions, and landing outcomes.

- Performed exploratory data analysis to identify patterns influencing landing success.

- Conducted data preprocessing including missing value handling and feature engineering.

- Built classification models using Linear Regression, SVM, Decision Tree Classifier to predict landing outcomes.

- Used cross-validation and hyperparameter tuning for optimal performance.

- Visualized model performance using confusion matrix and accuracy metrics.

- Achieved 88% accuracy on test data.

- Identified key factors impacting SpaceX landing success and failure.

House Price Prediction Project

May 2023 - July 2023

Tools: Python, NumPy, pandas, Scikit-learn, Seaborn

- Used real estate sales dataset to predict house prices with machine learning.

- Cleaned data and handled missing values.

- Explored dataset through visualization for insights on price distribution and feature correlations.

- Developed and evaluated Simple Linear Regression model using R-squared metric.

- Improved model performance with regularization techniques.

Technical Skills

- **Programming Languages:** Python, R, Java

- **Data Analysis:** Pandas, NumPy, SciPy

- **Data Cleaning & Preprocessing**

- **Data Visualization:** Matplotlib, Seaborn, Dash, Tableau

- **Machine Learning Libraries:** Scikit-learn

- **Statistical Analysis & Predictive Modeling**

- **Database Management:** SQL, SQLite

- **Tools:** Jupyter Notebook, Visual Studio Code, Microsoft Excel, PowerPoint

- **Big Data:** PySpark (Basic)

- **Web Scraping:** BeautifulSoup4

- **Cloud Computing:** AWS (Basic)

Key Attributes

- Strong problem-solving and analytical skills

- Excellent communication and teamwork abilities

- Enthusiastic learner, actively engaged in online data science communities and courses

""")

Filename: Homepage.py

```
import streamlit as st


st.title("About Me")

st.write("""


Hello! I’m Jestin Jose, currently pursuing my Master’s in Data Science. I completed my Bachelor’s degree in Computer Science, where I developed a strong foundation in programming, algorithms, and data analysis.


My passion lies in turning complex data into actionable insights using machine learning and statistical modeling. I am eager to build a career as a data scientist, where I can leverage my skills to solve real-world problems.


Academic Background

MS in Data Science (Ongoing) – Focus on machine learning, data visualization and statistical modeling
```

BS in Computer Science – Solid understanding of software development, databases, and computational theory.

Career Aspirations

I aspire to work as a data scientist in industries such as healthcare or finance, where data-driven decisions can significantly improve outcomes.

Professional Interests

Machine Learning & Deep Learning

Data Visualization & Communication

Predictive Analytics & Statistical Modeling

Personal Interests

Outside of academics, I enjoy traveling, playing sudoku ,chess, and experimenting with new programming languages. I believe that creativity and curiosity are essential to continuous learning, both in tech and life.

""")

MODEL DEVELOPMENT AND IMPLEMENTATION

DTSC - 691 Capstone Project - Stroke Risk Prediction

Importing the Dataset

```
#import libraries
import numpy as np
import pandas as pd

#import the dataset
health_data = pd.read_csv("StrockDataset.csv")
```

Exploratory Data Analysis

Understanding the Data

```
#Summary of the data
health_data.info()

#Display first 10 rows of the data
health_data.head(10)
```

Class Distribution of Categorical Variables

```
categorical_cols = ['gender','ever_married','work_type','Residence_type','smo
king_status']

for col in categorical_cols:
    print(f"\n{col} distribution:")
    print(health_data[col].value_counts())
```

Descriptive Statistics

```
#Generate summary statistics of the data
health_data.describe()
```

By observing the summary statistics, I found that the minimum age is 0.08, which is not a realistic value. Similarly, the maximum BMI is 97, which is unusually high and should be addressed.

Data Cleaning

```
#Identify the missing values in each column
health_data.isnull().sum()
```

I have 201 missing values for BMI . We would handle the missing values by median imputation that would fill in the missing values with the median of BMI

```
#Find the median of BMI
health_data['bmi'].median()
```

I use scikit-learn tool Simple Imputer to fill in missing values with the median 28.1.

```
#Import sci-kit learn  and Simple Imputer
from sklearn.impute import SimpleImputer

#Imputers
bmi_imputer = SimpleImputer(strategy='median')

#Apply Imputator to fill in the missing values with median of BMI
health_data['bmi'] = bmi_imputer.fit_transform(health_data[['bmi']])
```

Verify that the dataset contains no missing values

```
health_data.isnull().sum()
```

Previously, in our summary statistics, we observed that BMI had a maximum value of 97 and age had a minimum value of 0.08, both of which are unrealistic. I need to address these discrepancies. Values above 60 are very rare and likely data errors or extreme outliers. So capping BMI at 60 is a reasonable data cleaning choice to handle extreme values, even if it's not a "healthy" or "approved" medical cutoff therefore I will cap the maximum BMI value at 60 by replacing all BMI values greater than 60 with 60.

```
#Modify the BMI values by giving 60 as maximum value
health_data.loc[health_data['bmi'] > 60,'bmi'] = 60
```

I will remove rows where the age is less than 1. Since the age column is currently of float datatype, I will convert it to an integer type.

```
#remove age below 1
health_data = health_data[health_data['age']>= 1]

#Convert age into integer type
health_data['age'] = health_data['age'].round().astype(int)
```

I will also remove the id column because it will not be relevant for my analysis

```
#drop id column
health_data = health_data.drop('id',axis=1)
```

Let's observe summary statistics again after cleaning. I can see that we dont have id column anymore , minimum age is 1, and the maximum BMI has been capped at 60.

```
health_data.describe()
```

Visualization

Box Plot of Age Grouped by Stroke Occurence

```
# import libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Generate a box plot to visualize age distribution among individuals who had
stroke and who didnt
```

```
sns.boxplot(x='stroke', y='age', data=health_data)
plt.title('Age Grouped by Stroke Occurence')
plt.xlabel('Stroke (0 = No, 1 = Yes)')
plt.ylabel('Age')
plt.show()
```

The Above plot interprets that those with stroke , ages are generally higher , mostly concentrated between approximately 60 and 80 years, indicating that stroke patients tend to be older

Stacked Bar Chart

```
#Filter out unknown category
filtered_health_data = health_data[health_data['smoking_status'] != 'Unknown'
]

# Calculate proportions
smoking_stroke = pd.crosstab(filtered_health_data['smoking_status'], filtered
_health_data['stroke'], normalize='index')

# Generate a Stacked Bar chart reperesenting Smoking Status based on individu
als who had stroke and not
ax = smoking_stroke.plot(kind='bar', stacked=True, colormap='coolwarm', figsi
ze=(8, 5))

# Add percentage labels
for p in ax.patches:
    height = p.get_height()
    if height > 0:
        ax.text(p.get_x() + p.get_width()/2,
                p.get_y() + height/2,
                f'{height*100:.1f}%',
                ha='center', va='center', fontsize=15, color='black')

# Add Labels and title to the plot
plt.title('Proportion of Stroke Occurence within Smoking Status Groups')
plt.xlabel('Smoking Status')
plt.ylabel('Proportion')
plt.legend(title='Stroke', labels=['No', 'Yes'])
plt.xticks(rotation=15)
plt.tight_layout()
plt.show()
```

Due to highly imbalanced stroke class we receive a higher proportion of non-stroke cases compared to stroke. From the plot i found that people who are formerly smoked have more chance of experiencing a stroke followed by current smokers and those never smoked.

KDE Plot

```
#Generate a KDE plot to visualize average glucose levels among individuals wh
o had stroke and those who did not
```

```
sns.kdeplot(health_data[health_data['stroke'] == 0]['avg_glucose_level'], lab
el='No Stroke', fill=True)
sns.kdeplot(health_data[health_data['stroke'] == 1]['avg_glucose_level'], lab
el='Stroke', fill=True)
plt.title('Glucose Level Distribution Among Stroke and Non-Stroke Individuals
')
plt.xlabel('Average Glucose Level')
plt.ylabel('Density')
plt.legend()
plt.show()
```

The plot compares the distribution of average glucose levels between individuals who had a stroke (orange) and those who did not (blue). We see a higher and narrow peak around 100 mg/dL which indicates large number of people are under normal range. A secondary peak around 200 mg/dL, suggesting a notable subgroup with significantly elevated glucose levels, possibly linked to diabetes or hyperglycemia. This distribution indicates that both normal and high glucose levels are observed in stroke patients, but higher glucose levels are more prevalent among them than in non-stroke individuals.

Correlation HeatMap

```
# identify the metrics of features that are related to stroke
corr = health_data.corr(numeric_only=True)

#Generate a correlation heatmap
plt.figure(figsize=(10,8))
sns.heatmap(corr,annot=True,cmap='coolwarm',fmt='.2f',linewidths=0.5)
plt.title("Correlation Heatmap")
plt.show()
```

Age has a correlation of 0.25 with stroke, indicating a moderate positive relationship. This means that as age increases, the likelihood of having a stroke tends to increase slightly.

Balance the Stroke Class

Lets check the class distribution of the Stroke column which is our target variable

```
#class distribution of stroke column
health_data['stroke'].value_counts()
```

I observes a high class imbalance in stroke cases, with the majority being non-stroke. This imbalance can lead to poor model performance, so it is important to apply techniques to balance the data

To Address the stroke class imbalance we use CTGAN(Conditional Tabular Generative Adversarial Network) a deep learning model that learns underlying distributions of data and generates synthetic samples that mimics the original data

```
import sdv
from sdv.metadata import Metadata
```

```
import pandas as pd
from sdv.single_table import CTGANSynthesizer
import random
import torch

# Set seeds for reproducibility . Usually all random generators are called.
SEED = 42
# for Python in-built random-generators
random.seed(SEED)
# sets the seed for numpy and pandas random generator ,
np.random.seed(SEED)
# CTGAN primarily uses pytorch random generator because it is built on top of
pytorch
torch.manual_seed(SEED)

#Create a copy of health_data based on positive stroke cases
stroke_positive = health_data[health_data['stroke'] == 1].copy()

#Figures out the type of each column (e.g., categorical, numerical, boolean)
in our dataset
metadata = Metadata.detect_from_dataframe(data=stroke_positive)

synthesizer = CTGANSynthesizer(metadata)
synthesizer.fit(stroke_positive)

# Generate synthetic samples
synthetic_data = synthesizer.sample(num_rows=2409) # 50% of non-stroke cases
synthetic_data['stroke'] = 1  # Add target label back

# Combine with original dataset
health_data_balanced = pd.concat([health_data, synthetic_data], ignore_index=
True)
```

The model may take some time to run based on the OS. Verify the stroke class again

```
health_data_balanced['stroke'].value_counts()
```

Model Training

Split our balanced dataset for training and testing

```
from sklearn.model_selection import train_test_split

features = health_data_balanced.drop('stroke',axis=1)
target = health_data_balanced['stroke']

x_train_val,x_test,y_train_val,y_test = train_test_split(features,target,test
_size=0.2,random_state=42,stratify=target)
```

Since our hypertension and heart_disease columns have only values between 0 and 1 we dont need to standardize the data .Encode categorical features using one-hot encoding and standardize numerical features using StandardScaler for model training

```
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer

#create a variable called numerical_cols to extract numerical columns from th
e data
numerical_cols = ['age','avg_glucose_level','bmi']
#create a variable called numerical_cols to extract columns with binary value
s  from the data
binary_cols = ['hypertension','heart_disease']

preprocessor = ColumnTransformer(
    transformers = [
        ('cat',OneHotEncoder(handle_unknown='ignore'),categorical_cols),
        ('num',StandardScaler(),numerical_cols),
        ('bin','passthrough',binary_cols)

    ]
)


#Fit and tranform on training data and transform only on testing data
x_train_encoded = preprocessor.fit_transform(x_train_val)
x_test_encoded = preprocessor.transform(x_test)
```

I will be training and evaluating 7 different models and find the best performing model that would predict stroke risk . A good recall ,accuracy and better precision would be focus in our model evaluation.

ML Model Training and Evaluation

Random Forest : Training the Model

```
from sklearn.ensemble import RandomForestClassifier

random_f = RandomForestClassifier(n_estimators=100,random_state=42)
```

Hyperparameter Tuning using GridSearchCV

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import  make_scorer,recall_score

recall_scorer= make_scorer(recall_score,average='binary')

rf_param_grid = {
    'n_estimators': [100],
    'max_depth': [17],
    'min_samples_split': [10],
```

```
}


rf_grid_search = GridSearchCV(
    estimator=random_f,
    param_grid=rf_param_grid,
    scoring=recall_scorer,
    cv=3,
    n_jobs=-1,
    verbose=2
)

rf_model = rf_grid_search.fit(x_train_encoded,y_train_val)

print("Best parameters:", rf_grid_search.best_params_)
print("Best recall score:", rf_grid_search.best_score_)

rf_best_model = rf_grid_search.best_estimator_
```

Random Forest: Testing the Model

```
y_rand_proba = rf_best_model.predict_proba(x_test_encoded)[:, 1]
y_rand_custom = (y_rand_proba >= 0.5).astype(int)
y_rand_custom
```

Random Forest : Model Evaluation Metrics

Confusion Matrix

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
cm = confusion_matrix(y_test, y_rand_custom)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Random Forest - Confusion Matrix")
plt.show()
```

Classification Report

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_rand_custom))
```

ROC-AUC Curve

```
from sklearn.metrics import roc_curve,auc
#Calculate ROC curve and AUC
fpr, tpr , thresholds = roc_curve(y_test,y_rand_custom)
roc_auc = auc(fpr,tpr)

#plot ROC-AUC curve
plt.figure(figsize=(8,6))
plt.plot(fpr,tpr,label=f'ROC curve(AUC= {roc_auc:.2f})',linewidth=2)
```

```
plt.plot([0, 1], [0, 1], 'k--', label='Chance')  # Diagonal line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

Gradient Boost: Training the Model

```
from sklearn.ensemble import GradientBoostingClassifier

gboost_model = GradientBoostingClassifier(random_state=42)
```

Hyperparameter Tuning using GridSearchCV

```
from sklearn.model_selection import GridSearchCV


gb_param_grid = {
    'n_estimators': [100],
    'learning_rate': [0.05,0.1],
    'max_depth': [5],
}



gb_search = GridSearchCV(
    estimator=gboost_model,
    param_grid= gb_param_grid,
    scoring=recall_scorer,
    cv=3,
    n_jobs=-1,
    verbose=2,

)

gb_model = gb_search.fit(x_train_encoded,y_train_val)

print("Best parameters:",gb_search.best_params_)
print("Best recall score:", gb_search.best_score_)

gb_best_model = gb_search.best_estimator_
```

Testing the Model

```
y_gb_proba = gb_best_model.predict_proba(x_test_encoded)[:, 1]
y_gb_custom = (y_gb_proba >= 0.5).astype(int)
y_gb_custom
```

Classification Report

```
print(classification_report(y_test, y_gb_custom))
```

Confusion Matrix

```
cm = confusion_matrix(y_test, y_gb_custom)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Gradient Boost - Confusion Matrix")
plt.show()
```

ROC-AUC Curve

```
#Calculate ROC curve and AUC
fpr, tpr , thresholds = roc_curve(y_test,y_gb_custom)
roc_auc = auc(fpr,tpr)

#plot ROC-AUC curve
plt.figure(figsize=(8,6))
plt.plot(fpr,tpr,label=f'ROC curve(AUC= {roc_auc:.2f})',linewidth=2)
plt.plot([0, 1], [0, 1], 'k--', label='Chance')  # Diagonal line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

Ada Boost : Training the Model

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

# Define base estimator (optional)
base_estimator = DecisionTreeClassifier(max_depth=1)

# Define AdaBoost model
adaboost_model = AdaBoostClassifier(estimator=base_estimator,random_state=42)
```

Hyperparameter Tuning using GridSearchCV

```
# Define hyperparameter grid for AdaBoost
ada_param_grid = {
    'n_estimators': [100],
    'learning_rate': [0.01,0.05, 0.1],
    'estimator__max_depth': [5]  # tuning the base learner
}
```

```
# Grid Search
ada_search = GridSearchCV(
    estimator=adaboost_model,
    param_grid=ada_param_grid,
    scoring=recall_scorer,
    cv=3,
    n_jobs=-1,
    verbose=2
)

# Fit model
ada_model = ada_search.fit(x_train_encoded, y_train_val)

# Results
print("Best parameters:", ada_search.best_params_)
print("Best Recall score:", ada_search.best_score_)

# Best model
ada_best_model = ada_search.best_estimator_
```

Testing the Model

```
y_ab_proba = ada_best_model.predict_proba(x_test_encoded)[:, 1]
y_ab_custom = (y_ab_proba >= 0.5).astype(int)
y_ab_custom
```

Confusion Matrix

```
cm = confusion_matrix(y_test, y_ab_custom)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("AdaBoost - Confusion Matrix")
plt.show()
```

Classification Report

```
print(classification_report(y_test, y_ab_custom))
```

ROC-AUC Curve

```
#Calculate ROC curve and AUC
fpr, tpr , thresholds = roc_curve(y_test,y_ab_custom)
roc_auc = auc(fpr,tpr)

#plot ROC-AUC curve
plt.figure(figsize=(8,6))
plt.plot(fpr,tpr,label=f'ROC curve(AUC= {roc_auc:.2f})',linewidth=2)
plt.plot([0, 1], [0, 1], 'k--', label='Chance')  # Diagonal line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

Classification Neural Network Model

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.metrics import classification_report,confusion_matrix

#Define the model
NN_model = Sequential([
    Dense(100,activation='relu',input_shape=(x_train_encoded.shape[1],)),
    Dropout(0.3),
    Dense(60,activation='relu'),
    Dropout(0.3),
    Dense(60,activation='relu'),
    Dropout(0.3),
    Dense(1,activation='sigmoid')
])

NN_model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accura
cy'])

hist = NN_model.fit(x_train_encoded,y_train_val,epochs=10,batch_size=32)
```

Testing the model

```
y_pred_prob = NN_model.predict(x_test_encoded)
neural_predict = (y_pred_prob >= 0.5).astype(int)
```

Classification Report

```
print(classification_report(y_test, neural_predict))
```

Confusion Matrix

```
cm = confusion_matrix(y_test, neural_predict)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("NeuralNetwork - Confusion Matrix")
plt.show()
```

ROC-AUC Curve

```
#Calculate ROC curve and AUC
fpr, tpr , thresholds = roc_curve(y_test,neural_predict)
roc_auc = auc(fpr,tpr)
```

```
#plot ROC-AUC curve
plt.figure(figsize=(8,6))
plt.plot(fpr,tpr,label=f'ROC curve(AUC= {roc_auc:.2f})',linewidth=2)
plt.plot([0, 1], [0, 1], 'k--', label='Chance')  # Diagonal line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

Logistic Regression : Training the Model

```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression(max_iter=1000,class_weight='balanced',random_sta
te=42)
```

Hyperparameter Tuning using GridSearchCV

```
from sklearn.model_selection import GridSearchCV


param_grid = {
    'C': [0.01],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear']
}



grid_search = GridSearchCV(
    estimator=log_reg,
    param_grid=param_grid,
    scoring=recall_scorer,
    cv=3,
    n_jobs=-1,
    verbose=2

)

lg_model = grid_search.fit(x_train_encoded,y_train_val)

print("Best parameters:", grid_search.best_params_)
print("Best Recall score:", grid_search.best_score_)

best_model = grid_search.best_estimator_
```

Testing the Model

```
y_proba = best_model.predict_proba(x_test_encoded)[:, 1]
y_pred_custom = (y_proba >= 0.5).astype(int)
y_pred_custom
```

Logistic Regression: Model Evaluation Metrics

Confusion Matrix

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

# Plot the confusion matrix
cm = confusion_matrix(y_test, y_pred_custom)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Logistic Regression- Confusion Matrix")
plt.show()
```

Classification Report including Accuracy ,Precision , Recall and F1 Score of the Model

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred_custom))
```

ROC-AUC Curve

```
from sklearn.metrics import roc_curve,auc

#Calculate ROC curve and AUC
fpr, tpr , thresholds = roc_curve(y_test,y_pred_custom)
roc_auc = auc(fpr,tpr)

#plot ROC-AUC curve
plt.figure(figsize=(8,6))
plt.plot(fpr,tpr,label=f'ROC curve(AUC= {roc_auc:.2f})',linewidth=2)
plt.plot([0, 1], [0, 1], 'k--', label='Chance')  # Diagonal line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

Graph Neural Network Model

```
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import NearestNeighbors
import torch
from torch_geometric.data import Data
```

```
feature_cols = ['age','hypertension','heart_disease','avg_glucose_level']
graph_features = health_data_balanced[feature_cols].values

scaler = StandardScaler()
graph_features = scaler.fit_transform(graph_features)

target_var = health_data_balanced['stroke'].values

x = torch.tensor(graph_features,dtype=torch.float)
y = torch.tensor(target_var,dtype=torch.long)

#Create edges using KNN
k = 3
nbrs = NearestNeighbors(n_neighbors=k+1).fit(graph_features)
_, indices = nbrs.kneighbors(graph_features)


#Build edge list
edge_sources = []
edge_targets = []


for i, neighbors in enumerate(indices):
    for j in neighbors[1:]:
        edge_sources.append(i)
        edge_targets.append(j)

edge_index = torch.tensor([edge_sources,edge_targets],dtype=torch.long)

print(edge_index.shape)
```

Training the Model

```
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.nn import GCNConv

class StrokeGNN(nn.Module):
    def __init__(self, input_dim, hidden_dim=32):
        super(StrokeGNN,self).__init__()
        self.conv1 = GCNConv(input_dim,hidden_dim)
        self.conv2 = GCNConv(hidden_dim,hidden_dim)
        self.classifier = nn.Linear(hidden_dim,2)

    def forward(self,data):
        x,edge_index = data.x,data.edge_index

        x = self.conv1(x,edge_index)
        x = F.relu(x)
        x = self.conv2(x,edge_index)
```

```
        x= F.relu(x)

        return self.classifier(x)

from torch_geometric.utils import train_test_split_edges

data = Data(x=x,edge_index=edge_index,y=y)

# Split manually using sklearn
train_mask, test_mask = train_test_split(torch.arange(len(y)), test_size=0.2,
stratify=y)

data.train_mask = torch.zeros(data.num_nodes, dtype=torch.bool)
data.test_mask = torch.zeros(data.num_nodes, dtype=torch.bool)

data.train_mask[train_mask] = True
data.test_mask[test_mask] = True

from sklearn.utils.class_weight import compute_class_weight
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
gnn_model = StrokeGNN(input_dim=x.size(1)).to(device)
data = data.to(device)

optimizer = torch.optim.Adam(gnn_model.parameters(), lr=0.01)

# Compute weights based on training labels only
# Extract training labels
train_labels = data.y[data.train_mask].cpu().numpy()

# Convert classes to NumPy array
classes = np.array([0, 1])

# Compute weights
class_weights = compute_class_weight(class_weight='balanced', classes=classes
, y=train_labels)

# Convert to tensor and move to device
weights = torch.tensor(class_weights, dtype=torch.float).to(device)

# Define weighted loss
criterion = nn.CrossEntropyLoss(weight=weights)

def train():
    gnn_model.train()
    optimizer.zero_grad()
    out = gnn_model(data)
    loss = criterion(out[data.train_mask], data.y[data.train_mask])
    loss.backward()
    optimizer.step()
    return loss.item()
```

```
def test():
    gnn_model.eval()
    out = gnn_model(data)
    pred = out.argmax(dim=1)

    correct = pred[data.test_mask] == data.y[data.test_mask]
    acc = int(correct.sum()) / int(data.test_mask.sum())
    return acc

for epoch in range(1, 101):
    loss = train()
    acc = test()
    if epoch % 10 == 0:
        print(f"Epoch {epoch:3d} | Loss: {loss:.4f} | Test Acc: {acc:.4f}")
```

Testing the model

```
gnn_model.eval()

# Forward pass
out = gnn_model(data)

# Apply softmax to get probabilities
probs = torch.softmax(out, dim=1)

# Predicted class labels
preds = probs.argmax(dim=1)

# Filter for test nodes
y_true = data.y[data.test_mask].cpu().numpy()
y_pred_gnn = preds[data.test_mask].cpu().numpy()

y_pred_gnn
```

Classification Report

```
print(classification_report(y_true, y_pred_gnn))
```

Confusion Matrix

```
cm = confusion_matrix(y_true, y_pred_gnn)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Graph Neural Network - Confusion Matrix")
plt.show()
```

ROC-AUC Curve

```
#Calculate ROC curve and AUC
fpr, tpr , thresholds = roc_curve(y_true,y_pred_gnn)
```

```
roc_auc = auc(fpr,tpr)

#plot ROC-AUC curve
plt.figure(figsize=(8,6))
plt.plot(fpr,tpr,label=f'ROC curve(AUC= {roc_auc:.2f})',linewidth=2)
plt.plot([0, 1], [0, 1], 'k--', label='Chance')  # Diagonal line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

XGBoost: Training the Model

```
import xgboost as xgb
xgb_model = xgb.XGBClassifier( eval_metric='aucpr', random_state=42)
```

Hyperparameter Tuning using GridSearchCV

```
xgb_param_grid = {
    'max_depth': [6],
    'learning_rate': [0.05],
    'n_estimators': [300],
    'scale_pos_weight': [2]
}

xgb_grid_search = GridSearchCV(estimator=xgb_model, param_grid=xgb_param_grid
, scoring='recall', cv=3, verbose=2)
xgb_grid_search.fit(x_train_encoded, y_train_val)

print("Best params:", xgb_grid_search.best_params_)
print("Best recall:", xgb_grid_search.best_score_)

xgb_best_model = xgb_grid_search.best_estimator_
```

Testing the Model

```
xgb_proba = xgb_best_model.predict_proba(x_test_encoded)[:, 1]
xgb_custom = (xgb_proba >= 0.5).astype(int)
xgb_custom
```

Confusion Matrix

```
cm = confusion_matrix(y_test, xgb_custom)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("XGBoost - Confusion Matrix")
plt.show()
```

Classification Report

```
print(classification_report(y_test, xgb_custom))
```

ROC-AUC Curve

```
#Calculate ROC curve and AUC
fpr, tpr , thresholds = roc_curve(y_test,xgb_custom)
roc_auc = auc(fpr,tpr)

#plot ROC-AUC curve
plt.figure(figsize=(8,6))
plt.plot(fpr,tpr,label=f'ROC curve(AUC= {roc_auc:.2f})',linewidth=2)
plt.plot([0, 1], [0, 1], 'k--', label='Chance')  # Diagonal line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

SHAP

```
import shap
# Get the fitted OneHotEncoder and StandardScaler
ohe = preprocessor.named_transformers_['cat']
scaler = preprocessor.named_transformers_['num']

# Get encoded categorical feature names
encoded_cat_features = ohe.get_feature_names_out(categorical_cols)

# Combine with numerical column names
all_feature_names = np.concatenate([encoded_cat_features, numerical_cols])

passthrough_cols = [col for col in x_train_val.columns if col not in categori
cal_cols + numerical_cols]
all_feature_names = np.concatenate([encoded_cat_features, numerical_cols, pas
sthrough_cols])

explainer = shap.Explainer(xgb_best_model,x_test_encoded)
shap_values = explainer(x_test_encoded,check_additivity=False)

shap.summary_plot(shap_values, x_test_encoded,feature_names=all_feature_names
)
```

To deploy the model to streamlit , i will use a model pipeline that would preprocess data and fit the best model

```
from sklearn.pipeline import Pipeline
model_pipeline = Pipeline([
```

```
    ('preprocessor',preprocessor),
    ('classifier',xgb_best_model)
])

model_pipeline.fit(x_train_val,y_train_val)

print(model_pipeline)
```

Deployment

```
#for scikitlearn
import joblib
joblib.dump(model_pipeline,"stroke_model_sklearn.pkl")
```