University of Surrey

Computer Science and Electrical Engineering

Practical Business Analytics Project

(COMM053)

Group Name: SEP G05
Group Members:
Ximena Michelle Castro Villalobos [6515820]
Syeda Fizza Raza [6509217]
Ashwini Reddy Cheekala [6886963]
Sampada Shrestha [6892246]
Module Leader: Alaa Marshan

Table of Contents

1. Introduction *3*

1.1 Problem Statement 3

1.2 Solution or Objectives 4

1.3 Dataset Overview 5

1.4 CRISP-DM Approach 7

1.5 Visualizations and Graphical analysis before data preprocessing 10

1.6 Dataset after preprocessing 13

1.7 Models (Techniques we are using) 14

2. Data Understanding, Pre-processing and Exploration *18*

2.1 Cleaning the data 18

2.2 Feature Engineering, normalization and data split 18

2.3 Data Visualization after preprocessing 21

3. Modeling *23*

3.1 Sampada Shrestha [6892246] 23

3.2 Syeda Fizza Raza [6509217] 32

3.3 Ximena Michelle Castro Villalobos [6515820] 48

3.4 C Ashwini Reddy [6886963] 51

4. Performance Evaluation *60*

4.1 Sampada Shrestha [6892246] 60

4.2 Syeda Fizza Raza [6509217] 67

4.3 Ximena Michelle Castro Villalobos [6515820] 86

4.4 C Ashwini Reddy [6886963] 95

5. Discussion, Conclusion, and Recommendations *101*

5.1 Discussions and Comparisons 101

5.2 Conclusion and Recommendation 104

6. Author Contribution Statement *105*

7. References *106*

8. Appendix *107*

1. Introduction

1.1 Problem Statement

Strokes are a leading cause of disability worldwide and rank as the second leading cause of death globally, as reported by the World Health Organization (Organization, 2022). These staggering statistics underline the urgent need for effective prevention and intervention strategies to combat the devastating impact of strokes on individuals, families, and healthcare systems (Organization, 2022). Despite advancements in medical science, strokes continue to pose a significant threat to public health, often resulting in long-term disabilities, reduced quality of life, and a substantial economic burden (Feigin, et al., 2021).

One of the critical challenges in addressing this issue is the lack of timely preventative care for individuals in high-risk groups. Many people, especially those in underserved or low-resource settings, remain unaware of their vulnerability to stroke, resulting in missed opportunities for early intervention (Benjamin, et al., 2019). High-risk groups often include individuals with hypertension, obesity, diabetes, high cholesterol levels, and unhealthy lifestyle habits, such as smoking or physical inactivity. These factors, if not managed proactively, can significantly increase the likelihood of experiencing a stroke (Anon., 2018)

The ability to accurately predict the likelihood of a stroke based on a combination of demographic, medical, and lifestyle factors is crucial for saving lives. Early prediction can empower healthcare providers to implement tailored interventions, such as medication, lifestyle modifications, and regular monitoring, to mitigate risk (Sacco, et al., 2013).

Furthermore, predictive tools can educate individuals about their health risks and encourage proactive management of modifiable factors, reducing the overall incidence and severity of strokes (Ovbiagele, et al., 2015)

Innovative solutions that leverage data analytics, artificial intelligence, and personalized healthcare can play a transformative role in addressing this pressing global health issue. By integrating predictive models into routine care, healthcare systems can shift from reactive treatment to preventative care, ultimately improving patient outcomes and reducing the burden of strokes worldwide (Topol, 2019)

Importance of predicting stroke

The goal of this research is to use data to predict the likelihood of a stroke. There are different reasons for the importance of this from medical to financial ones:

Health outcomes: Because of high Mortality and disability Rates resulting from heart strokes. Survivors often face long-term physical, cognitive, and emotional impairments.

Strokes are among the most severe medical conditions, with high mortality and disability rates making them a global health priority. Each year, millions of people experience strokes, and for those who survive, the consequences can be life-altering. Survivors often face long-term physical impairments, such as paralysis or difficulty with motor skills, which can limit their independence and quality of life. Additionally, cognitive impairments like memory loss, difficulties in problem-solving, and speech disorders are common. Emotional challenges, including depression and anxiety, further compound the burden on survivors (Feigin, et al., 2021). The ripple effect of these outcomes extends to families, caregivers, and communities, underscoring the importance of predictive measures that can help mitigate these devastating impacts.

Financial burden: Strokes lead to high medical costs, loss of productivity, and the emotional toll on families and caregivers. The economic implications of strokes are enormous, affecting both individuals and healthcare systems. Medical costs associated with stroke treatment, rehabilitation, and long-term care can be substantial, often straining the financial resources of patients and their families. Furthermore, strokes frequently result in lost productivity, as survivors may be unable to return to work or require extended periods of recovery. The emotional toll on caregivers, who often provide unpaid support while balancing other responsibilities, adds another layer of stress (Benjamin, et al., 2019).The cumulative financial burden highlights the need for effective prevention strategies that can reduce the incidence and severity of strokes.

Prevention: One of the most promising aspects of stroke prevention is that a significant proportion of strokes are preventable. Early detection and intervention can dramatically lower the risk of stroke and its associated complications. By identifying individuals at risk—through tools that analyze factors like age, hypertension, obesity, diabetes, cholesterol levels, and lifestyle—healthcare providers can recommend tailored interventions. These may include lifestyle changes such as improving diet, increasing physical activity, and quitting smoking, as well as medical treatments like blood pressure or cholesterol-lowering medications (Anon., 2018).Predictive strategies enable healthcare systems to shift from reactive treatment to proactive prevention, ultimately saving lives and reducing the burden of this life-threatening condition.

1.2 Solution or Objectives

The primary objective of this stroke prediction machine learning model is to accurately predict the likelihood of an individual experiencing a stroke based on various medical and demographic factors. The model will provide healthcare professionals with a reliable tool to identify individuals at high risk of having a stroke, enabling early intervention, timely medical attention, and effective prevention strategies. By using our model into medical

applications, the goal is to improve stroke prevention, reduce the incidence of strokes, and ultimately save lives by facilitating proactive healthcare management.
Given the complexity of stroke risk factors, one-size-fits-all interventions are not enough. Instead, using machine learning models to analyze patient data can provide:

- Personalized risk assessments are based on a combination of medical history, demographics, and lifestyle factors.

- Real-time monitoring through wearable devices or mobile apps that can track key vitals like blood pressure, heart rate, and blood glucose levels.

- Proactive interventions, such as reminders for medication or doctor visits, tailored to an individual's risk profile.

By focusing on preventative care and timely warnings, such solutions can not only save lives but also reduce the economic and emotional toll of strokes.
(Chahine Y, 2023) (P.Divya, 2025) (Andrews, 2023)

1.3 Dataset Overview

Dataset is taken from Kaggle: link to our dataset

The dataset comprises 5,110 records with the following features:

- Demographics: Age, gender, residence type (urban or rural).
- Medical History: Hypertension, heart disease, average glucose level, BMI.
- Lifestyle Factors: Smoking status, marital status, and employment type.
- Target Variable: Stroke occurrence (1 = Stroke, 0 = No Stroke).

Data Challenges

- Missing Data: The BMI feature contains missing values, so we decided to assign meaning according to the genders.

- Imbalanced Classes: The dataset likely has a class imbalance (fewer stroke cases), Only 249 people have a stroke according to the dataset. If we balance it we'll be left with not too much data (500 approx.), Hence we decided to create

artificial data using Synthetic Minority Oversampling Technique (SMOTE) i.e. SMOTE to balance our data.

Exploration/ Findings from dataset:

- Outliers: BMI and glucose levels contain some outliers.
- Categorical Features: smoking status, work type, and marital status require one-hot encoding or binary encoding for use in machine learning models.
- Direct Correlation: Stroke is positively correlated with age, hypertension, heart disease, and glucose levels.
- Age Risk Factor: The dataset contains more older age group which relate to real-world cases where strokes are more prevalent in older populations as apparent from the dataset anyone under the age of 0 does not have a stroke. Thus, we decided that the data for the age group below 0 will be dropped. The data with the 'Other' gender also did not have a stroke and had just one data record, thus that record is also dropped.
- ID column is dropped as it was not needed.

Assumptions from the dataset

- Individuals with a combination of hypertension, higher avg glucose level, and advanced age are likely to be identified as high-risk based on the model's predictions.
- We assume that the dataset represents a diverse population and that features like "Age" and "residence_type" contribute meaningfully to stroke prediction.
- **Age**: Older adults are at a significantly higher risk due to the natural weakening of blood vessels.
- **Hypertension (High Blood Pressure)**: The single most important modifiable risk factor, hypertension increases the likelihood of stroke by up to four times.
- **Higher avg glucose levels**: High avg glucose level damage blood vessels over time, increasing the risk of strokes.

- **Lifestyle Factors**: Smoking, obesity, and lack of physical activity are major contributors. Cholesterol Levels: High LDL (bad cholesterol) can cause blockages in arteries, leading to strokes.

- **Gender and Genetics**: Women have a slightly higher lifetime risk, and family history plays a significant role.

Leading contributors include:

- Hypertension: A leading factor in strokes.
- Age: Older individuals are at higher risk.
- Smoking Status: Smoking significantly elevates risk.
- BMI: Obesity is a key risk factor.
- Gender: Gender can be a risk factor.
- Residence type: Living in a polluted environment can affect stroke risk.
- Glucose level: Higher glucose levels can be a risk factor.

1.4 CRISP-DM Approach

1. **Business Understanding**

   Define the Problem:

   (a) Stroke is a critical health issue that can lead to severe disability or death. This project aims to predict the likelihood of stroke occurrence based on patient data and identify the key factors that contribute to its onset. The objective is to enable early interventions and improve healthcare outcomes.

   Determine Success Criteria:

   (a) Predictive Accuracy: The model should achieve high accuracy and other performance metrics (e.g., precision, recall, F1-score) to ensure reliable predictions.
   (b) Actionable Insights: The results should provide clear, interpretable insights into factors contributing to stroke risk, enabling healthcare providers to design preventive measures.

2. **Data Understanding:**

   Inspect the Provided Dataset:

   Load and explore the dataset located at /data/healthcare-dataset-stroke-data.csv. Gain an understanding of the data's structure, size, and features, such as:

   (a) The types of variables (categorical, numerical).
   (b) Presence of any missing or inconsistent data.
   (c) Range, distribution, and summary statistics for each variable.

   Understand Data Quality:

   (a) Identify missing, duplicate, or inconsistent records.
   (b) Detect outliers or anomalous values that could impact model performance.
   (c) Evaluate data representativeness—ensure the dataset covers diverse demographics to avoid bias in predictions.

3. **Data Preparation:**

   Handle Missing or Inconsistent Data:

   a) Impute missing values using appropriate methods (e.g., mean/mode for numerical data, or categorical imputation).
   b) Remove duplicates and correct inconsistencies.

   Feature Engineering:

   a) Create derived features or transform existing variables to enhance the predictive power of the dataset (e.g., binning continuous features, creating interaction terms, or encoding categorical variables).
   b) Normalize or scale numerical variables to ensure consistent ranges.
   c) Handle categorical variables using encoding techniques like one-hot encoding or label encoding.

Data Splitting:

a) Split the cleaned dataset into training, validation, and testing subsets (e.g., 70/15/15 or 80/20 split) to evaluate and fine-tune model performance effectively.

4. **Modelling:**

Develop Models:

a) Experiment with various machine learning algorithms such as logistic regression, decision trees, random forests, gradient boosting, and neural networks.
b) Use automated or manual hyperparameter tuning techniques to optimize model performance.

Evaluate Models:

a) Use metrics such as accuracy, precision, recall, and F1-score to assess model effectiveness.
b) Analyze the confusion matrix to understand error patterns.
c) Perform cross-validation to ensure the model generalizes well to unseen data.

5. **Evaluation:**

Assess Model Performance:

a) Compare models against the success criteria (predictive accuracy and actionable insights).
b) Determine whether the models meet the desired thresholds for healthcare applications, ensuring low false negatives (i.e., missed stroke cases).

Ensure Actionable Insights:

a) Interpret the model's outputs to identify the most significant predictors of stroke (e.g., feature importance analysis or SHAP values).
b) Validate that the insights are understandable and practical for healthcare professionals.

6. **Deployment:**

   Present Findings:

   a) Summarize key findings and visualizations in a report or dashboard for stakeholders.
   b) Clearly highlight the main contributors to stroke risk and actionable recommendations for prevention.

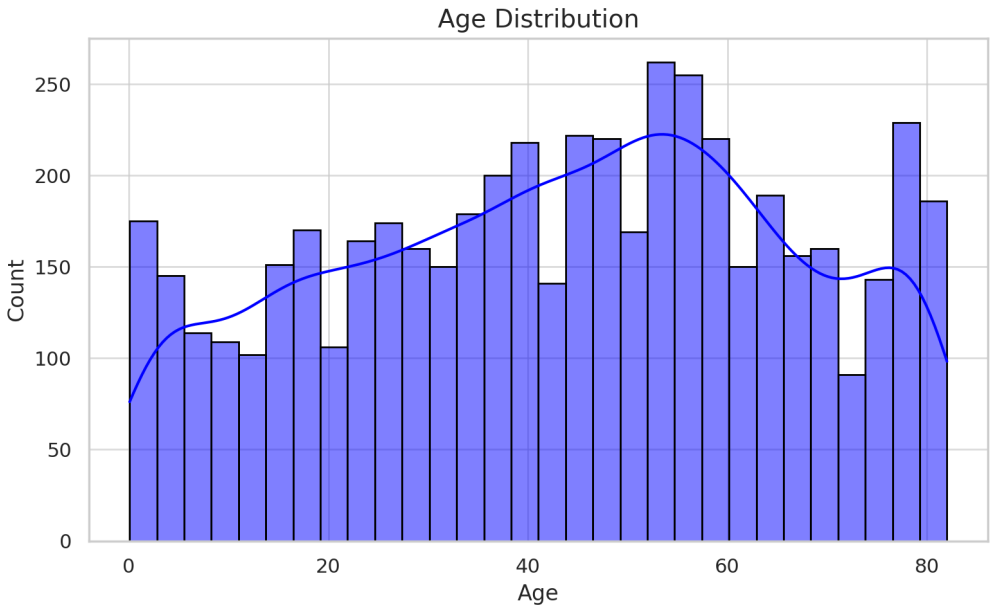
   Suggest Integration:

   a) Propose how the predictive model can be integrated into healthcare decision systems, such as clinical workflows, electronic health records, or patient monitoring systems.
   b) Include guidelines for periodic model retraining and monitoring to maintain accuracy over time.

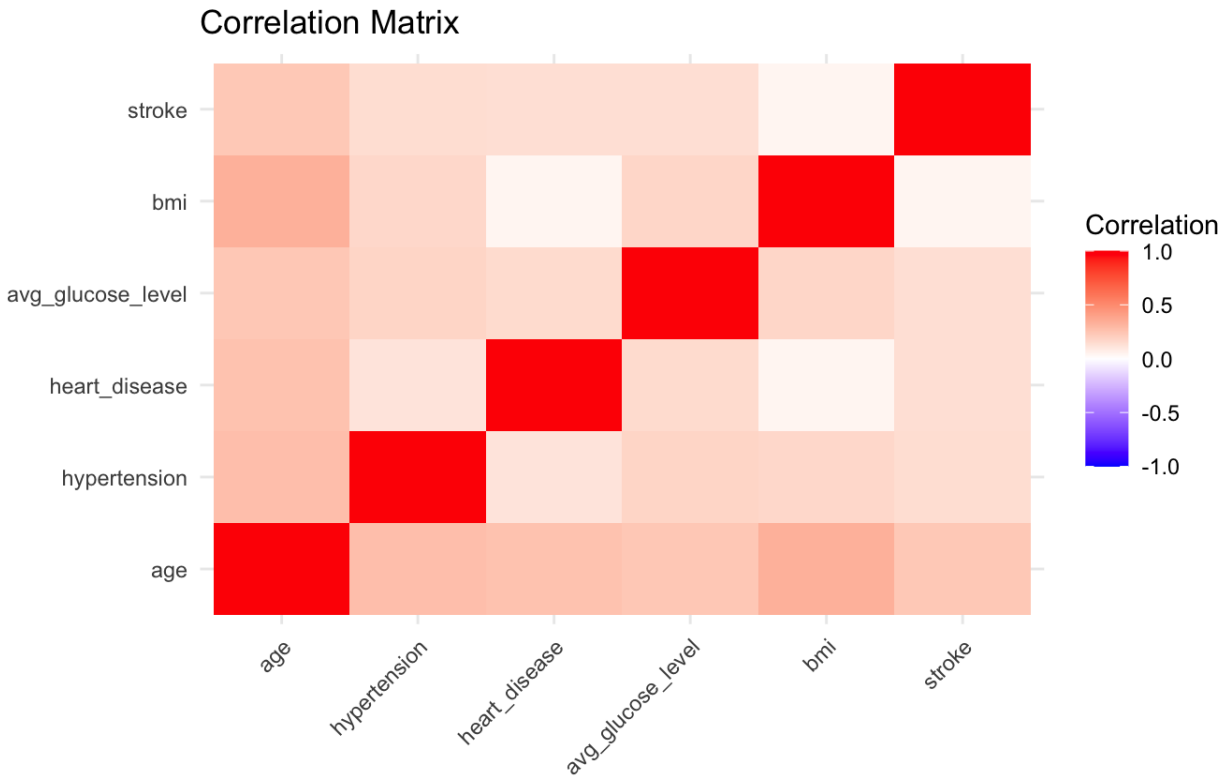
1.5 Visualizations and Graphical analysis before data preprocessing

Below graphs show some the data before preprocessing.

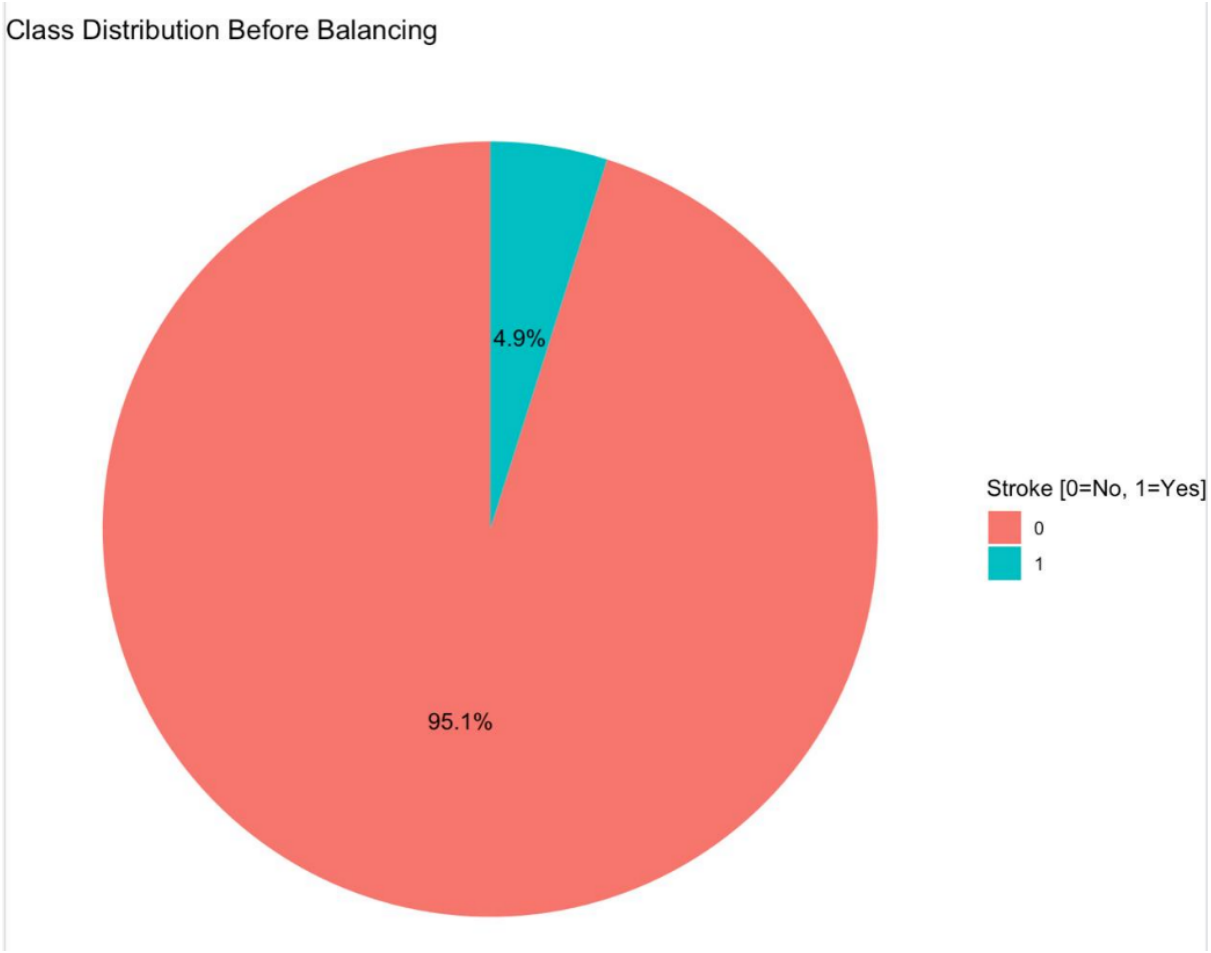
This bar chart shows the diversity of age between patients used for this research. It shows the majority of patients are within a range between late 50s and 60s.



This heat map shows the relationship between features, such as age, hypertension, heart disease, glucose levels, BMI, and stroke.



Below pie chart shows Imbalance dataset as it shows clearly the imbalance between stroke and non-stroke cases, highlighting the rarity of strokes in the dataset.



1.6 Dataset after preprocessing

Features Included After Preprocessing

The preprocessed dataset contains 27 columns. Here's a summary of the features included:

1. Age Groups (Binned): 0_10, 11_20, 21_30, …, 81_and_above: Age is represented in categorical bins for better handling of age-based patterns.

2. Demographic Features:

- age: The actual age value (not binned).
- gender: Gender of the patient.
- Residence_type: Urban or rural residence.

3. Medical History:

- hypertension: Presence of hypertension (0 = No, 1 = Yes).
- heart_disease: Presence of heart disease (0 = No, 1 = Yes).
- avg_glucose_level: Average glucose level.
- bmi: Body Mass Index.

4. Lifestyle Features:

- formerly_smoked, never_smoked, smokes, Unknown: Smoking status encoded into one-hot categories.

5. Employment Information:

- Private, Self_employed, Govt_job, children, Never_worked: Work type encoded into one-hot categories.

6. Target Variable:

- stroke: Whether the patient experienced a stroke (1 = Yes, 0 = No).

Features Dropped During Preprocessing

1. Original Categorical Columns: Columns such as work_type and smoking_status were replaced with their one-hot encoded versions.
2. Irrelevant or Redundant Features: id: This unique identifier for patients does not contribute to stroke prediction and was dropped.

With the uploaded dataset and suitable preprocessing, we have 5 possible models that should be able to predict stroke risks with high accuracy (>75% in balanced datasets).

1.7 Models (Techniques we are using)

After data exploration, some machine learning models were chosen to train and model the data to achieve our classification goals. The modelling techniques chosen were FNN (Feed-forward Neural Network) which will be used by all group members and compared later on to identify which parameters, or modelling options might be best suited for this case. We chose FNN for its ability to learn from unseen data when an FNN is trained well with the selection of hyper parameters, and the choice of the proper optimization algorithm.

The other models used for this project were Random Forest, XGBoost, SVM with optimization algorithm and stacking ensemble with the following based models LightGBM, CatBoostClassifier, and HistGradientBoostingClassfier.

Evaluation metrics used in our models: Accuracy, confusion matrix, recall, f1 score, ROC AUC, MCC

Data Loading

1. Data Set Loading

```
data_stroke <- read.csv("healthcare-dataset-stroke-data.csv")
```

2. We dropped the id column as it was irrelevant to prediction

```
data_stroke <- data_stroke %>% select(-id)
```

Data Inspection

1. Used mutate to ensure columns like smoking_status and work_type to be identified as categorial variables

```
data_stroke <- data_stroke %>%
  mutate(smoking_status = gsub(" ","_", smoking_status))%>%
  mutate(across(smoking_status, ~ as.factor(.))) %>%
```

2. Numeric column BMI is converted to numeric for processing

```
data_stroke$bmi <- as.numeric(data_stroke$bmi)
```

3. Str () is used to verify the type of columns

```
str(data_stroke$bmi)
```

Data Preprocessing

1. Data Cleaning: Removed rows with invalid value and rounded up the values for age column. Handled missing values in BMI by imputing mean, grouped by gender.

```
#Finding the mean/ average values of BMI depending on its gender
mean_bmi_gen<-data_stroke %>%
  group_by(gender) %>%
  summarise(mean_bmi = mean(bmi, na.rm = TRUE))

#Replacing the N/A BMI values with the mean values depending on its gender
data_stroke <- data_stroke %>%
  left_join(mean_bmi_gen, by = "gender") %>%
  mutate(bmi = ifelse(is.na(bmi), mean_bmi, bmi)) %>%
  select(-mean_bmi)
```

2. One-Hot Encoding: used pivot_wider to perform one-hot encoding on categorial values lik smoking_status and age_range.

```
#One-hot encoding the 'age_range' column
data_stroke <- data_stroke %>%
  pivot_wider(names_from = age_range,
              values_from = age_range,
              values_fn = length,
              values_fill = 0)
```

3. Binary Encoding: used to assign binary values (0,1) to categorial columns like gender and ever_married

```
data_stroke<- data_stroke %>%
  mutate(gender = ifelse(gender =="Female",1, 0))
```

Hypothesis for field selection

Inclusion Criteria:
Variables that are directly associated with health such as age, BMI, glucose levels, smoking status, work type etc. are included

Exclusion Criteria:
Variables such as id are excluded, categories with insufficient representation as gender=" others" are also removed

Balancing the dataset

1. SMOTE: To balance the minority class

```
#Applying SMOTE
data_smote <- SMOTE(
  X = train_data[, -which(names(train_data) == "stroke")],
  target = train_data$stroke,
  K = 5,
  dup_size = 3
)
```

2. Undersampling: To equalize the class distribution

```
#Appling under sampling
data_undersampled <- ovun.sample(
  stroke ~ .,
  data = smote_train_data,
  method = "under",
  N = 2 * sum(smote_train_data$stroke == 1)
)$data
```

Visualization after preprocessing

1. Class distribution is visualized before and after balancing

```
#Visualizing class distribution before balancing data using barplot
ggplot(data_stroke, aes(x = factor(stroke))) + geom_bar(fill='skyblue') +
  labs(title = "Class Distribution Before Balancing", x = "Stroke [0=No, 1=Yes]", y = "Count") +
  theme_minimal()
```

2. PCA helps to understand the dataset variance

```
pca <- prcomp(data_scaled, center=TRUE, scale.=TRUE)   #performing PCA
summary(pca)

#Visualizing PC1 by stroke status using boxplot
pc1 <- pca$x[,1]
boxplot_data <- data.frame(PC1=pc1, stroke=data_undersampled$stroke)
```

Exporting the cleaned data

The cleaned and dataset that we will be working on is exported to a csv file

```
#Converting the under sampled data set to csv file
write.csv(data_undersampled,'data_stroke_undersampled.csv', row.names = FALSE)
```

2. Data Understanding, Pre-processing and Exploration

The dataset used for this project includes various independent variables, such as age, BMI, work type, and others, alongside the target variable, which indicates whether the individual experienced a stroke. Several issues in the dataset required resolution to prevent overfitting and other complications during modeling, for this the following steps were done.

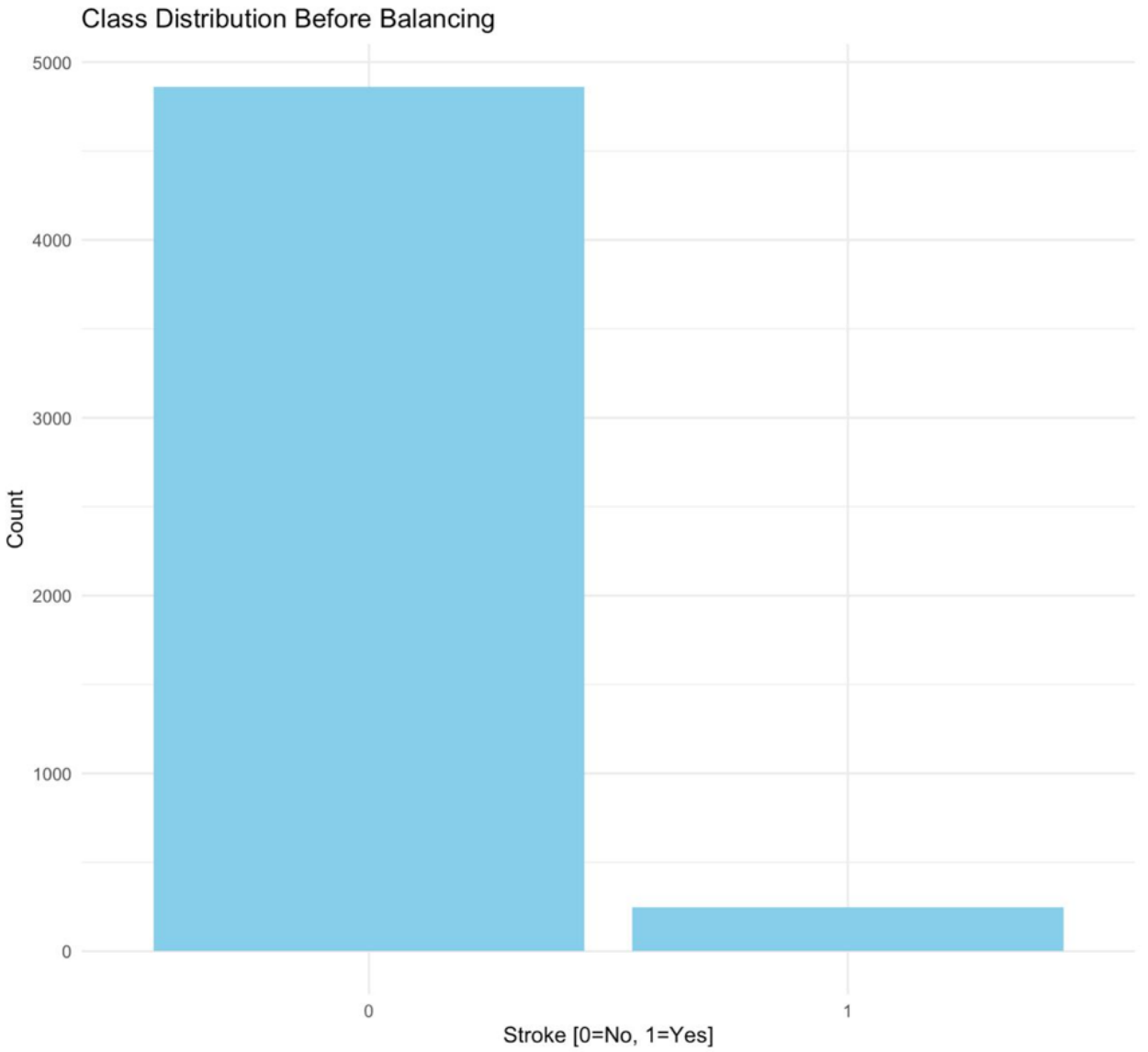
2.1 Cleaning the data

We started by removing the id column as it was irrelevant to the objectives of the research.
One notable issue was the missing data in the BMI column. To address this, we calculated the mean BMI for each gender and inserted the missing values accordingly.
For the “age” column, all values below 0 were discarded, as no patient that was below this age had a stroke. The remaining ages were rounded up to the nearest whole number. Then the ages were grouped in different ranges. We created a column for each range.

2.2 Feature Engineering, normalization and data split

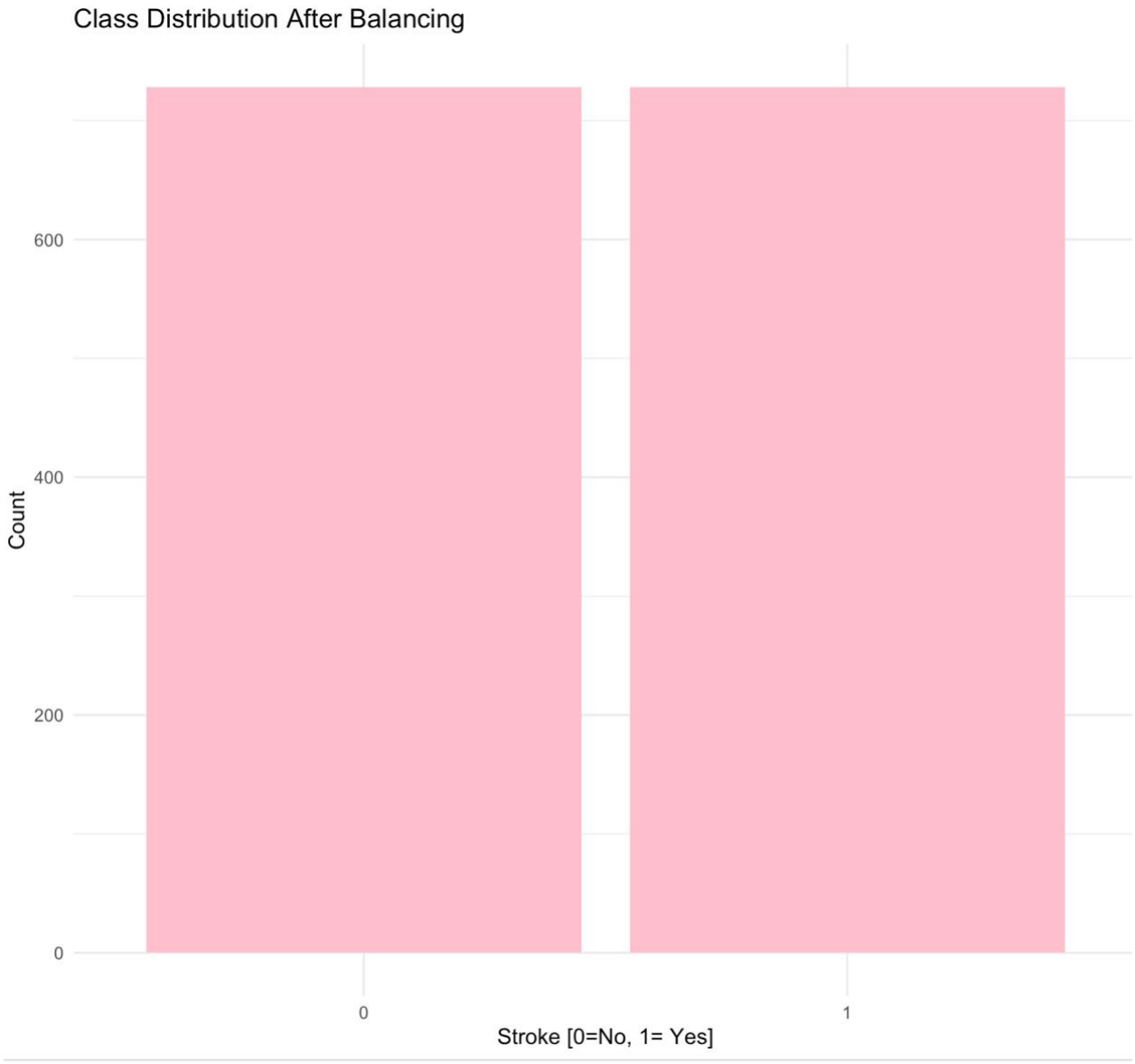
We transformed several existing features to align them with our modeling requirements. Categorical columns with more than two options, such as smoking status, age range, and the work type, were one-hot encoded. For features with only two possible values, binary encoding was applied. This approach was used for the residence type, marriage status, and gender. In the case of the “gender” column, we excluded the “other” category since it contained only one individual who did not have a stroke. A decision for encoding values was taken as one hot encoding and binary encoding, both improve interpretability and compatibility across ML models.

The dataset was imbalanced, with only 249 individuals out of 500 having a stroke as shown in the image below.



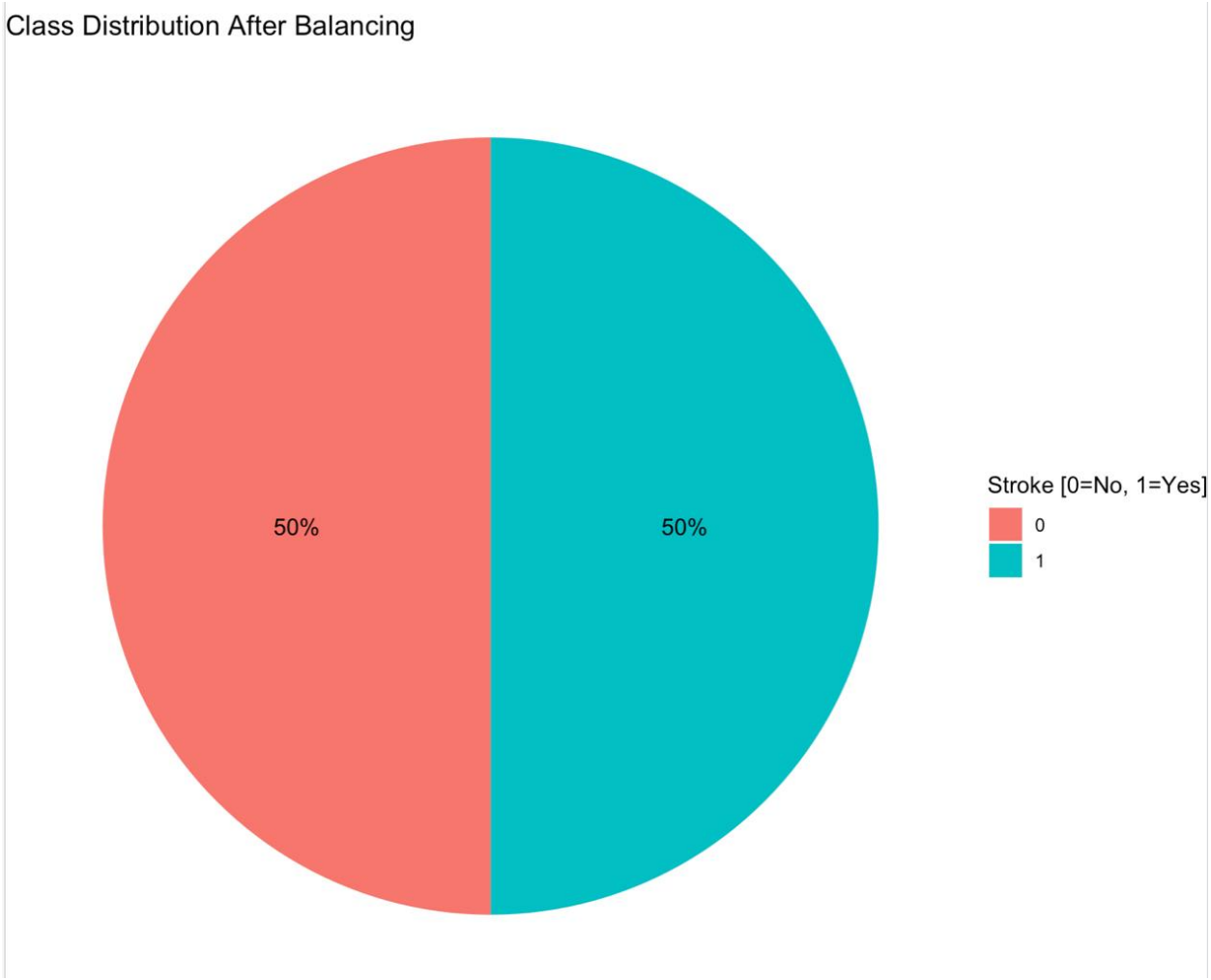
Imbalanced datasets can introduce challenges during modeling, such as misleading performance metrics, bias toward the majority class, and potential overfitting to the minority class. To address this, we scaled the data and employed a hybrid approach to balance the dataset. After splitting the data into 70% training and 30% testing sets, we applied a combination of under sampling and SMOTE (Synthetic Minority Oversampling Technique) to the training data. In this case, the seed was set to 42 for the split, this seed was also kept for the modelling phase in order to create reproducibility, across all modelling techniques.

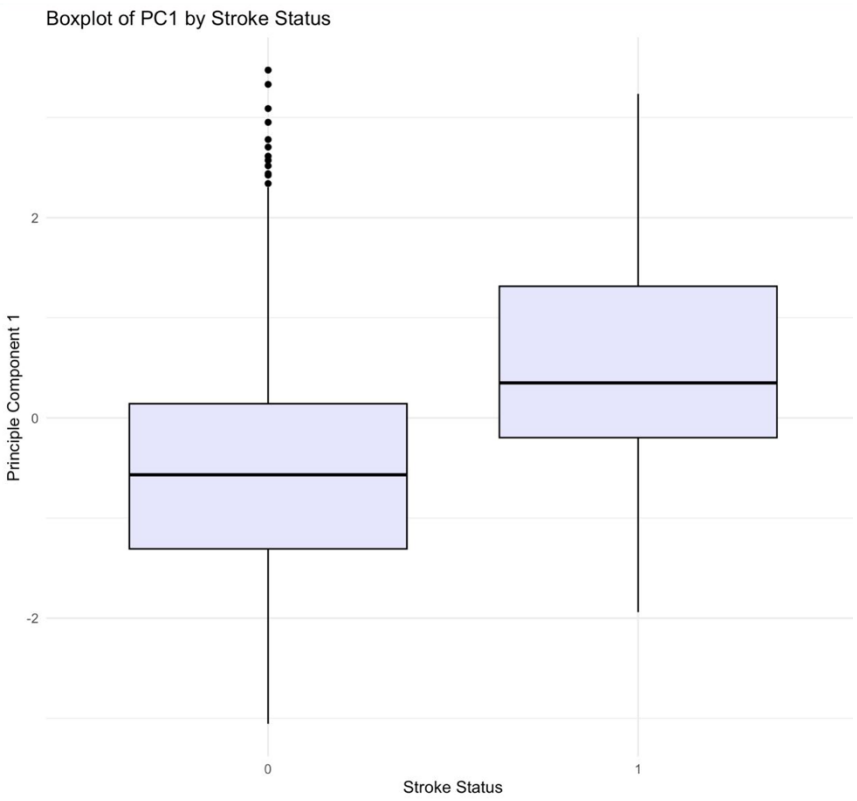
This resulted in the same amount of data for class 0 and class 1 as shown in the following images. This enabled a more efficient modelling process and therefore better classification results.



2.3 Data Visualization after preprocessing

A pie chart of the preprocessed data was also included in the introduction section, the following one shows the data after being cleaned and process to show the difference.






```
$stats
           [,1]       [,2]
[1,] -3.0554866 -1.9403817
[2,] -1.3094607 -0.1973484
[3,] -0.5682533  0.3496043
[4,]  0.1440360  1.3162399
[5,]  2.3036226  3.2366256
```

For further data exploration, a Principal Component Analysis was done based on the processed data after cleaning and balancing it. This shows the data distribution for each class. PC1 reduces the dimensions whilst still retaining as much variance as possible. This diagram shows the difference in data after preprocessing between the two classes. The stats image shows [, 2] as class 1, i.e., patients that had a stroke, and [,1] as class 0. According to this, the median for class 1 (0.35) is higher than class 0 (-0.57) which is due to higher values of BMI, avg glucose level, and age for patients that had a stroke. We can also appreciate that there are far more outliers in class 0 than in class 1 which indicates that there is a greater variability in the features like age, BMI, and glucose levels among people that have not had a stroke.

3. Modeling

3.1 Sampada Shrestha [6892246]

Random Forest (unique model)

Random Forest is an effective machine learning model which works by creating several decision trees during the training phase. The model combines the outcomes of all trees by voting for classification tasks. This collective decision-making process, supported by multiple trees and their insights, provides stable results. This ML model is widely used for both classification and regression tasks, as they are simple, can handle complex data, reduce overfitting, and provide reliable predictions (GeeksforGeeks, 2024).
Random Forest is selected as a machine learning model for this project due to its ability to be simple yet effective, handle outliers, and class imbalance. Some of the strengths and weaknesses of the model are mentioned below.

Strengths

- It performs well with noisy data as it can handles outlier effectively.
- It can work with both numerical and categorical data, like in the dataset used for this project.
- Handles imbalance in data with the ‘class_weight=balanced’ parameter, which adjusts the weight to address the underlying class imbalance.
- It does not require much of data preprocessing.
- It has built-in feature importance where feature variables are ranked based on their impact/importance in regard to the target variable. This can help identify major risk factors for stroke.

Weaknesses

- It is black box in nature, which makes this model less interpretable.
- This model is not as scalable compared to other models.

Development of the Model

The preprocessed dataset was loaded using pandas and split into training (70%) and testing (30%) sets. The dataset was divided using stratification (stratify=y) to keep the class distribution consistent throughout both train and test sets.

```
#loading the dataset
stroke_data= pd.read_csv('data_stroke_undersampled.csv')
stroke_data.head()

x= stroke_data.drop(columns=['stroke'])      #feature variable
y= stroke_data['stroke']                     #target variable

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42, stratify=y)
```

A standard random forest model was then implemented to establish a point of comparison for further improvements. The default model parameters were n_estimators=100 (the number of trees to be used in the model), random_state=42 (for reproducibility), and class_weight='balanced' (for handling any underlying class imbalance). The model was then trained with the training dataset, made predictions, and performance evaluation of the model was done using the test dataset.

```
#Before hyperparameter tuning

random_forest= RandomForestClassifier(n_estimators=100, random_state=42, class_weight='balanced')
random_forest.fit(x_train, y_train)             #training the model using train dataset

predict_y= random_forest.predict(x_test)           #making prediction on test data
prob_y= random_forest.predict_proba(x_test)[:,1]    #calculating probabilty estimates for Stroke (
```

The evaluation metrics for the standard random forest model was as follow:

- Accuracy: 0.8627002288329519
- Precision: Class 0 [Not Stroke]: 0.94, Class 1 [Stroke]: 0.81
- Recall: Class 0 [Not Stroke]: 0.77, Class 1 [Stroke]: 0.95
- F1 Score: Class 0 [Not Stroke]: 0.85, Class 1 [Stroke]: 0.87
- Matthews Correlation Coefficient (MCC): 0.74
- ROC AUC Score: 0.9586527585773532

Based on the standard random forest model's performance, hyperparameter tuning was done using RandomizedSearchCV to optimize the model's performance. Through this approach, a given set of hyperparameters is explored and the best combination is selected based on cross-validation performance. For this model, 100 iterations and 5-fold cross validation was used to find the best parameter combination. The hyperparameters that were explored are as follow:

- n_estimators: This specifies the number of decision trees in the model. A range of 100-500 was assigned and explored.

- max_depth: This controls maximum depth of each tree in the forest. The values None, 10, 20, and 30 was explored to test how different depths would affect the model performance.

- min_samples_split: This parameter was chosen to help prevent overfitting by adjusting the number of minimum samples required to split a node from 2 to 10.

- min_samples_leaf: This parameter was also helps in preventing overfitting by setting a minimum number of sampled at each leaf node. The values ranging from 1 to 10 were tested.

- max_features: This defines the number of features to consider when determining the best split at each node. None, sqrt and log2 were tested.

- bootstrap: This parameter determines whether bootstrap sampling is used when building trees, with values either True or False.

These parameters were selected as they have a significant impact on performance and generalization capacity of a Random Forest model, making them ideal for hyperparameter tuning.

```
#After Hyperparameter tuning

parameters= {'n_estimators':randint(100,500),
             'max_depth':[None, 10,20,30],
             'min_samples_split': randint(2, 10),
             'min_samples_leaf': randint(1, 10),
             'max_features': [None, 'sqrt', 'log2'],
             'bootstrap': [True, False]
            }

random_search= RandomizedSearchCV(estimator=random_forest, param_distributions=parameters,n_iter=100, cv=5, n_jobs=-1, random_state=42, verbose=2)

random_search.fit(x_train, y_train)               #training the model using train dataset

best_params =random_search.best_params_
print('Best Parameters:', best_params)

best_rf = random_search.best_estimator_
```

The best combination of parameters selected by the RandomizedSearchCV was as follow:

- n_estimators: 483
- max_depth: 20

- min_samples_split: 5
- min_samples_leaf: 1
- max_features: sqrt
- bootstrap: False

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
Best Parameters: {'bootstrap': False, 'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split':
5, 'n_estimators': 483}
```

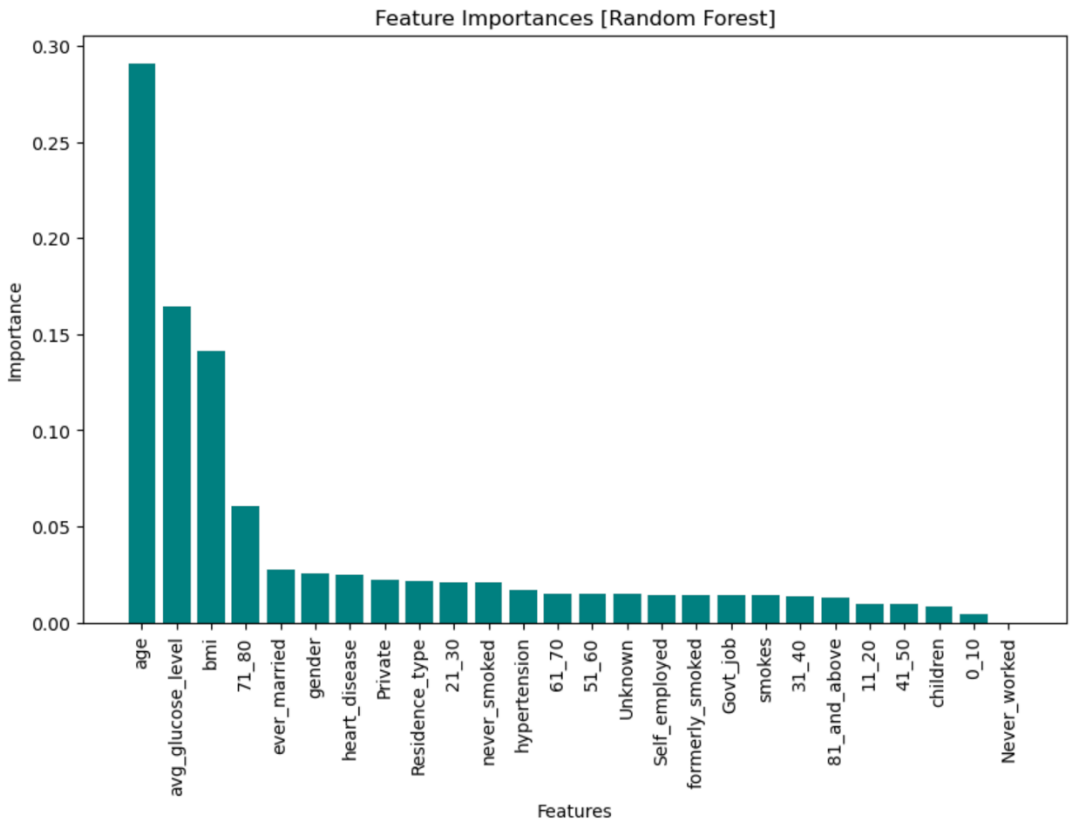
The evaluation metrics for the random forest model after hyperparameter tuning were as follow:

- Accuracy: 0.88
- Precision: Class 0 [Not Stroke]: 0.94, Class 1 [Stroke]: 0.84
- Recall: Class 0 [Not Stroke]: 0.82, Class 1 [Stroke]: 0.95
- F1 Score: Class 0 [Not Stroke]: 0.88, Class 1 [Stroke]: 0.89
- Matthews Correlation Coefficient (MCC): 0.77
- ROC AUC Score: 0.961250052364794

In addition to hyperpamater tuning, feature importance was also evaluated by extracting the feature importances from the trained Random Forest model. Feature importance helps identify which features or input variables plays a more significant role in helping the model predict between the two target classes: Stroke and Not Stroke.

```python
#Feature Importance
feature_imp = best_rf.feature_importances_
indices = np.argsort(feature_imp)[::-1]
features = x_train.columns

#Visualizing feature importance
plt.figure(figsize=(10, 6))
plt.title("Feature Importances [Random Forest]")
plt.bar(range(x_train.shape[1]), feature_imp[indices], align="center", color="teal")
plt.xticks(range(x_train.shape[1]), features[indices], rotation=90)
plt.xlabel("Features")
plt.ylabel("Importance")
plt.show()
```



The graph above indicates that age, avg glucose level and BMI were the features with the highest importance in predicting stroke risk.

Feedforward Neural Network (shared model)

Feedforward Neural Network is a form of artificial neural network where the information flows in a single direction, i.e. from input layer to output layer. This neural network consists of three main layers: input layer, one or multiple hidden layers and an output layer. The input layer is made up of the neurons that receive input data. The hidden layers learn the complex patterns in the data and each neuron in this layer applies weighted sum of inputs, followed by a non-linear activation function. These activation functions add non-linearity to the model. Finally, the output layer provides the final output of the model. For classification tasks, the number of neurons in output layer represents the number of classes. Each connection between the neurons in these layers has a weight that is modified during training to reduce prediction errors (GeeksforGeeks, 2024).

Feedforward Neural Network is selected as a machine learning model for this project mainly due to its ability to capture non-linear relationships in the data and its scalability. Some of the strengths and weaknesses of the model are mentioned below.

Strengths

- It can effectively capture complex nonlinear relationship in data, which can exist in a heath related dataset used for the project.
- It has flexible architecture that can be customized according to meet the requirements of the dataset.
- It is also scalable and can be adjusted for real-time predictions as well.

Weaknesses

- It is sensitive to unscaled data and requires the feature data to be scaled.
- It is computationally expensive compared to other models.
- If proper regularization and early stopping is not implemented, the model may lead to overfitting.

Development of the Model

The same data split of 70% training and 30% testing is used for feedforward neural network as well. Additionally, the dataset was also scaled using StandardScaler to normalize the feature variables, ensuring that all features are on a similar scale, before

```
#Scaling data for FNN
scaler = StandardScaler()
scale_x_train= scaler.fit_transform(x_train)
scale_x_test= scaler.transform(x_test)

scale_x_train= pd.DataFrame(scale_x_train, columns=x_train.columns)
scale_x_test= pd.DataFrame(scale_x_test, columns=x_test.columns)
```

training the FNN model. The training and testing datasets were scaled separately in order to prevent data leakage. The scaled data was converted back to the DataFrame format for easier handling.

The Optuna library was used to perform hyperparameter tuning for the FNN model, as this library can efficiently explore a large hyperparameter space to find the best combination of the parameters for the model. The following hyperparameter were selected for tuning:

- Dense Layers: The Layer1 and Layer2 represent the dense (hidden) layers of the neural network where the network learns about the patterns and relationships between feature and target variables. The Layer1 and Layer2 define the number of neurons in the network's first and second layers respectively. While a larger number of neurons may improve the model's performance, it can also cause the model to overfit, which is why a range from 16 to 64 with step 16 (16, 32, 48, 64) are tested to find the optimal number of neurons for these hidden layers.

- Dropout Rates: The dropout rates were added to avoid overfitting by randomly dropping a part of the neurons during the training process. The range from 0.2 to 0.5 with an increment of 0.1 were tested.

- L2 Regularization: The L2 regularization can help better model training by penalizing large weights in the hidden layers, preventing overfitting and improving generalization. The regularization strength was tuned between 0.0001 and 0.001.

- Learning rate: The Adam's optimizer's learning rate controls how fast a model learns, which is tuned between 0.0001 and 0.001.

```
np.random.seed(42)
tf.random.set_seed(42)
random.seed(42)

def hyperparameters(hp_test):
    layer1= hp_test.suggest_int('layer1', 16, 64, step=16)
    layer2= hp_test.suggest_int('layer2', 16, 64, step=16)
    dropout1= hp_test.suggest_float('dropout1',0.2, 0.5, step=0.1)
    dropout2= hp_test.suggest_float('dropout2',0.2, 0.5, step=0.1)
    l2_regularizer1= hp_test.suggest_float('l2_regularizer1',0.0001, 0.001, log=True)
    l2_regularizer2= hp_test.suggest_float('l2_regularizer2',0.0001, 0.001, log=True)
    learning_rate= hp_test.suggest_float('learning_rate', 0.0001, 0.001, log=True)

    fnn= Sequential([
    Input(shape=(scale_x_train.shape[1],)),
    Dense(layer1, activation='relu', kernel_regularizer=l2(l2_regularizer1)),
    Dropout(dropout1),
    Dense(layer2, activation='relu', kernel_regularizer=l2(l2_regularizer2)),
    Dropout(dropout2),
    Dense(1,activation='sigmoid')
    ])

    fnn.compile(optimizer=Adam(learning_rate=learning_rate),loss='binary_crossentropy', metrics=['accuracy'])

    fnn.fit(
        scale_x_train, y_train,
        epochs=30,
        batch_size=32,
        validation_data=(scale_x_test, y_test),
        verbose=0
    )

    val_loss, val_accuracy= fnn.evaluate(scale_x_test, y_test, verbose=0)
    return val_loss

study= optuna.create_study(direction='minimize', sampler=optuna.samplers.TPESampler(seed=42))
study.optimize(hyperparameters, n_trials=10)
```

Optuna performed 10 trials to find the best combination of the above-mentioned hyperparameters. Seed was included in the tuning process as well for reproducibility. The following are the best combination of hyperparameters selected by Optuna:

- Layer 1: 32
- Layer 2: 32
- Dropout 1: 0.4
- Dropout 2: 0.2
- L2_regularizer1: 0.0006341572775495276
- L2_regularizer2: 0.00011872731425335906
- Learning rate: 0.0009702573394120733

```
Best Parameters:
{'layer1': 32, 'layer2': 32, 'dropout1': 0.4, 'dropout2': 0.2, 'l2_regularizer1': 0.0006341572775495276, 'l2_regularizer2':
0.00011872731425335906, 'learning_rate': 0.0009702573394120733}
```

Furthermore, early stopping was also applied to prevent overfitting. For this model, early stopping monitored validation loss and when it did not improve for three consecutive epoch, the early stopping would be triggered.

```
earlystop= EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
```

The model was trained for a maximum of 50 epochs with a batch size of 32. The validation data was passed to evaluate the performance of the model on new, unseen data after each epoch during training process. The model then made predictions, which were converted to 0 or 1 (if the predicted value was less then 0.5 then 0 or else 1) for binary classification.

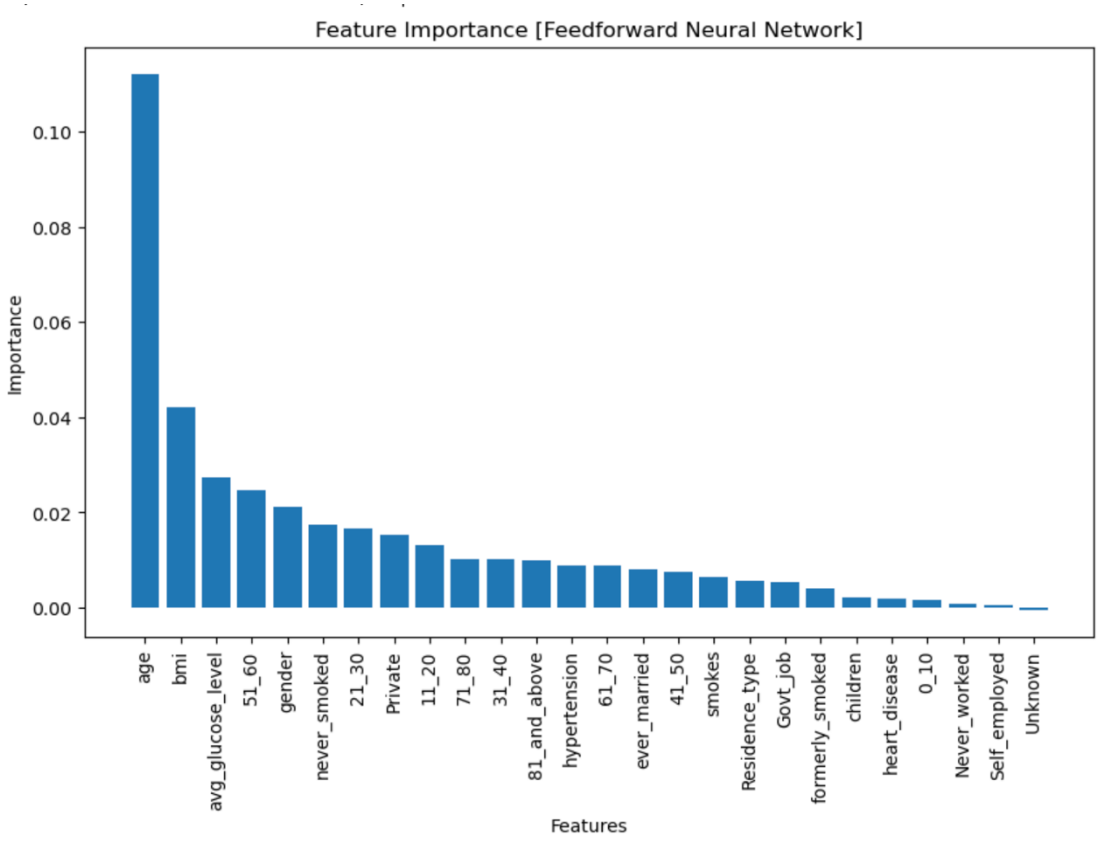
```
fnn_model= best_fnn.fit(
    scale_x_train, y_train,
    epochs=50,
    batch_size=32,
    validation_data=(scale_x_test, y_test),
    callbacks=earlystop,
    verbose=1
    )

fnn_prob_y= best_fnn.predict(scale_x_test)                #making predictions
fnn_predict_y= (fnn_prob_y>0.5).astype(int).flatten()     #converting the predictions to class 0 or 1
```

The evaluation metrics for the FNN model using the best hyperparameter combination were as follow:

- Accuracy: 0.78
- Precision: Class 0 [Not Stroke]: 0.83, Class 1 [Stroke]: 0.75
- Recall: Class 0 [Not Stroke]: 0.72, Class 1 [Stroke]: 0.85
- F1 Score: Class 0 [Not Stroke]: 0.77, Class 1 [Stroke]: 0.80
- Matthews Correlation Coefficient (MCC): 0.57
- ROC AUC Score: 0.7850425202128106

Lastly, feature importance evaluation was also done after training the FNN model using the permutation importance method. For this, a custom scoring function was defined to convert the predictions to binary values, and the importance scores were calculated for 10 repeats. The permutation importance method measured the significance of each feature variable by calculating the decline in model accuracy when feature's value where shuffled. The evaluation showed that age, BMI and avg glucose levels are the most important features in predicting stroke.



3.2 Syeda Fizza Raza [6509217]

Stacking Ensemble Model (unique model)

Gradient Boosting-Focused Ensemble
Base Models:

- LightGBM
- CatBoostClassifier
- HistGradientBoostingClassifier - A Scikit-learn implementation of gradient boosting that is highly efficient and handles both numerical and categorical features.

Meta-Model:

- CatBoostClassifier

Evaluation: Use metrics such as Accuracy, Precision, Recall, F1-score, and AUC-ROC for comparison with FNN and other models to be consistent.

Why choose this ensemble combination?

Performance, minimal preprocessing, and ease of implementation. It is well-suited for tabular data with a mix of categorical and numerical variables, and the gradient boosting models will likely yield higher accuracy and clearer feature importance.

Strengths:

- **Optimized for Tabular Data:** Gradient boosting models like LightGBM, CatBoost, and HistGradientBoosting are specifically designed for structured/tabular datasets, making them ideal for stroke prediction.

- **Handles Missing and Categorical Data:** CatBoost and LightGBM handle categorical data natively without preprocessing.Gradient boosting models are robust against missing values, simplifying preprocessing.

- **Minimal Preprocessing Required:** No need to scale data or apply extensive encoding techniques.

- **High Accuracy and Interpretability:** Gradient boosting models provide feature importance metrics, making it easier to identify key predictors for strokes (e.g., hypertension, age, BMI).

- **Ease of Implementation:** It is very straightforward to implement in Python using libraries like LightGBM,CatBoost, and scikit-learn. It requires hyperparameter tuning for optimal performance, but default settings often yield competitive results.

Implementation of Stacking Ensemble

This model uses a Stacking Classifier to predict strokes based on a dataset of features. Below is a breakdown of the implementation:

1. Data splitting

```
X = stroke_data.drop(columns=['stroke']) y = stroke_data['stroke'] X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.3, random_state=42, stratify=y )
```

Splitting the data ensures the model is trained on one portion (70%) and tested on unseen data (30%) to evaluate its performance.
Stratify=y ensures the class distribution (e.g., stroke vs. no stroke) is maintained in both training and test sets, critical for imbalanced datasets like stroke prediction.

2. Feature Scaling

```
scaler = StandardScaler() X_train = scaler.fit_transform(X_train) X_test = scaler.transform(X_test)
```

Standardized numerical features by centering them around the mean and scaling to unit variance. This is important because many machine learning models (e.g., gradient boosting) perform better when features are on a similar scale. It avoids features with larger ranges dominating the learning process.

3. Stacking Classifier Setup

```
base_estimators = [
    ('lightgbm', LGBMClassifier(random_state=42)),
    ('catboost', CatBoostClassifier(verbose=0, random_state=42)),
    ('histgradient', HistGradientBoostingClassifier(random_state=42)),
]
meta_model = CatBoostClassifier(verbose=0, random_state=42)
stacking_clf = StackingClassifier( estimators=base_estimators, final_estimator=meta_model, cv=5 )
```

Stacking combines multiple base models and uses their predictions as inputs to a meta-model.
Cross-Validation (cv=5): Ensures robustness by splitting the training data into 5 folds.Each base model is trained on 4 folds and validated on the remaining fold.The meta-model is trained on the predictions from these folds, ensuring no data leakage.

The stacking classifier is trained on the training data. Each base model is trained independently, and their predictions are passed to the meta-model.

4. Model Training

```
stacking_clf.fit(X_train, y_train)
```

The stacking classifier is trained on the training data. Each base model is trained independently, and their predictions are passed to the meta-model.

5. Prediction

```
y_pred = stacking_clf.predict(X_test)
y_pred_proba = stacking_clf.predict_proba(X_test)[:, 1]
```

predict: Outputs binary predictions (stroke or no stroke).
predict_proba: Outputs probabilities for each class. For stroke prediction, we use the probabilities of the positive class (stroke).

Hyperparameter Tuning on the Stacking Ensemble

Applied Hyperparameter tuning to further improve performance and stroke prediction. The model is fine-tuned using RandomizedSearchCV, which optimizes hyperparameters for better performance.

Hyperparameter Tuning implementation:

RandomizedSearchCV
I used RandomizedSearchCV as it randomly samples hyperparameter combinations from a specified distribution. It is faster than GridSearchCV, especially when the hyperparameter space is large. It uses cross-validation to ensure robustness.

Key Parameters:

- param_distributions: Specifies the range of hyperparameters to tune.
- n_iter=10: Limits the search to 10 random combinations, balancing computational cost and performance.
- cv=5: Ensures robust evaluation using 5-fold cross-validation.
- n_jobs=-1: Utilizes all available CPU cores for faster computation.

```
random_search = RandomizedSearchCV( estimator=stacking_clf, param_distributions=param_dist, n_iter=10, cv=5, n_jobs=-1, verbose=10, random_state=42 )
```

Parameter Distribution

Base Models:

- LightGBM: Tuning n_estimators, learning_rate, and max_depth controls the number of trees, their learning speed, and complexity.
- CatBoost: Tuning iterations, depth, and learning_rate optimizes the number of trees, tree depth, and learning speed.
- HistGradientBoosting: Tuning max_iter and learning_rate adjusts the number of boosting iterations and learning speed.

Meta-Model:

- Tuning iterations, depth, and learning_rate optimizes the meta-model's complexity and learning speed.

```
param_dist = { 'lightgbm__n_estimators': [50, 100, 150], 'lightgbm__learning_rate': uniform(0.05, 0.1), 'lightgbm__max_depth': [3, 5, 7, 10], 'catboost__iterations': [50, 100, 150], 'catboost__depth': [4, 5], 'catboost__learning_rate': uniform(0.01, 0.1), 'histgradient__max_iter': [100, 150], 'histgradient__learning_rate': uniform(0.05, 0.2), 'final_estimator__iterations': [100, 150, 200], 'final_estimator__depth': [3, 4, 5, 6], 'final_estimator__learning_rate': uniform(0.01, 0.1), }
```

Training the model with hyperparameters

```
random_search.fit(X_train, y_train)
```

RandomizedSearchCV trains the stacking classifier on different hyperparameter combinations. Evaluates each combination using 5-fold cross-validation. Selects the best combination based on cross-validation performance.

Refining the hyperparameter process

I refined the hyperparameter tuning process for the stacking classifier to build upon the results of the initial RandomizedSearchCV to further optimize the model's performance.

1. Analyzing All Hyperparameter Combinations

```
results = random_search.cv_results_ df_results = pd.DataFrame(results) df_results_sorted = df_results.sort_values(by='mean_test_score', ascending=False) print("\nTop Best Hyperparameter Combinations:\n") print(df_results_sorted)
```

This inspects all combinations of hyperparameters tested during the initial search and sorts the combinations by their mean test score to identify the top-performing ones.

2. Refine the search

```
best_params = random_search.best_params_
param_dist_refined = {
 'lightgbm__n_estimators': [best_params['lightgbm__n_estimators']],
'lightgbm__learning_rate': uniform(max(0.01, best_params['lightgbm__learning_rate'] - 0.03), 0.06),
 'lightgbm__max_depth': [best_params['lightgbm__max_depth'] - 1, best_params['lightgbm__max_depth'], best_params['lightgbm__max_depth'] + 1],  ...
}
```

I kept some parameters fixed (e.g., n_estimators) and refined others (e.g., learning_rate, max_depth) within a smaller range.

3. Setting Up the Refined Search

```
random_search_refined = RandomizedSearchCV( estimator=stacking_clf, param_distributions=param_dist_refined, n_iter=15, cv=5, n_jobs=-1, verbose=10, random_state=42 )
```

Adjustments:
param_distributions: Uses the refined hyperparameter grid.
n_iter=15: Increases the number of iterations slightly for more thorough exploration.
cv=5: Maintains 5-fold cross-validation for robust evaluation.
n_jobs=-1: Utilizes all CPU cores to speed up the search.

4. Fitting the Refined Search

The refined search is performed, testing the new hyperparameter combinations.Each combination is evaluated using 5-fold cross-validation.The best combination is selected based on the mean test score.
My approach of implementation is a two-stage hyperparameter tuning process:
Initial RandomizedSearchCV: Broadly explores the hyperparameter space.
Refined RandomizedSearchCV: Focuses on the most promising regions of the parameter space for fine-tuning.

Predicting and Evaluating using the best Refined Stacking Ensemble

I then evaluated the refined stacking model by using the best parameters identified in the second stage of hyperparameter tuning. I assessed the model's performance on the test set using various metrics.

1. Get the best model after Refined Tuning

Retrieves the best model (stacking classifier) from the refined search, trained with the optimal parameters.
best_model_refined = random_search_refined.best_estimator_

2. Calculate predictions and probabilities

y_pred_refined = best_model_refined.predict(X_test) y_pred_proba_refined = best_model_refined.predict_proba(X_test)[:, 1]
y_pred_refined: Predicts class labels (e.g., stroke/no-stroke) for the test set. y_pred_proba_refined: Predicts the probabilities for the positive class (stroke).

3. Get the best parameters for my Stacking Ensemble model

Best hyperparameters for Stacking Ensemble
Best Parameters for Stacking Model:
Test Size: 0.3
Cross-Validation Folds: 5
Randomized Search Iterations: 15
Refined Best Score: 0.8538

LightGBM Parameters:
- Learning Rate: 0.0598
- Max Depth: 6
- Number of Estimators: 150

CatBoost Parameters:
- Learning Rate: 0.1012
- Depth: 4
- Iterations: 50

HistGradientBoosting Parameters:
- Learning Rate: 0.2378
- Max Iterations: 150

Meta-Model (CatBoost) Parameters:

- Learning Rate: 0.1153
- Depth: 3
- Iterations: 100

4. Evaluation

At the end I calculated various metrics to measure the refined model's performance:

```
accuracy_refined = accuracy_score(y_test, y_pred_refined)
f1_refined = f1_score(y_test, y_pred_refined)
roc_auc_refined = roc_auc_score(y_test, y_pred_proba_refined)
mcc_refined = matthews_corrcoef(y_test, y_pred_refined)
conf_matrix_refined = confusion_matrix(y_test, y_pred_refined)
precision_refined = precision_score(y_test, y_pred_refined)
recall_refined = recall_score(y_test, y_pred_refined)
```

Summary:
Refined Parameters: The model was fine-tuned to find the best settings.
Best Model: The optimized model was selected for testing.
Predictions: The model was tested on unseen data, predicting stroke cases and their probabilities.
Evaluation: Multiple metrics were calculated to check how good the model is at identifying strokes.
Results: Everything was printed in an organized way to summarize the model's performance.

Feedforward Neural Network (FNN) (shared model)

We chose FNN for its ability to learn from unseen data when an FNN is trained well with the selection of hyper parameters, and the choice of the proper optimization algorithm. Below are some points which played a role in selecting FNN for our dataset and stroke prediction objective.

1. Suitability for complex and nonlinear data and simple to use: FNNs are well-suited for tasks involving complex and nonlinear data, such as medical datasets where factors influencing stroke are not linearly correlated. FNNs can model these nonlinear relationships effectively using activation functions like ReLU, sigmoid, or tanh. FNNs can also handle a variety of data types, including tabular data, which is common in stroke prediction datasets (e.g., patient demographics, medical history, lifestyle factors, etc.).

2. Flexibility in Model Design: FNNs allow customization in terms of the number of layers, neurons, and activation functions, enabling fine-tuning depending on the complexity of the dataset

Model Implementation

Implementation of FNN model

Before implementing the model, I did a bit of data exploration.

Data Exploration

1. Loading the Dataset and Initial Examination:
    a. Head and Info: Functions like.head() and .info() are used to preview the first few rows and check data types, non-null counts, and column names.
    b. Summary Statistics: Using .describe(), the dataset's numerical columns are summarized (mean, median, standard deviation, etc.).

2. Exploratory Data Analysis (EDA):
    a. Dataset Overview
        - The dataset contains 1456 rows and 27 columns.
        - There were no missing values, ensuring a complete dataset for analysis and modeling.
        - Column Types: Categorical Features: Variables such as gender, hypertension, heart_disease, ever_married, and Residence_type.
        - Numerical Features: Variables such as age, avg_glucose_level, and bmi.
        - Target Variable: stroke (binary classification: 0 = No Stroke, 1 = Stroke).

    b. Data Balance
    The target variable stroke is balanced, with 50% of the samples representing each class (stroke = 0 and stroke = 1). This balance ensures that the model will not be biased toward one class during training.

    c. Correlation with Target Variable:
    Positive correlations (closer to +1) indicate features strongly associated with stroke occurrence:
    age: Correlation coefficient = 0.586.
    71_80: Correlation coefficient = 0.361.
    ever_married: Correlation coefficient = 0.266.

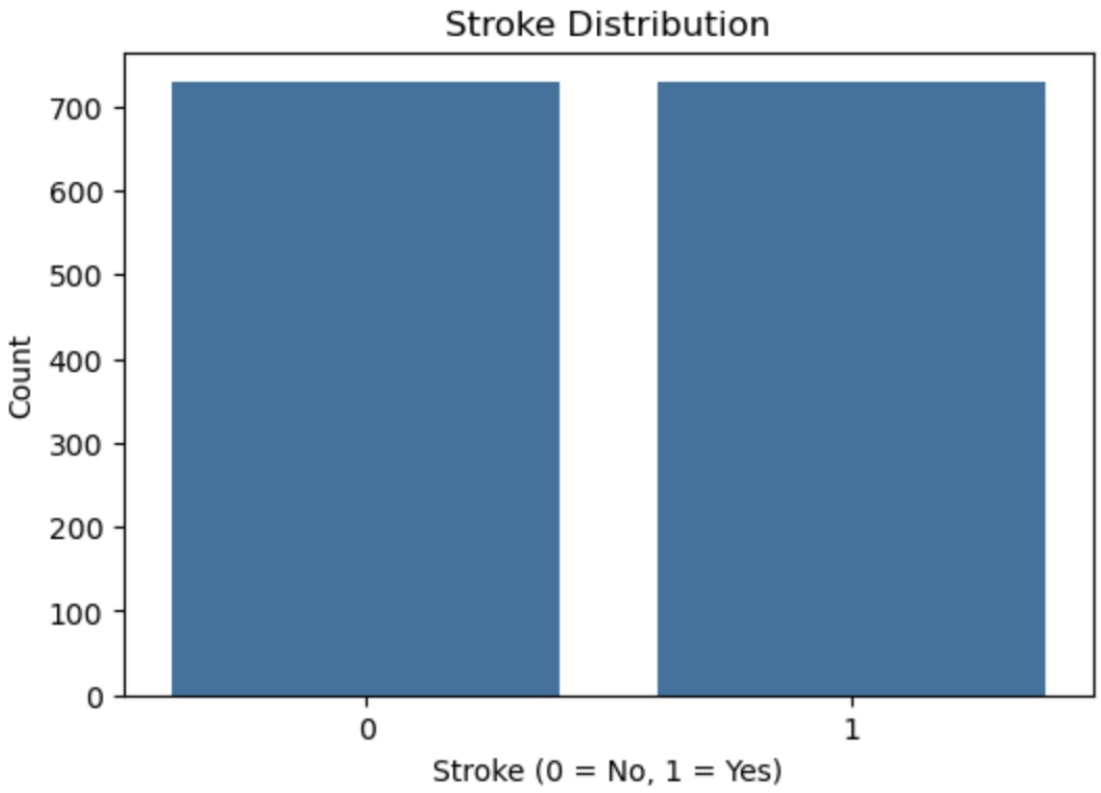
avg_glucose_level: Correlation coefficient = 0.250.

Interpretation

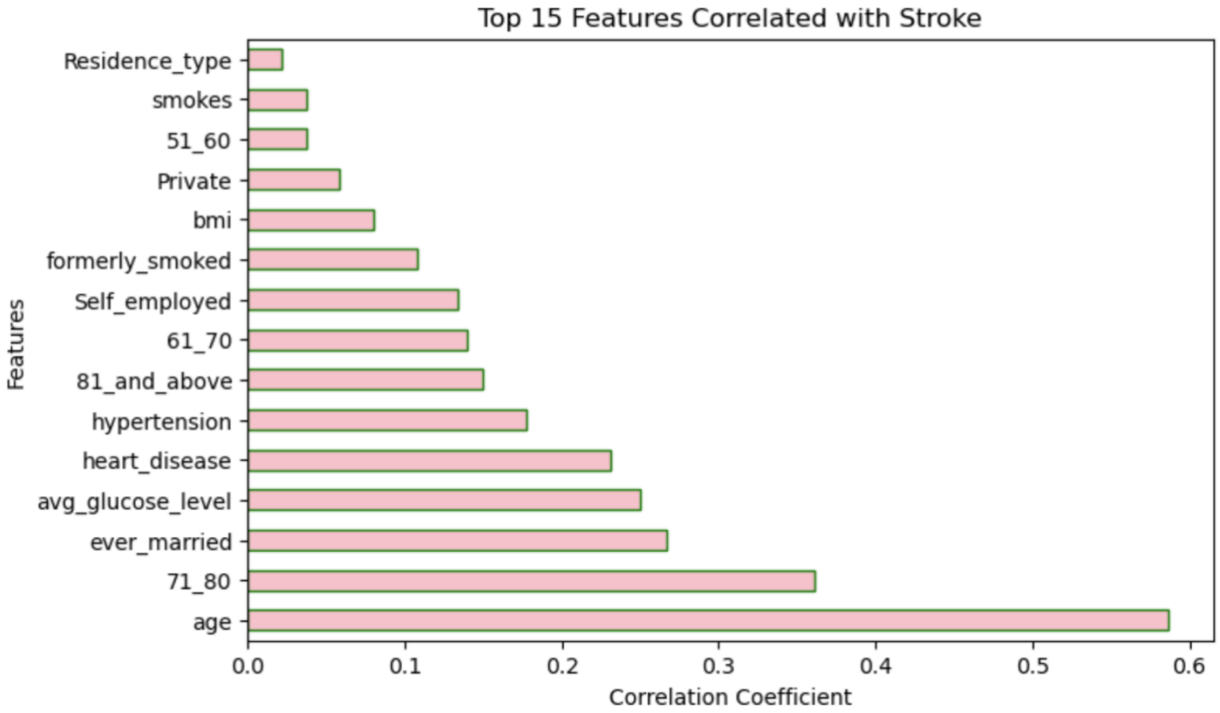
- Older individuals (especially aged 71 and above) and those with higher glucose levels are more likely to have a stroke.
- Being married also shows a moderate correlation, possibly reflecting lifestyle or health-related factors.

Visualizations

1. Distribution of Target Variable:
A count plot visualizes the distribution of stroke (0 vs. 1), confirming the balanced nature of the dataset.



2. Correlation Bar chart:



A bar chart to visualize the relationships between all features and the target variable. Features with higher correlation values are strong predictors of stroke.
The balanced dataset and clear correlations between features and stroke occurrence suggested good potential for accurate modeling. The stroke dataset was already preprocessed in R and loaded in python notebook for modeling hence there was no need for preprocessing steps like scaling or encoding categorical variables.

Feedforward Neural Network (FNN) Implementation for Stroke Prediction

1. Introduction

The implementation of a Feedforward Neural Network (FNN) for stroke prediction involves preprocessing the dataset, building the model, and tuning hyperparameters to optimize performance. Below is a detailed explanation of the steps and decisions involved in building the model.

2. Preprocessing the Dataset

Dataset Splitting:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

The dataset is split into training (70%) and testing (30%) sets with stratification to maintain the class distribution.

Feature Normalization:

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

StandardScaler is used to normalize features for stable training.

3. Building the FNN Model

Key Decisions:

- Hyperparameter Tuning: The number of layers, number of units, activation functions, dropout rates, optimizer, and learning rate are all selected dynamically to optimize the model's performance.
- Activation Functions: A variety of activation functions are available to introduce non-linearity into the model and help it learn complex patterns.
- Dropout: Dropout layers are used to prevent overfitting, especially when training on limited data.
- Optimizers: Different optimizers are available, allowing the model to choose the most effective method for weight updates based on the data and task.
- Binary Classification: The model is designed for binary classification, with a sigmoid output layer and binary crossentropy loss.

Using keras Tuner as a hyperparameter tuning library

Hyperparameter tuning is crucial for improving a model's performance and overall learning process by finding the best combination of parameters like the number of layers, learning rate, batch size, and activation functions.
I have used Keras Tuner because it is easy to implement and select the best hyperparameters through it for neural networks. It uses Random Search to test random hyperparameter combinations.
For hyperparameter tuning, the search algorithm used is RandomSearch.
I used random search as it is best suited for this binary classification problem.

Model Initialization:

```
model = Sequential()
```

The model is initialized as a Sequential model. A Sequential model is a linear stack of layers where each layer has exactly one input and one output. This structure is commonly used for simple neural networks where layers are added one after another.

Input Layer:

```
model.add(Input(shape=(X_train.shape[1],)))
```

The input layer is added with the shape determined by the number of features in the training data (X_train.shape[1]). This ensures that the model knows how many input features it will receive. The input layer doesn't have any learnable parameters; it simply serves as a placeholder for the data that enters the model.

Hidden Layers:

```
for i in range(hp.Int('num_layers', 2, 6)):
model.add(Dense( units=hp.Int(f'units_{i}', min_value=32, max_value=256, step=32), activation=hp.Choice('activation', ['relu', 'tanh', 'sigmoid', 'elu'])
 ))
model.add(Dropout(rate=hp.Float(f'dropout_rate_{i}', min_value=0.3, max_value=0.5, step=0.1)))
```

The number of hidden layers is dynamically selected using Hyperparameter Tuning. For each hidden layer, the model dynamically tunes:
Number of Layers: Between 2 and 6.
Neurons per Layer: Between 32 and 256, in steps of 32.
Activation Functions: ReLU, Tanh, Sigmoid, or ELU.
Dropout Rate: Between 0.3 and 0.5 to prevent overfitting.

Dropout Layer:

```
model.add(Dropout(rate=hp.Float(f'dropout_rate_{i}', min_value=0.3, max_value=0.5, step=0.1)))
```

A Dropout layer is added after each hidden layer to prevent overfitting. Dropout works by randomly “dropping” (i.e., setting to zero) a fraction of the neurons during each training iteration. The fraction of neurons to be dropped is dynamically chosen with values between 0.3 and 0.5, in steps of 0.1. This helps the model generalize better by reducing reliance on any single neuron.

Output layers:

```
model.add(Dense(1, activation='sigmoid'))
```

The output layer consists of a single neuron with a sigmoid activation function. The sigmoid function outputs a value between 0 and 1, making it suitable for binary classification tasks, where the model predicts a probability of the input belonging to one of two classes.

Optimizer and learning Rate:

```
optimizer = hp.Choice('optimizer', ['adam', 'sgd', 'rmsprop', 'adamx']) learning_rate = hp.Choice('learning_rate', [5e-3, 1e-3, 5e-4, 1e-4])
```

The model tunes the optimizer (Adam, SGD, RMSprop, Adamax) and learning rate to improve training.

4. Model Compilation

```
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
```

The model uses:
Binary Crossentropy: Suitable for binary classification.
Accuracy: As the performance metric.

5. Hyperparameter Tuning

Hyperparameter tuning is performed using Keras Tuner to find the optimal model configuration.
Random Search:

```
tuner = RandomSearch(
build_model,
objective='val_accuracy',
max_trials=20,
 executions_per_trial=2,
```

directory='tuner_results',
project_name='stroke_prediction')

Objective: Maximize validation accuracy.
Trials: Test 20 different configurations, with each configuration executed twice.
The tuner splits 20% of training data for validation and trains for 15 epochs per trial.
tuner.search(X_train, y_train, epochs=15, validation_split=0.2, batch_size=64)

6. Training and Evaluation:
Get best hyperparameters
After tuning, the best model is trained and evaluated.
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]

7. Build and Train Best Model:
best_model = tuner.hypermodel.build(best_hps) history = best_model.fit(X_train, y_train, epochs=30, validation_split=0.2, batch_size=64, verbose=1)

8. Evaluate Model:
The model's predictions are thresholded at 0.5 to classify as stroke or no stroke.
predictions = (best_model.predict(X_test) > 0.5).astype(int)

Evaluation Metrics
Metrics used are Accuracy, Confusion matrix, Precision, Recall, ROC AUC Curve and (MCC) Mathews correlation coefficient. These were calculated before improving the model and after applying tuned parameters to compare performance.

9. Improving the model for overfitting
To enhance the performance and generalization of the model we made following changes to our FNN model .The updated approach introduced advanced techniques, including regularization, batch normalization, additional hyperparameter tuning, and callbacks, to improve the model's generalization, stability, and performance.

1. Regularization for Overfitting Mitigation
   - L1 (Lasso) and L2 (Ridge) regularization techniques were incorporated into the dense layers.
   - Regularization penalizes large weights, reducing the likelihood of overfitting.

regularizer_type = hp.Choice('regularizer_type', ['l1', 'l2']) if regularizer_type == 'l1': regularizer = l1(hp.Float('l1_value', min_value=0.001, max_value=0.01, step=0.002)) else: regularizer = l2(hp.Float('l2_value', min_value=0.001, max_value=0.01, step=0.002))

- Hyperparameters for regularization strength (l1_value or l2_value) are dynamically tuned.
- L1 sparsifies weights, while L2 minimizes large weights.

2. Batch Normalization for Stable Training
model.add(BatchNormalization())

- Applied after each dense layer, Batch Normalization ensures consistent input distributions for subsequent layers.
- Batch Normalization normalizes layer outputs to have zero mean and unit variance.

3. Enhanced Hyperparameter Tuning

- Increased the search space for:
- Number of Neurons: From 32 to 512.
- Dropout Rate: From 0.3–0.5 to 0.3–0.6.
- Regularization Type and Strength: Added L1/L2 options.
- Optimizers: Included nadam in addition to adam, sgd, rmsprop, and adamax.
- Increased Trials: to 30

4. Advanced Callbacks for Robust Training
Three callbacks were added to improve training efficiency and performance.

- Early Stopping:
  EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

  Stops training if the validation loss does not improve for 5 consecutive epochs, preventing overfitting.

- Learning Rate Scheduler:
  ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=1e-6)

  Reduces the learning rate by a factor of 0.5 if the validation loss plateaus for 3 epochs, enabling finer weight updates.

- Model Checkpoint:
  ModelCheckpoint('best_model.keras',monitor='val_accuracy',
  save_best_only=True, mode='max')
  Saves the model with the best validation accuracy during training.

5. Increased Training Epochs and Validation
history2 = best_model.fit(X_train, y_train, epochs=50, validation_split=0.2, batch_size=64, verbose=1, callbacks=[early_stopping, lr_scheduler, checkpoint])

- Increased training epochs to 50 (previously 30) for improved learning.
- Incorporated a 20% validation split for better model evaluation during training.

6. Additional Optimizer Option

- Added Nadam (Nesterov-accelerated Adaptive Moment Estimation) optimizer, which combines the benefits of Nesterov momentum and Adam.

Benefits of These Improvements

The enhanced model incorporates advanced regularization, normalization, and hyperparameter tuning, supported by efficient training strategies through callbacks. These improvements collectively enhance the model's stability, generalization, and performance in stroke prediction. The improved model now includes:
Batch Normalization: Normalizes activations, which stabilizes training and helps prevent overfitting.
Dropout Layers: Randomly disables neurons during training, improving generalization.
L1/L2 Regularization: Penalizes large weights, reducing model complexity.
Callbacks:
Early Stopping: Prevents overtraining by stopping when validation performance stops improving.
Learning Rate Scheduler: Dynamically adjusts the learning rate to refine training.
These additions help the model generalize better, as shown by the metrics comparison in visualizations as shown in the evaluation and visualization section below.

Best Hyperparameters found

Layer Details:

Layer	Neurons	Activation	Dropout Rate	Regularizer	Regularizer Value
Layer 1	384	tanh	0.3	l2	0.001
Layer 2	384	tanh	0.3	l2	0.001
Layer 3	32	tanh	0.3	l2	0.001
Output Layer	1	sigmoid	N/A	None	N/A

Training Parameters:

Parameter	Value
Test Size	0.3
Validation Split	0.2
Epochs	50
Patience	5
Batch Size	34
Learning Rate	0.005
Optimizer	adamax

3.3 Ximena Michelle Castro Villalobos [6515820]

Modelling:

For the modelling section, the processed dataset was used. It was split in 80% for training and 20% for testing and the two chosen models were SVM with optimization algorithm and FNN with hyperparameters tuning.

Support Vector Machine (unique model)

Support Vector Machines (SVM) were chosen as the individual modeling technique due to their efficiency with smaller dataset sizes. Although SVM can become computationally expensive with larger datasets, the manageable sample size of this dataset made it an appropriate choice. SVMs are inherently designed for binary classification tasks, which aligns well with the binary nature of the target variable, "stroke."

SVM excels in handling high-dimensional datasets, as it effectively maximizes the margin between classes. This capability is particularly beneficial for this dataset, given its multiple features and the need to operate in a high-dimensional space. By selecting the right parameters—such as C, gamma, and kernel—SVM can significantly enhance performance.

To optimize the SVM model, three different techniques were employed: polynomial kernel adjustments, hyperparameter tuning with GridSearch, and Bayesian optimization using the Optuna library. These were then compared to find the one that tuned the best hyperparameters for the model. To model non-linear relationships, a polynomial kernel with degree 3 was used. This degree was chosen to strike a balance between underfitting (common with lower degrees) and overfitting (common with higher degrees), making it an efficient compromise for this dataset's complexity. GridSearch conducted an exhaustive search over predefined hyperparameter combinations and identified C=10, gamma=0.1, and kernel=rbf as the best parameters, achieving an accuracy and F1 score of 0.88. However, this approach was computationally expensive. Bayesian optimization, on the other hand, used the Tree-structured Parzen Estimator (TPE) in Optuna to probabilistically explore hyperparameter space based on prior results. This method efficiently identified the optimal parameters—C= 59.91, gamma=0.05, and kernel=rbf—which outperformed the GridSearch and polynomial kernel results, achieving the highest accuracy of 0.88 and MCC of 0.763. By comparing these techniques, Bayesian optimization proved to be the most efficient and effective approach for tuning the SVM model.

A key advantage of SVM lies in its robustness to noise, an important factor considering the use of SMOTE to balance the dataset. While synthetic data can introduce noisiness

and outliers, SVM maximizes the margin and ignores points close to it, ensuring noise resilience. Additionally, SVM's typical disadvantage in handling class imbalance was mitigated by preprocessing the dataset to achieve balanced classes. This allowed SVM to focus on finding an optimal decision boundary without bias toward the majority class.
Despite its strengths, SVM can struggle with complex feature relationships, where more flexible models like Feedforward Neural Networks (FNNs) might excel. For this reason, FNN was selected as the group model, leveraging its ability to model intricate non-linear patterns in the data more effectively than SVM.

Feedforward Neural Network (shared model)

Feedforward Neural Networks (FNNs) are well-suited for this dataset due to their ability to model non-linear relationships and capture complex interactions between variables such as "age," "BMI”," "glucose levels," and "smoking status." Unlike simpler models, FNNs automatically learn feature interactions without requiring explicit engineering, making them effective in uncovering intricate patterns relevant to stroke prediction. Additionally, FNNs perform well with larger datasets, and the use of SMOTE to balance the target classes enhances their scalability and robustness by addressing class imbalance.

The flexibility of FNN architecture was another advantage, as it allowed customization of the number of layers and neurons to match the dataset's complexity. For example, a shallow network can suffice for simpler relationships, while a deeper network is better equipped for more complex patterns. Probabilistic outputs from FNNs provided confidence levels for predictions, which is particularly useful in medical contexts where understanding the certainty of predictions can influence decision-making.

Hyperparameter tuning was crucial to maximize the performance of FNNs. GridSearch was employed to identify the best model parameters, such as activation functions, dropout rates, and optimizers, by systematically exploring various configurations. This approach led to significant improvements in model performance compared to a basic FNN. The downside to this approach is the computational cost and execution runtime. To avoid this the parameter “n_jobs” in GridSearch was set to -1 to enable parallelization and therefore speed up the search, however it caused a warning due to a short worker timeout this might be due to computational cost issues such as memory constraints. For this, an extra parameter was added: pre_dispatch="2*n_jobs". This adjusts parallelization by dispatching only twice the number of jobs as workers. However, after this customization the warning was still persisting, disabling parallelization was also tried but the runtime was far too long. Therefore, it was kept with n_jobs = -1 and assumed that it affected the model performance. A custom Keras wrapper classifier was built as the Keras

classifier package from the Keras library was not working for me. This enabled GridSearch, ensuring the model was optimally tuned for the dataset. The Best Parameters obtained with GridSearch for this model were:
activation: 'tanh'
batch_size: 16
dropout_rate: 0.0
epochs: 20
neurons: 64
optimizer: 'adam'

The activation function tanh was chosen as it performs particularly well with normalized data as in this case.

Smaller batch sizes often lead to more stochastic updates, which can escape shallow local minima and enable better generalization. A batch size of 16 provided a good balance between computation speed and generalization.

A dropout rate parameter of 0.0 indicated that the grid search found the model did not suffer from overfitting, and it therefore did not need to be modified.

Grid search found 20 epochs as the best parameter for the model, as it is likely that the model's loss validation stabilized at this point. Therefore, showing the point where the model has learned sufficient patterns from the data without overfitting

64 neurons were used in the dense layer to enable enough representational power. This captured the complexity of the data without being excessively large, which could lead to overfitting or computational inefficiency.

The Adam optimizer combined the benefits of RMSprop and momentum, adapting the learning rate for each parameter dynamically. This helped the model converge faster and more effectively.

Despite their strengths, FNNs showed some challenges. They require a large amount of data for effective training, and overfitting can occur if the dataset is small, even after SMOTE. Additionally, FNNs lack the interpretability of simpler models like SVM, which can be a limitation in medical applications. However, the use of interpretable models (SVM) beforehand helped mitigate this issue in this case. In this research the tuned FNN model provided a powerful and flexible framework for the predictive modeling of this dataset.

3.4 C Ashwini Reddy [6886963]

eXtreme Gradient Boosting Model (unique model)

It is short for short for eXtreme Gradient Boosting. It builds a sequence of decision trees, where each tree corrects the errors of the previous ones by minimizing a specified loss function.

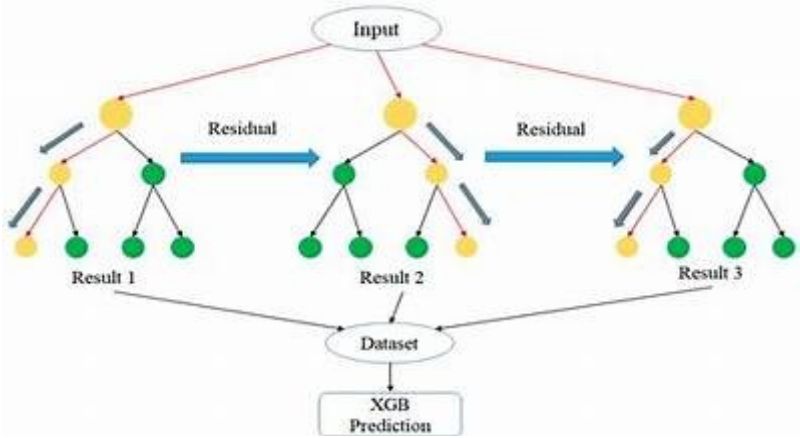


Figure 1: XGBoost Model (Wang, et al., 2022)

Architecture of XGBoost

It follows the principle of Gradient Boosting, which is an ensemble technique that combines weak learners (usually decision trees) in a sequential manner:

- Each tree corrects the errors of its predecessor by minimizing a loss function.
- Predictions are updated iteratively by adding the output of each new tree to the ensemble.

Major Strengths of XGBoost for Medical Datasets

- Handling Imbalanced Data: It provides us options such as weighted classes and customized loss functions, making it highly effective in handling missing and imbalanced dataset.
- Feature Importance and Interpretability: XGBoost computes feature importance scores, allowing us to understand which medical contribute most to predictions.

Major Weaknesses of XGBoost for Medical Datasets

- Data Preprocessing and Sensitivity to Quality: XGBoost can handle some missing values, but may require significant preprocessing to ensure data quality, which can be time-consuming.

- Risk of Overfitting: Due to its high flexibility and capacity to learn complex patterns, XGBoost is prone to overfitting.

Parameter tuning:

Parameters	Range of typical values	My model	Why I used this value
n_estimators	[50 - 1000]	[50, 100, 150]	Range from smaller to moderate values to balance computational and cost of the model
Max_depth	[2, 3, 5, 7, 10, 15]	[3, 5, 7]	Typical value to capture meaningful interactions without overfitting noise
Learning_rate	[0.001, 0.01, 0.05, 0.1, 0.2, 0.3]	[0.01, 0.1, 0.2]	Medical dataset benefit from smaller rates due to potential noise
subsample	[0.5, 0.6, 0.7, 0.8, 0.9, 1.0]	[0.8, 1.0]	Use 80 percent of dataset for more regularization and 100 for no randomness
Colsample_bytree	[0.3, 0.5, 0.7, 0.8, 0.9, 1.0]	[0.8, 1.0]	80 percent to reduce complexity and improve generalization and 100 percent use of all features

Training setting used in my model:

I have used GridSearchCV with the following parameters:

- scoring='roc_auc': Used to optimize Area Under the Receiver Operating Characteristic Curve (AUC) ensuring that the model performs well in distinguishing between classes, critical in medical datasets.
- cv=3: To ensure that the model generalizes well and avoids overfitting by training and testing on different splits of the data.
- verbose=1: Model provides updates during training.
- n_jobs=-1: Utilizes all available CPU cores to speed up the grid search, as testing multiple parameter combinations is expensive.

Code:

```
param_grid = {
    'n_estimators': [50, 100, 150],    # Number of trees in the ensemble
    'max_depth': [3, 5, 7],    # Max depth of each tree
    'learning_rate': [0.01, 0.1, 0.2],    # Step size for weight updates
    'subsample': [0.8, 1.0],    # Fraction of sample used to train each tree
    'colsample_bytree': [0.8, 1.0]    # Fraction of feature used to train each tree
}
grid_search = GridSearchCV(estimator=xgb, param_grid=param_grid,
                           scoring='roc_auc', cv=3, verbose=1, n_jobs=-1)
grid_search.fit(X_train, y_train)
```

Code Explanation:

param_grid: A dictionary that defines the hyperparameters to tune.

- n_estimators: The number of trees in the ensemble.
- max_depth: The maximum depth of each tree.
- learning_rate: The step size used to update the weights.
- subsample: The fraction of samples to use for fitting each tree.
- colsample_bytree: The fraction of features to use for each tree.

GridSearchCV: A search over all combinations of the parameters specified in param_grid to find the best combination based on a cross-validation score.

- estimator=xgb: The model that GridSearchCV will optimize, assumed to be an instance of xgboost.XGBClassifier or a similar model.
- param_grid: The grid of hyperparameters to test.
- scoring='roc_auc': The scoring metric used to evaluate each combination of parameters. Here, you are using the ROC AUC score.
- cv=3: The number of cross-validation folds (i.e., 3-fold cross-validation).
- verbose=1: This means the grid search will provide progress output (increased verbosity).
- n_jobs=-1: This enables parallel processing, using all available cores.
- grid_search.fit(X_train, y_train): Finally, you fit the grid search to the training data X_train and labels y_train, which will perform the hyperparameter tuning.

Feedforward Neural Network (Shared Model)

A Feedforward Neural Network consists of an input layer, one or more hidden layers and an output layer and the information flows in one direction without forming circles hence the name feedforward neural network (GeeksforGeeks, 2024).

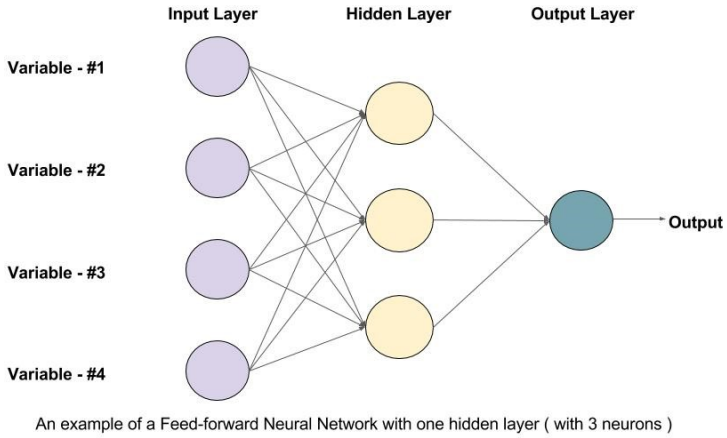


Figure 2: Feedforward Neural Network (Gupta, 2017)

Architecture of a feedforward neural network

- Input layer: It is the first layer that receives the raw input data, each node represents a single input feature.
- Hidden layer: These form the intermediate layer that process the input data, extract increasingly complex and abstract features.
- Output layer: Final layer produces the networks prediction or output.
- Neurons: It is present within different layer that receive the input from the previous layer, perform computation and pass it on to the next layer.
- Weights: Each connection between neurons has a weight associated with it which determines its strength and helps optimize a model.
- Bias: It acts as an offset to the weighted sum of inputs.
- Activation function: It helps to introduce non-linearity to the model enabling it to learn complex patterns. (Ohiri, 2024)

Major Strengths of FNNs with Medical Dataset:

- Ability to model complex relationships: Medical dataset have nonlinear relations between features and FNN excel at capturing those.
- Handling high dimensionality: FNN learns from diverse feature spaces without requiring explicit dimensionality reduction.

Major Weakness of FNNs with Medical Dataset:

- High Sensitivity to Data Quality: FNNs are highly sensitive to missing data, which is normal in medical dataset, we need to handling missing values carefully during preprocessing of data.
- Lack of Interpretability: FNNs function as "black-box" models, making it difficult to provide actionable explanations, which is a significant drawback in case of medical data.

Parameter tuning:

Parameter	Range of Typical Values	My model	Why I used this value
Number of Hidden Layers	2-3	2	To learn complex patterns and relationships in the data
Neurons per Layer	16-512	64 and 32	To help the model learn complex patterns from the data.
Dropout Rate	0.1-0.5	0.2	To help prevent overfitting. It is a very helpful technique while working with small datasets.
Activation Function	ReLU, Leaky ReLU, tanh	ReLU	It is computationally efficient and helps to avoid vanishing gradients during training.
Optimizer	Adam, RMSprop, SGD	Adam	Helps in faster converge and adapts to the learning rates for each parameter individually.
Learning Rate	0.0001-0.01	0.001	Ensures stable and efficient training.
Batch Size	16-128	34	Strikes a balance between efficiency and stability.

Training settings used in my model:

- Epochs = 40: The model tarins for up to 40 epochs but early stopping ensures that training is stopped if validation performance degrades.
- Batch size = 34: Using a moderate batch size strikes a balance between efficiency and stability during training.
- Validation Split = 0.2: 20% of the training data is held out (validation set) to monitor performance on the unseen data during training which helps in generalization ability of the model.
- Verbose = 1: The training progress is printed, which helps to monitor the training process in real-time.

- Early Callback = patience of 5 epochs: If the validation loss does not improve after % consecutive epochs the training will stop.

Hyperparameter training Explanation:

Code:

```
model = Sequential([
    Input(shape=(X_train.shape[1],)),  # Explicitly define input shape
    Dense(128, activation='relu'),  # First hidden layer with 128 neurons
    Dropout(0.2),  # Dropout layer to prevent overfitting
    Dense(64, activation='relu',),  # Second hidden layer with 64 neurons
    Dropout(0.2),  # Dropout layer
    Dense(1, activation='sigmoid')  # Output layer for binary classification (0 or 1)
])
```

Code Explanation:

Sequential Model: To help build a layer-by-layer structure model where each layer's output becomes the next layer's input.

Input(shape=(X_train.shape[1],)):

- It specifies the input shape, equal to the number of features in X_train.
- Ensures the model knows the dimensionality of the input data.

Dense (128, activation='relu'):

- First hidden layer with 128 neurons.
- The ReLU activation function introduces non-linearity, helping the model learn complex patterns.
- Higher number of neurons allows capturing more features and relationships in the input.

Dropout (0.2):

- Randomly drops 20% of neurons during training to prevent overfitting.
- Helps the model generalize better to unseen data.

Dense (64, activation='relu'):

- Second hidden layer with 64 neurons.
- Reduced number of neurons and balance complexity while avoiding overparameterization.

Another Dropout (0.2):

- This helps to add robustness to the model and prevents overfitting by again dropping 20% of neurons.

Dense (1, activation='sigmoid'):

- Final output layer with 1 neuron for binary classification.
- Sigmoid activation function squashes the output between 0 and 1, making it suitable for probability interpretation.

Code:

```
model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])
```

Code Explanation:
optimizer=Adam(learning_rate=0.001)

Adam Optimizer:

- Helps combines the benefits of RMSProp (adaptive learning rates) and Momentum (faster convergence).
- Dynamically adjusts the learning rate for each parameter during training.

Learning Rate:

- It is set to 0.001, which is a common default value for training neural networks.
- Controls the step size of weight updates. Smaller values ensure stable convergence but might slow down training, while larger values can speed up training but risk overshooting the optimal solution.

loss='binary_crossentropy'

- Suitable for binary classification tasks, where the output is either 0 or 1.
- Encourages the model to output probabilities close to the true labels.

metrics=['accuracy']

- Tracks the accuracy metric during training and validation.
- While accuracy is easy to interpret, it may not fully represent performance for imbalanced datasets (consider additional metrics like ROC AUC, precision, and recall).

Code:

```
early_stopping = EarlyStopping(
    monitor='val_loss',  # Monitor validation loss
    patience=5,          # Stop after 5 epochs if no improvement
    restore_best_weights=True  # Restore model weights from the best epoch
)
```

Code Explanation:

monitor='val_loss':

- Specifies which metric to monitor during training.
- val_loss refers to the validation loss, which is a key indicator of how well the model generalizes to unseen data.
- Early stopping halts training if the validation loss stops improving, preventing overfitting to the training data.

patience=5:

- Defines the number of epochs to wait for improvement before stopping.
- If the validation loss does not improve for 5 consecutive epochs, training stops.
- This ensures the model doesn't terminate training too early due to minor fluctuations in the loss.

restore_best_weights=True:

- After stopping, the model automatically reverts to the weights from the epoch with the lowest validation loss.
- This ensures the final model is the one with the best generalization performance, even if the last few epochs were less optimal.

Code:

```
early_stopping = EarlyStopping(
    monitor='val_loss',  # Monitor validation loss
    patience=5,          # Stop after 5 epochs if no improvement
    restore_best_weights=True  # Restore model weights from the best epoch
)
```

Code Explanation:

epochs=40:

- Provides sufficient room for training without overextending if early stopping is not triggered.

batch_size=34:

- A moderate batch size balances memory usage and training speed.

validation_split=0.2:

- Ensures the model's generalization is evaluated during training without needing a separate validation dataset.

callbacks=[early_stopping]:

- Prevents overfitting and saves training time by stopping when validation loss stagnates.

4. Performance Evaluation

4.1 Sampada Shrestha [6892246]

For this project, I implemented and evaluated two ML models; Random Forest and Feedforward Neural Network. The performance of both the models were evaluated using the key classification metrics such as accuracy, precision, recall, F1 score, Matthews Correlation Coefficient (MCC), and ROC AUC score. Accuracy measures the overall ability of the model to correctly predict the correct class. The confusion matrix represents the number of true positives, true negatives, false positives and false negatives, providing thorough understanding of model's performance on each class. Precision calculates the proportion of true positives out of all the positive predictions and recall calculates the proportion of actual positives that were correctly classified as positive by the model. The F1 score is the harmonic mean of precision and recall. It combines the precision and recall values into a single value and provides a balance between them. The Matthews Correlation Coefficient (MCC) considers both true and false positives and negatives and is commonly used for health and medical datasets. The ROC AUC score assesses the model's ability to differentiate between the two target classes. All these metrics combined provide a comprehensive view of the model's performance, particularly its ability to correctly classify both the classes: Stroke and Not Stroke.
The performance of the both the Random Forest and the FNN model based on these evaluation metrics are discussed below.

Random Forest (unique model)
The performance of the Random Forest model before and after hyperparameter tuning based on the evaluation metrics is mentioned below.

Performance Before Hyperparameter Tuning

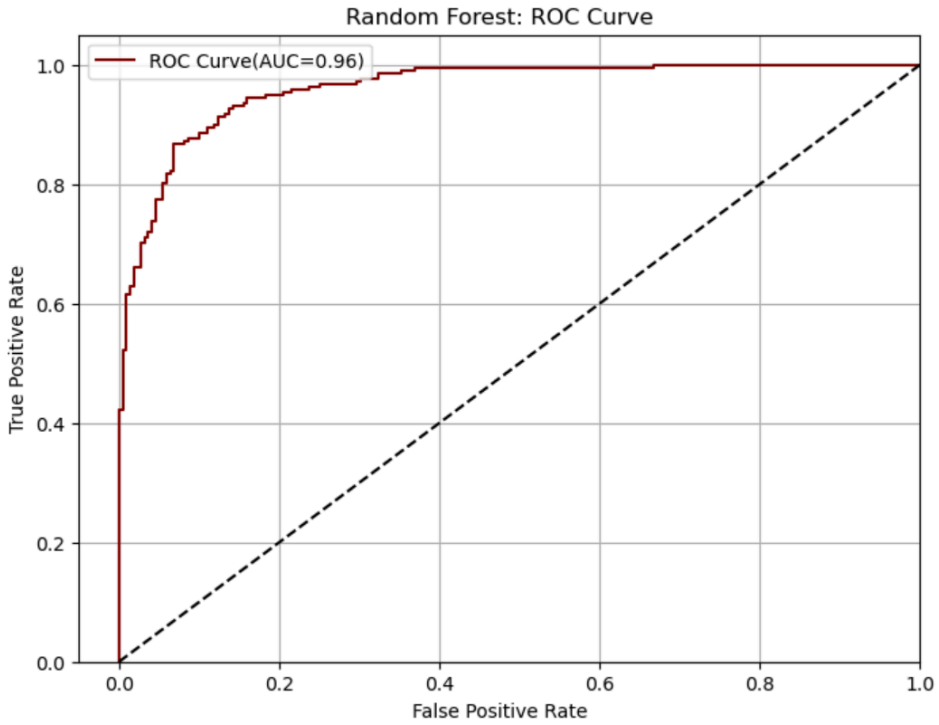
- Accuracy: 0.8627002288329519
- Precision: Class 0 [Not Stroke]: 0.94, Class 1 [Stroke]: 0.81
- Recall: Class 0 [Not Stroke]: 0.77, Class 1 [Stroke]: 0.95
- F1 Score: Class 0 [Not Stroke]: 0.85, Class 1 [Stroke]: 0.87
- Matthews Correlation Coefficient (MCC): 0.74
- ROC AUC Score: 0.9586527585773532

Performance After Hyperparameter Tuning

- Accuracy: 0.88
- Precision: Class 0 [Not Stroke]: 0.94, Class 1 [Stroke]: 0.84
- Recall: Class 0 [Not Stroke]: 0.82, Class 1 [Stroke]: 0.95

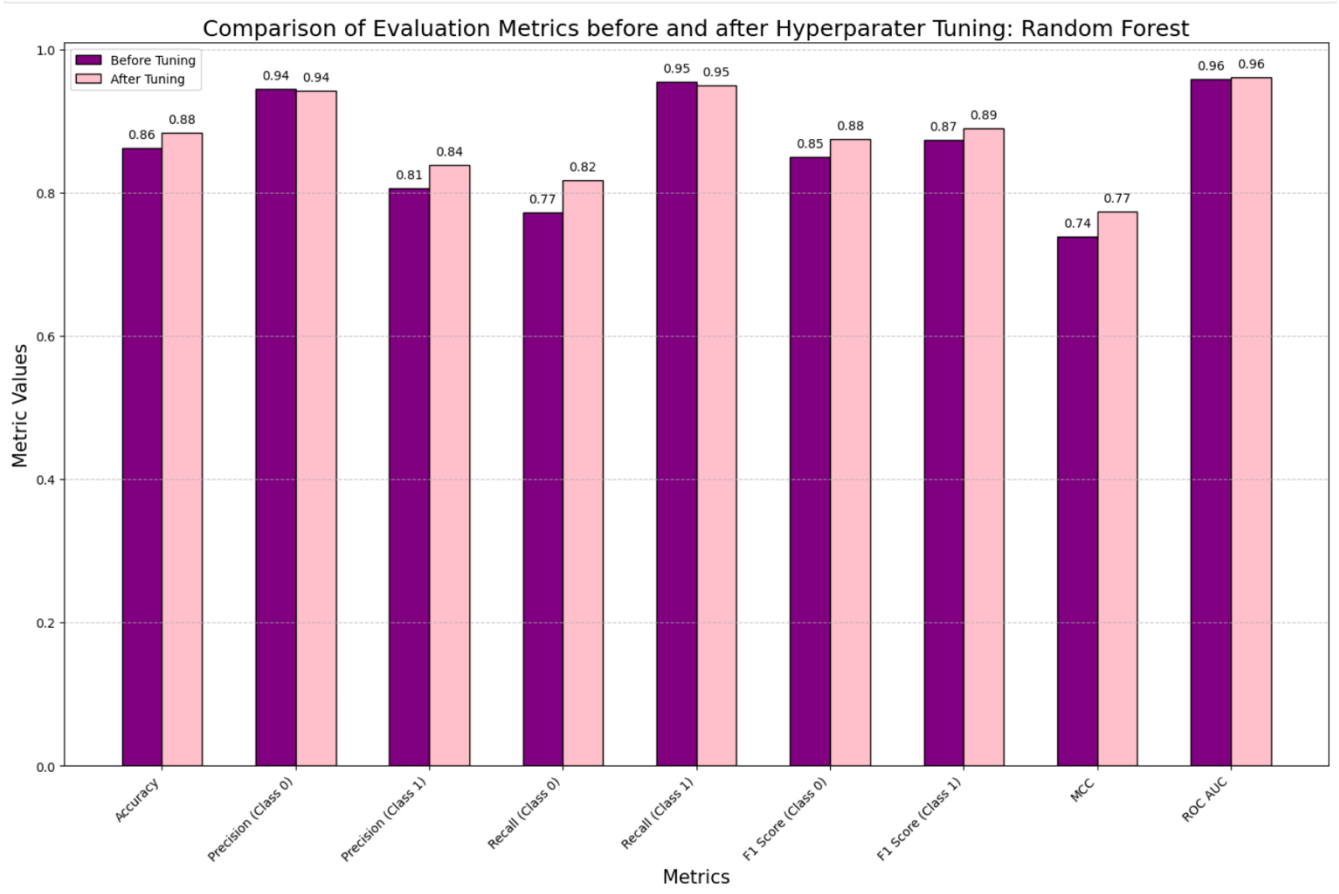
- F1 Score: Class 0 [Not Stroke]: 0.88, Class 1 [Stroke]: 0.89
- Matthews Correlation Coefficient (MCC): 0.77
- ROC AUC Score: 0.961250052364794

The ROC AUC curve visualization after hyperparameter tuning is shown below.

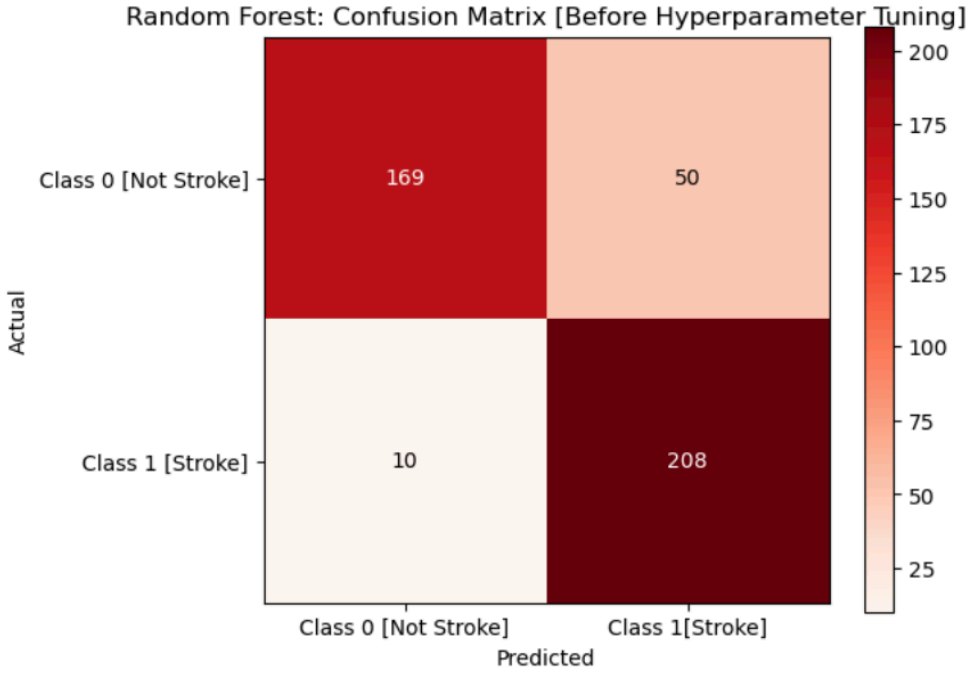


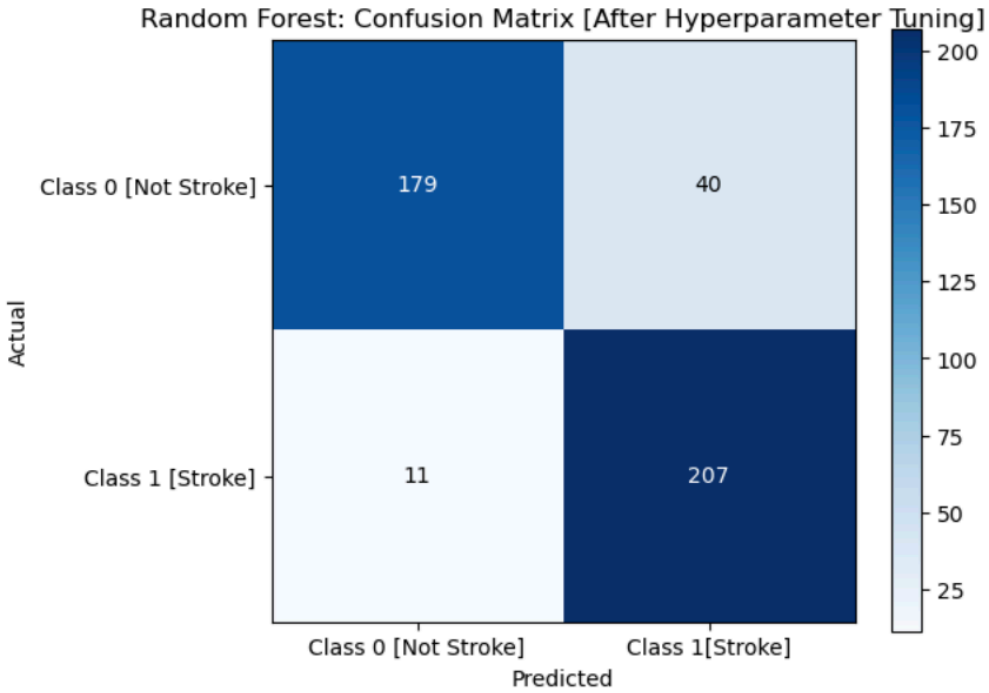
From the values above, it can be seen that after hyperparameter tuning, the Random Forest model performed better across several evaluation metrics. The overall model accuracy increased from 0.86 to 0.88, indicating that model's overall classification ability was enhanced. The precision score for class 1(Stroke) increased from 0.81 to 0.84, which shows that there were lower number of false positive predictions for class 1 after hyperparameter tuning. The recall score for class 0 (Not Stroke) improved from 0.77 to 0.82, indicating a decline in the number of false negative predictions. The F1 score for both the classes have also increased from 0.85 to 0.88 for class 0 and 0.87 to 0.89 for class 1, reflecting better overall balance between precision and recall. Along with that the MCC score also improved from 0.74 to 0.77, meaning that the model's prediction are more dependable and represent a better balance of classes. The ROC AUC score has only slightly increased from 0.985 to 0.961, which indicates that the model can efficiently differentiate between the two target classes.

The graph below summarizes the improvement in the Random Forest model's performance before and after hyperparameter tuning.



Additionally, there is an improvement in confusion matrix before and after hyperparameter tuning as well. The two diagrams below represent the confusion matrix before and after hyperparameter tuning.





From the two confusion matrices above we can we that the number of true positives increased from 169 to 179, the number of false positives decreased from 50 to 40. This improvement indicates a decrease in misclassifications, as the model performs better in differentiating between the two classes.

Feedforward Neural Network (shared model)

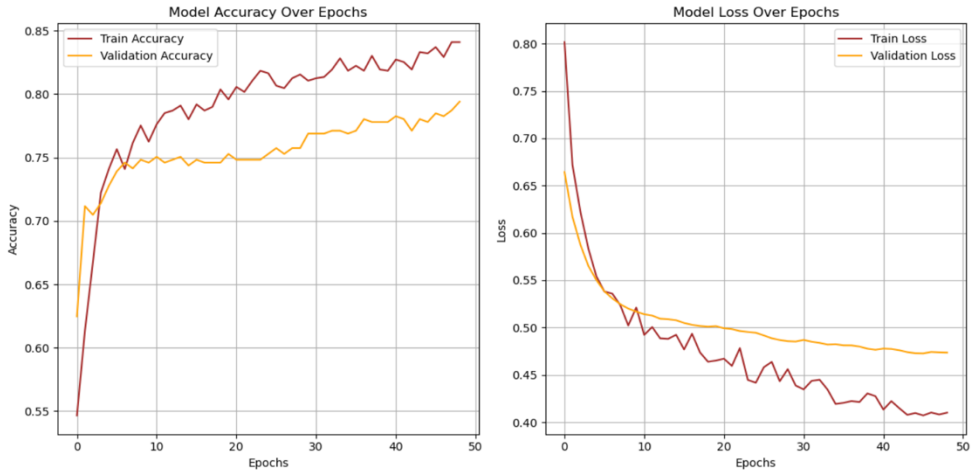
The performance of the FNN model based on the evaluation metrics is mentioned below.

- Model Accuracy: 0.78
- Precision: Class 0 [Not Stroke]: 0.83, Class 1 [Stroke]: 0.75
- Recall: Class 0 [Not Stroke]: 0.72, Class 1 [Stroke]: 0.85
- F1 Score: Class 0 [Not Stroke]: 0.77, Class 1 [Stroke]: 0.80
- Matthews Correlation Coefficient (MCC): 0.57
- ROC AUC Score: 0.7850425202128106

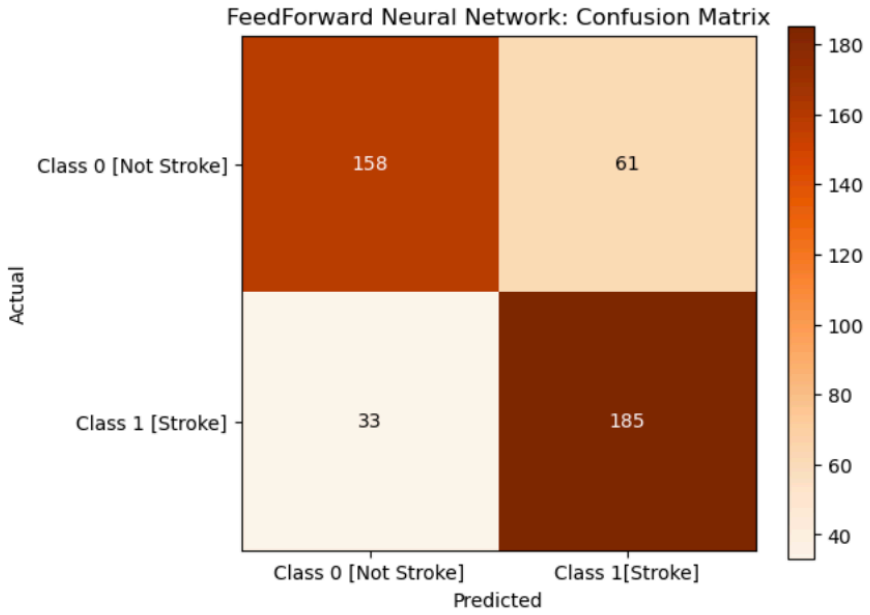
The model's performance after end of the 50 epochs based on training data and validation is shown below.

```
Epoch 49/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8551 - loss: 0.4086 - val_accuracy: 0.7941 - val_loss: 0.4735
```

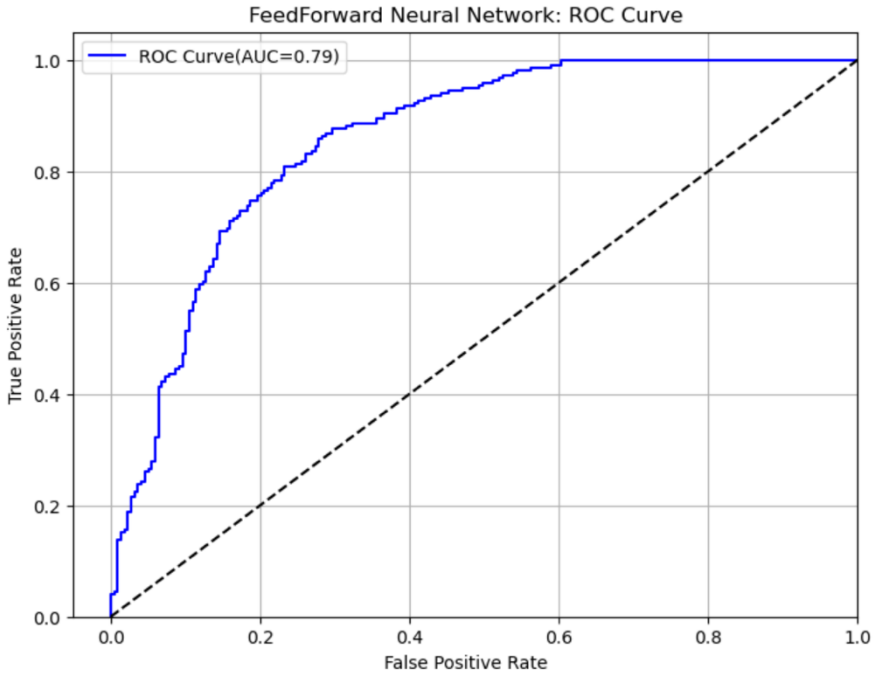
The model accuracy and model loss over training epochs were also visualzed to gain further insights insights on how the model is performing. The plot shows that the model is learning effectively, as seen by the consistent increase in training accuracy and decrease in training loss. However, there is a gap between the training and validation accuracy indicating that the model is slightly overfitting as model is performaing better on training data than validation data.



The confusion matrix for the FNN model is shown below.



The ROC AUC curve visualization is shown below.



The FNN model achieved a training accuracy of 0.8551 and a validation accuracy of 0.7941 after the end of 50 epochs. The relatively high accuracy score for both training and validation sets indicates that the model generalizes and performs well. The model accuracy score of 0.78 represents that the model accurately classified 78% of the data. This accuracy score is based on test data. The precision score for class 0 is 0.83 and class 1 is 0.75, suggesting that the model is more accurate in predicting class 0 than class 1. This means that the model makes fewer false postive predicition for class 0. The recall scores for class 0 is 0.72 and class 1 is 0.85, meaning the model makes fewer false negative predictions for class 1. The F1 score for class 0 and class 1 are 0.77 and 0.80 respectively, indicating that model is better at predictiing class 1. The MCC score of 0.57 indicates moderately positive correlation between the predicted and true values. The model has the ROC AUC score of 0.785, representing that it can distinguish between the two classes quite well.

Comparison of the Model's Performance (Random Forest and FNN)

The Random Forest model, after hyperparameter tuning, has consistently outperformed the FNN model across all evaluation metrics. The Random Forest has a higher score of 0.88 for overall model accuracy, compared to the accuracy score of 0.78 for FNN model. Along with accuracy, it also performs well in precision, recall and F1 scores, suggesting its ability to correctly predict stroke occurrences while maintaining a moderate balance between false positives and false negatives. Moreover, the Random Forest has a significantly higher MCC score of 0.77 compared to the MCC score of FNN (0.57), indicating that the predictions of the Random Forest model are more reliable. The Random Forest also have an ROC AUC score of 0.96 signifying that the model can efficiently differentiate between the two target classes. Thus, based on the results from this dataset, the Random Forest proves to be a more effective model for stroke prediction when compared to the FNN model. This could be due to the ability of Random Forest models to create several decision trees and combine their predictions, enabling them to handle complex datasets more effectively. This also makes Random Forest models less sensitive to overfitting, particularly on smaller to moderate sized datasets. Random Forest's tolerance to noisy data and its relatively simpler trainig process may have provided it with an advantage when working with the dataset chosen for this project.

4.2 Syeda Fizza Raza [6509217]

Evaluation: Use metrics such as Accuracy, Precision, Recall, F1-score, and AUC-ROC for comparison with FNN and other models to be consistent.

Why these evaluation metrics?

- Accuracy: Measures overall correctness but can be misleading for imbalanced datasets.
- F1 Score: Balances precision and recall, especially important for imbalanced datasets.
- ROC AUC: Evaluates the model's ability to distinguish between classes.
- Precision & Recall: Key for understanding false positives and false negatives, critical in medical predictions like stroke.
- Matthews Correlation Coefficient (MCC): A balanced measure even for imbalanced classes.

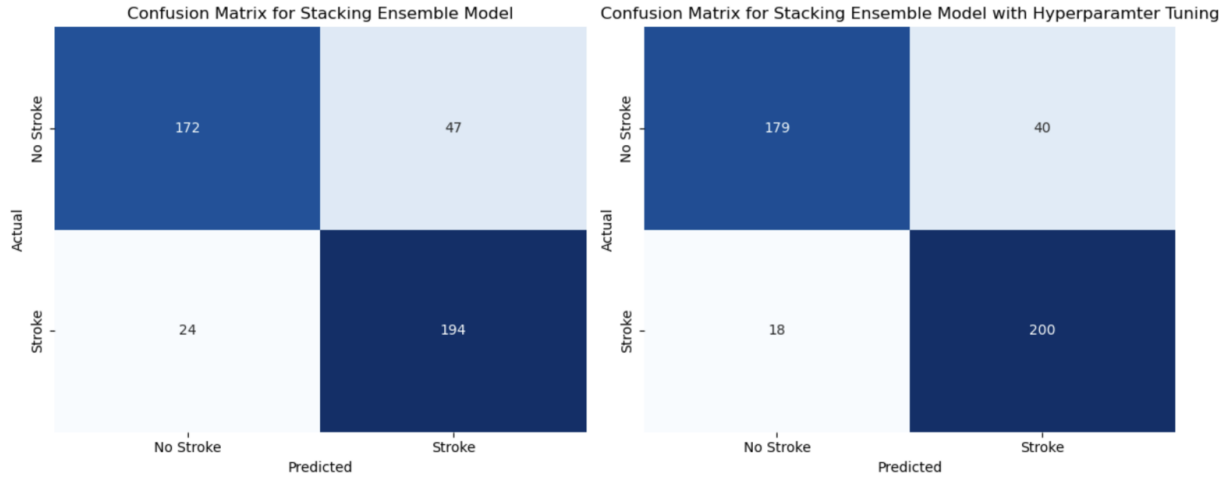
Stacking Ensemble Model (unique model)

Evaluation Metrics:
Both the metrics for first model and improved ensemble model were calculated to be compared.

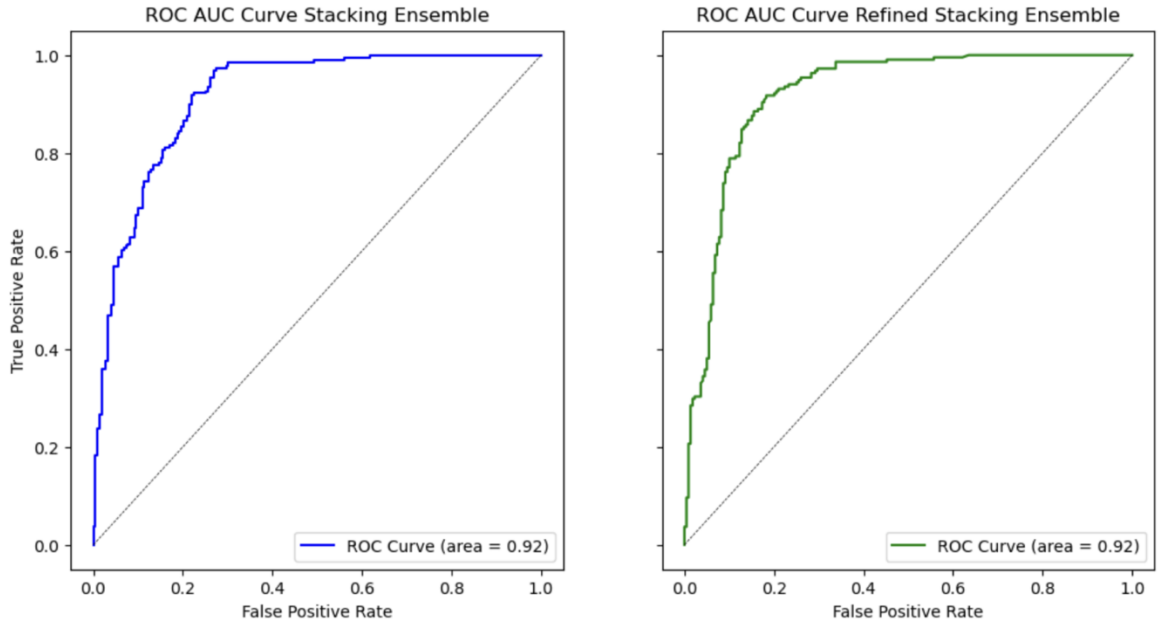
	Stacking ensemble Model	Improved stacking ensemble Model
Accuracy	0.8375286041189931	0.8672768878718535
Precision	0.8049792531120332	0.8333333333333334
Recall	0.8899082568807339	0.9174311926605505
F1 Score	0.8453159041394336	0.8733624454148472
ROC AUC	0.9159649784257049	0.9199865946127099
MCC	0.6789036943338258	0.7383640555162516

Visualization:

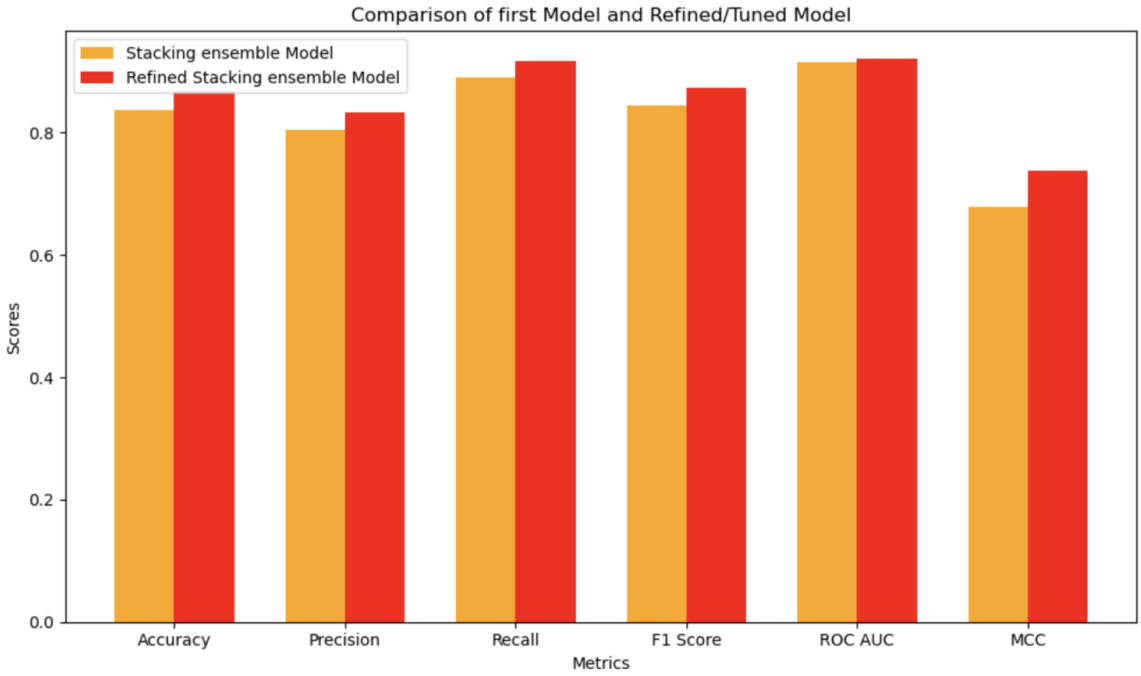
Confusion Matrix Heatmap:



ROC AUC Line curve:

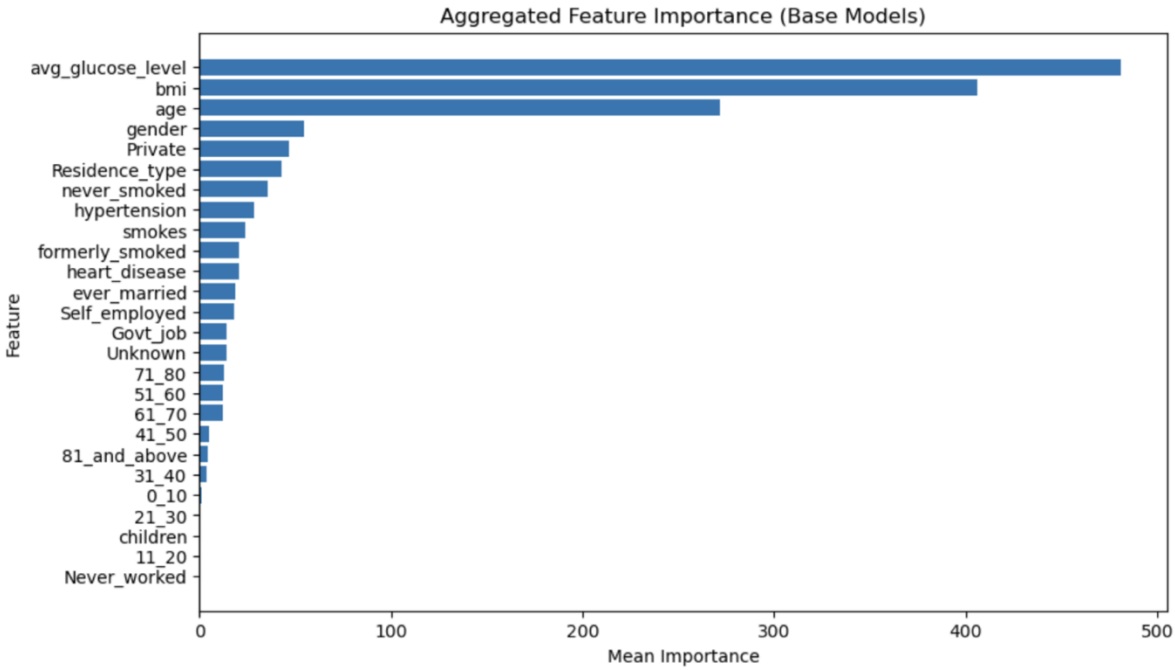


Bar Charts for model evaluation metrics:

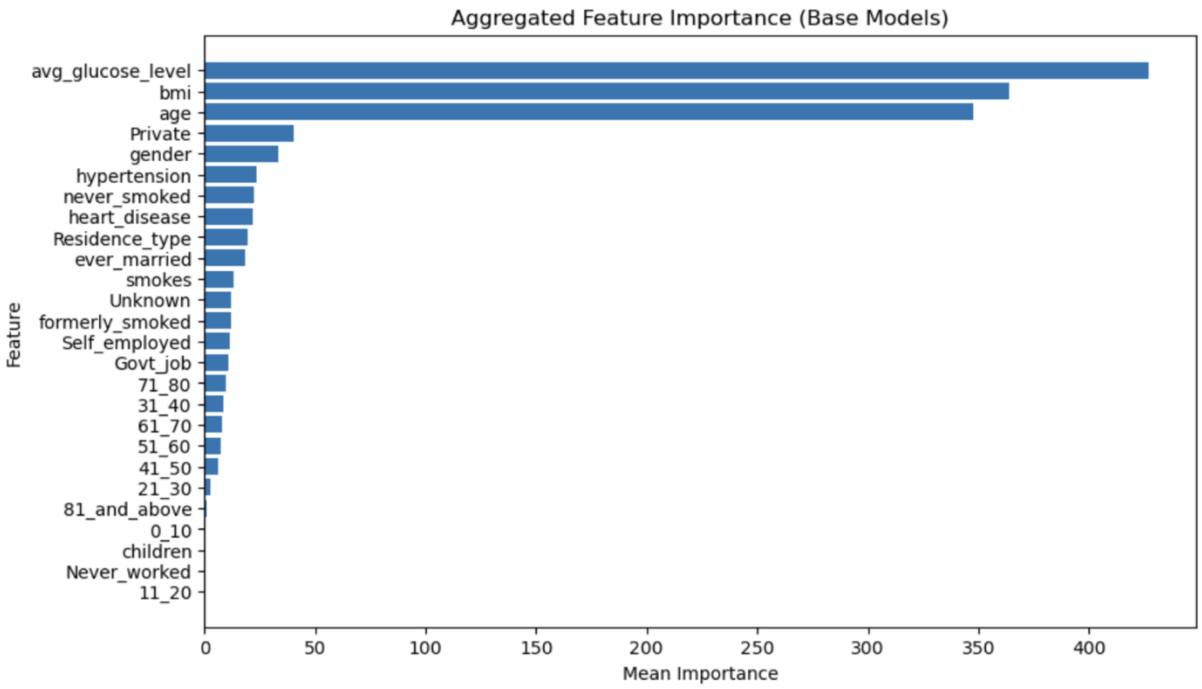


Feature Importance

Bar chart of feature importance calculated from first stacking ensemble model



Bar chart for feature importance calculated from improved stacking ensemble model



Top Features before Tuning:

```
Top Features Contributing to Stroke Prediction:
                    lightgbm   catboost  Mean Importance
avg_glucose_level        946  16.796492       481.398246
bmi                      796  16.279450       406.139725
age                      517  26.864923       271.932462
gender                   104   4.704465        54.352232
Private                   89   4.141284        46.570642
Residence_type            83   3.019266        43.009633
never_smoked              69   2.942167        35.971084
hypertension              54   2.504698        28.252349
smokes                    46   1.774249        23.887124
formerly_smoked           40   1.726274        20.863137
heart_disease             38   2.730603        20.365302
ever_married              35   2.661962        18.830981
Self_employed             35   1.622662        18.311331
Govt_job                  27   1.826829        14.413415
Unknown                   26   2.169479        14.084740
71_80                     24   1.236864        12.618432
51_60                     23   1.651358        12.325679
61_70                     22   1.967314        11.983657
41_50                      9   0.897008         4.948504
81_and_above               8   0.317595         4.158798
31_40                      7   0.703351         3.851675
0_10                       2   0.309372         1.154686
21_30                      0   0.539906         0.269953
children                   0   0.403145         0.201573
11_20                      0   0.194780         0.097390
Never_worked               0   0.014502         0.007251
```

Top Features after tuning:

Top Features Contributing to Stroke Prediction:

	lightgbm	catboost	Mean Importance
avg_glucose_level	842	12.528674	427.264337
bmi	711	17.408222	364.204111
age	648	47.113237	347.556618
Private	77	4.184422	40.592211
gender	65	2.034131	33.517065
hypertension	46	1.032032	23.516016
never_smoked	44	1.071189	22.535595
heart_disease	43	0.749173	21.874587
Residence_type	39	0.573572	19.786786
ever_married	37	0.287585	18.643793
smokes	25	1.534718	13.267359
Unknown	22	2.138901	12.069451
formerly_smoked	23	1.060980	12.030490
Self_employed	22	0.561548	11.280774
Govt_job	21	1.093763	11.046881
71_80	19	0.645167	9.822584
31_40	17	0.196918	8.598459
61_70	14	1.839284	7.919642
51_60	13	1.169462	7.084731
41_50	12	0.217849	6.108925
21_30	4	1.585753	2.792876
81_and_above	2	0.179141	1.089571
0_10	1	0.259473	0.629736
children	0	0.532951	0.266476
Never_worked	0	0.001854	0.000927
11_20	0	0.000000	0.000000

Top Features (Avg Glucose Level, BMI, Age):

Pre-Tuning:
"Avg Glucose Level" had a mean importance of 427.26, followed by BMI (364.20) and Age (347.55).

Post-Tuning:
The importance of these features increased significantly. "Avg Glucose Level" rose to 481.40, BMI to 406.14, and Age decreased slightly to 271.93.
These refinements suggest the model has become more reliant on metabolic indicators (glucose, BMI), which likely better align with medical insights.

Categorical Features (Gender, Private, Residence Type):

Pre-Tuning:
Gender (33.51), Private (40.59), and Residence Type (19.79) were moderately important.

Post-Tuning:
The importance of Gender increased slightly (54.35), indicating better capture of gender-related risk.
Private and Residence Type remained steady but dropped slightly in ranking, showing a shift towards focusing on medical indicators.

Hypertension and Smoking Behaviors:

Pre-Tuning:
Hypertension (23.51) and smoking behaviors (e.g., "Never Smoked" 22.53) contributed significantly.

Post-Tuning:
Hypertension (28.25) gained importance, reflecting its established role in stroke risk.
Smoking behaviors remained relevant but had slight variations in mean importance.

Features with Diminished Importance:

Children, Never Worked, 11–20 Age Range:
These features were negligible pre-tuning and remained the same post-tuning, with near-zero importance.

Impact of Refinement:
Post-tuning shifted focus toward features like Avg Glucose Level and BMI, which are more clinically significant for stroke prediction.
Lower-ranked features such as "Residence Type" and employment categories were deprioritized, aligning with domain knowledge.
Overall Improvements
Efficiency: By reducing reliance on low-impact features, the refined model likely achieves better generalization.

Analysis of evaluation metrics for the best model

	Stacking ensemble Model	Improved stacking ensemble Model
Accuracy	0.8375286041189931	0.8672768878718535
Precision	0.8049792531120332	0.8333333333333334
Recall	0.8899082568807339	0.9174311926605505
F1 Score	0.8453159041394336	0.8733624454148472
ROC AUC	0.9159649784257049	0.9199865946127099
MCC	0.6789036943338258	0.7383640555162516

• MCC (0.7383):
MCC is a balanced measure of classification performance that takes into account true positives, true negatives, false positives, and false negatives. It ranges from -1 (perfectly wrong) to +1 (perfectly correct), with 0 indicating no better than random guessing. The increase from 0.6789 to 0.7122 suggests that the improved model provides a more balanced and reliable performance, with fewer misclassifications overall.

• F1 Score (0.8734):
The F1 score is the harmonic mean of precision and recall, and it provides a balanced measure of the model's performance. The increase in the F1 score from 0.8453 to 0.8615 suggests that the improved model is not only better at detecting strokes but also at minimizing both false positives and false negatives. This is a good indicator of the overall effectiveness of the model, especially when dealing with imbalanced datasets like stroke prediction.

• Accuracy (0.8672):
The improved model shows a clear increase in accuracy, suggesting that the improvements made (e.g., through hyperparameter tuning or adjusting the model's

components) led to better overall performance. The model is correctly predicting more instances of stroke versus non-stroke.

• Confusion Matrix:
Improvement in True Positives and True Negatives:
After hyperparameter tuning, the model identified 6 additional stroke cases correctly (TP: 194 vs. 200). The number of correctly predicted no-stroke cases increased (TN: 172 vs. 179).

Reduction in Errors:
False negatives (missed stroke cases) dropped from 24 to 18, which is critical in a health-related model. False positives (wrongly identified stroke cases) reduced slightly from 47 to 40.

Overall Impact:
Hyperparameter tuning improved the model's sensitivity (ability to detect strokes) and slightly improved specificity (ability to detect no strokes). This means the tuned model is better at reducing life-critical errors, particularly in avoiding missed stroke diagnoses. In a health-related scenario like stroke detection, improving recall is crucial since false negatives (missed stroke cases) can have severe consequences. The improvement here demonstrates a meaningful advancement in the model's ability to minimize critical errors. This means the tuned model is better at identifying stroke cases, reducing the likelihood of missing true stroke patients.

• ROC AUC (0.92):
ROC AUC measures the ability of the model to distinguish between the classes (stroke vs. non-stroke) across all possible thresholds. The improved model shows a small but noticeable increase in ROC AUC, indicating that it has a slightly better overall ability to discriminate between the positive and negative classes. This is a strong indicator of the model’s robustness. The model is excellent at distinguishing between stroke and non-stroke cases.

• Precision (0.8333):
Precision measures the proportion of positive predictions (stroke) that are actually correct. The improvement from 0.805 to 0.8333 indicates that the refined model has a slightly better ability to avoid false positives (predicting stroke when there is none). This is important in medical applications, as fewer false positives mean fewer people are wrongly diagnosed as having a stroke.

• Recall (0.9174):
Recall measures the proportion of actual positive cases (true strokes) that are correctly identified by the model. The improved model's recall of 0.9174 shows a better ability to detect stroke cases compared to the stacking ensemble model (0.8899). This improvement is crucial because it means the improved model is identifying more stroke cases, which is vital for preventing undiagnosed strokes. The model correctly identifies 91.74% of the people who actually have a stroke, which is excellent for my goal of maximizing recall to catch all stroke cases.

Factors most likely to cause stroke from feature Importance

Below are the observations about the factors contributing to stroke prediction:

avg_glucose_level:
This feature has the highest importance, suggesting that blood glucose levels are a significant predictor of stroke risk. It likely indicates a strong correlation between elevated glucose levels and the occurrence of strokes.

BMI:
Body Mass Index is another highly important feature, showing that obesity or weight-related issues might influence stroke risk.

Age:
Age is also a critical factor, with older individuals being at higher risk for strokes. Demographic and Lifestyle Factors:

Gender:
Gender has some predictive value, though less important than physiological metrics like glucose and BMI.

Residence_type and Private (possibly representing employment type):
These socio-demographic features contribute moderately to predictions but are less critical compared to health-related features.

Hypertension and heart_disease:
These health-related conditions are well-known risk factors for strokes, and their inclusion in the model as moderately important is consistent with clinical understanding.

Smoking Behavior:
smokes, formerly_smoked, and never_smoked: Smoking habits contribute to stroke prediction, with active smokers likely being at higher risk compared to non-smokers or those who quit.

Age Ranges:

Specific age ranges (e.g., 61_70, 71_80, 51_60) contribute moderately to the model. This indicates that stroke risk may increase significantly in these age brackets, aligning with general medical knowledge.

Interpretation:
The results align with medical and clinical understanding of stroke risk:
Physiological indicators (e.g., glucose level, BMI, age) are the most critical factors. Health conditions (e.g., hypertension, heart disease) and lifestyle factors (e.g., smoking, marital status) provide additional context to assess risk. Demographic factors (e.g., gender, residence type) are less influential but still play a role in the overall prediction.

Outcomes from stroke prediction:

Health Management:
Focus on controlling blood glucose levels and maintaining a healthy BMI. Monitor and manage chronic conditions like hypertension and heart disease.

Lifestyle Changes:
Encourage smoking cessation and a healthier lifestyle to reduce stroke risks.
Demographic Analysis: Tailor public health initiatives to specific age groups and demographics based on the model's insights.

Feedforward Neural Network (shared model)

Visualizations and comparison of evaluation metrics

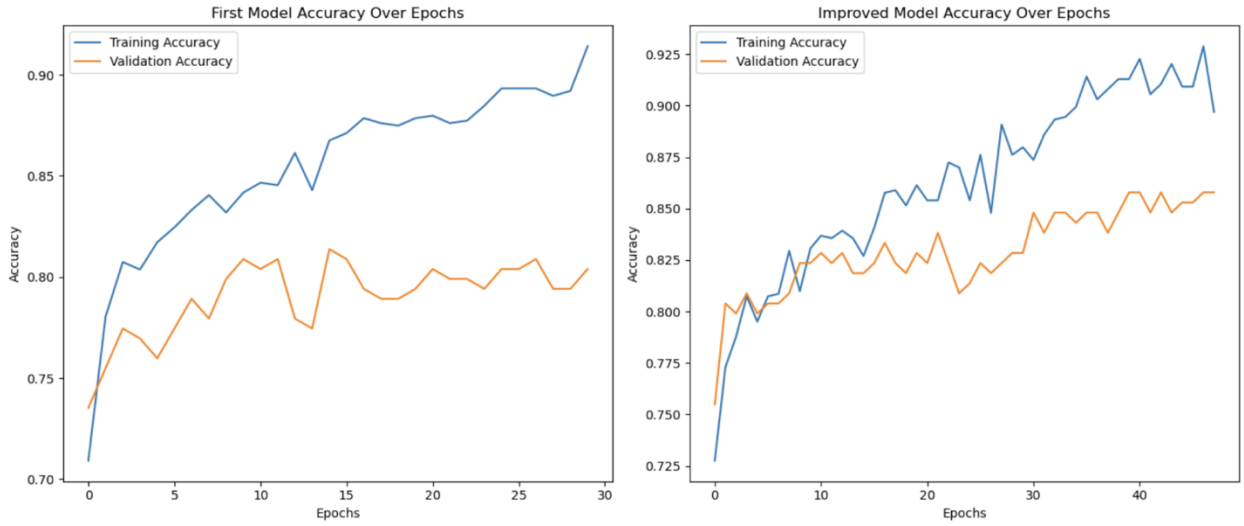
Evaluation Metrics:

	FNN Model	Improved FNN Model
Accuracy	0.8169336384439360	0.8421052631578947
Precision	0.7923728813559322	0.7944664031620553
Recall	0.8577981651376146	0.9220183486238532
F1 Score	0.8237885462555066	0.8535031847133758
ROC AUC	0.8732562523564158	0.8985379749486826
MCC	0.6360956617692703	0.6932700484153758

Accuracy over epochs Line plots

These plots provide a clear visual representation of how each model's accuracy changes over time during training. The side-by-side comparison of two models highlights differences in performance, enabling us to identify which model performs better over time.

The divergence between training and validation accuracy curves can indicate overfitting. If the training accuracy is high while validation accuracy stagnates or decreases, it suggests the model is overfitting to the training data.



Left First FNN Model before improving parameters

The training accuracy increases steadily, showing that the model is learning the training data well, but it reaches very high accuracy (above 0.90) early on, suggesting potential overfitting to the training data.

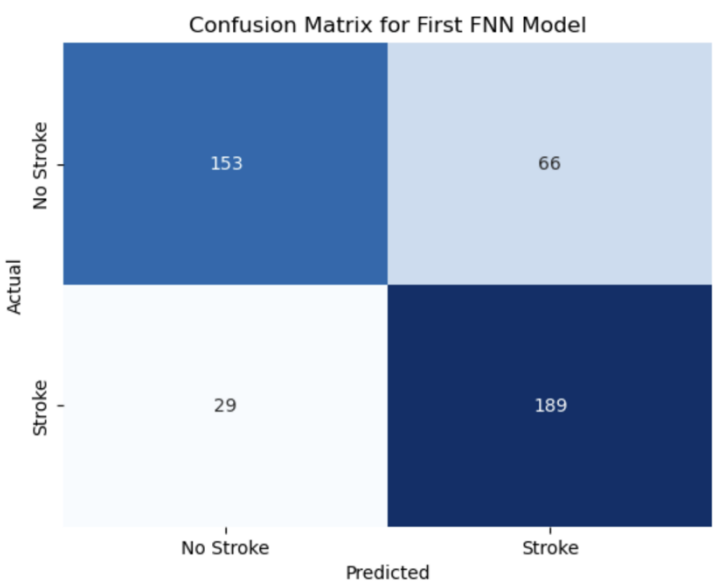
The validation accuracy remains stagnant and does not improve significantly after the initial epochs. The gap between training and validation accuracy widens as training progresses, which is a sign of overfitting.

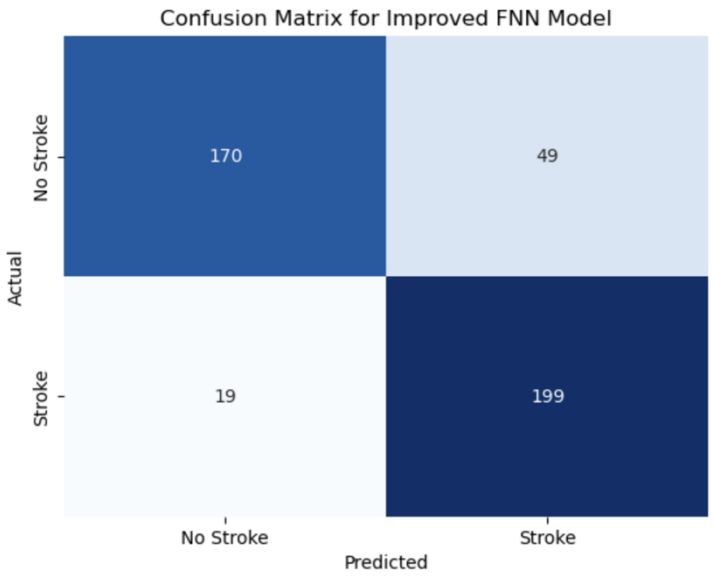
Right Improved FNN model

The training accuracy increases more gradually compared to the first model. It peaks at a slightly lower value than the first model, suggesting that regularization techniques have controlled overfitting. The validation accuracy is more stable and closely follows the training accuracy. There is less divergence between training and validation accuracy, indicating better generalization to unseen data.

In the improved model the smaller gap between training and validation accuracies suggests better generalization for stroke prediction because it generalizes well to new data.

Heatmap for Confusion matrix





Left Confusion matrix for First FNN Model

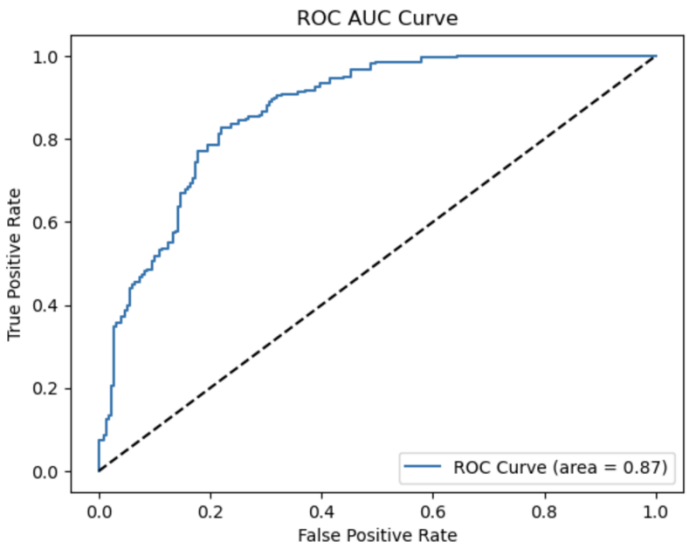
The false negative count (29) is relatively high, which is concerning in a medical context, as missing stroke cases can have severe consequences. The false positive count (66) indicates that some non-stroke cases were misclassified, but this is less critical than false negatives in this scenario. The model shows moderate accuracy but has room for improvement in sensitivity (recall for stroke cases).

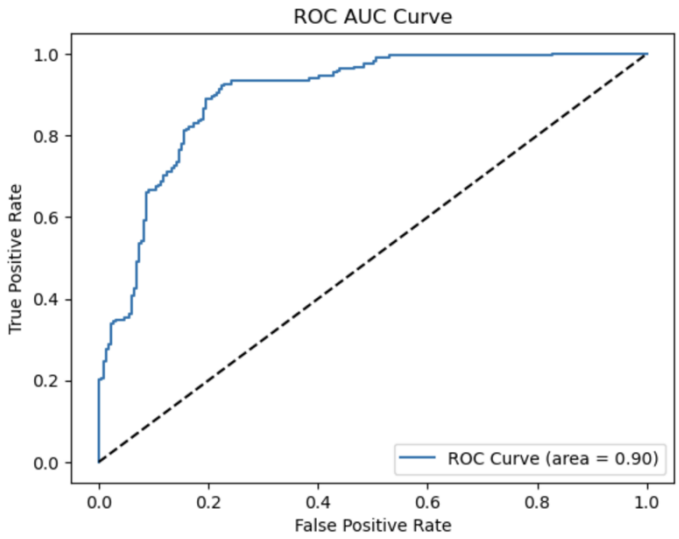
Right Confusion matrix for improved FNN model

The false negative count has been reduced significantly (from 29 to 19), indicating that the model is better at identifying stroke cases. This is critical in medical applications where recall (sensitivity) is prioritized. The slight decrease in false positives (from 66 to 49) is acceptable, as it prioritizes catching more stroke cases. The overall performance improvement reflects better sensitivity and slightly reduced specificity.

The improved stacking ensemble model demonstrated better performance, particularly in recall, which is critical for stroke prediction. While there is a minor trade-off in specificity, the overall improvements make the model more reliable for medical use cases.

ROC AUC Curve





The first graph has an AUC of 0.87.

The second graph has a higher AUC of 0.90.

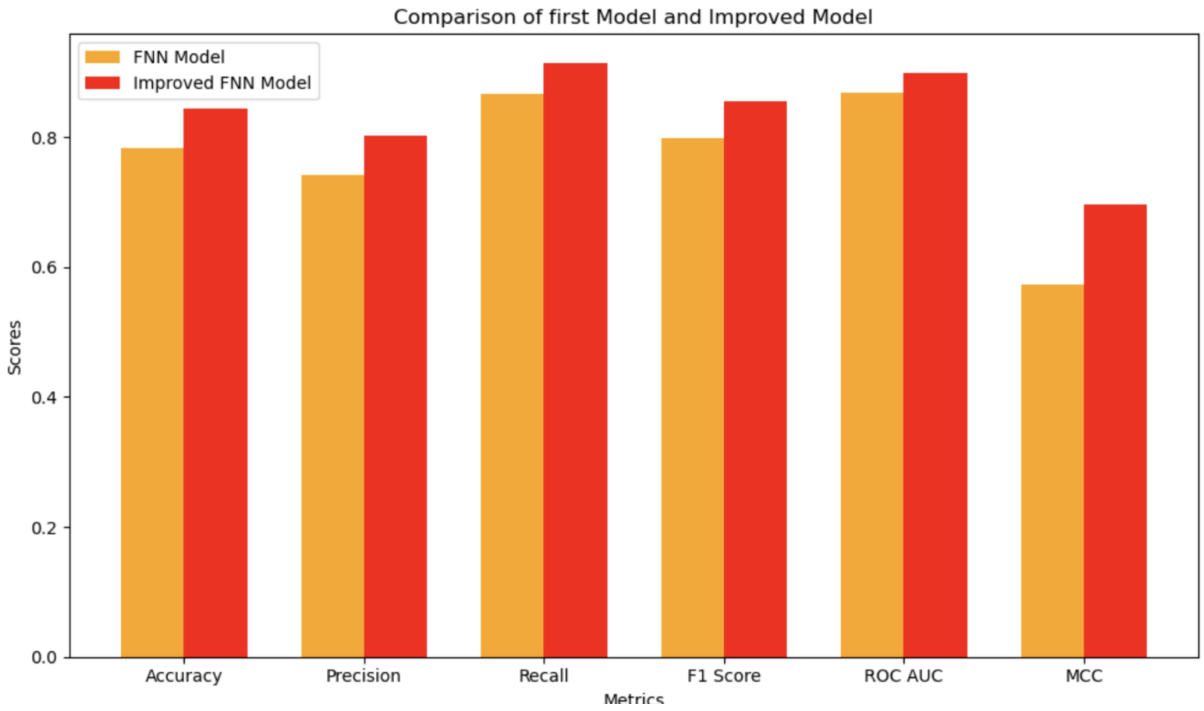
A higher AUC indicates better model performance in distinguishing between positive and negative classes. The second improved FNN model performs better overall. Both curves start at (0, 0) and end at (1, 1), as expected for ROC curves. The second curve (AUC = 0.90) is closer to the top-left corner than the first, indicating a higher true positive rate for the same false positive rate.

The improved FNN model suggests a moderate improvement in predictive capability, especially if applied in contexts with balanced classes.

The second improved model (AUC = 0.90) consistently achieves higher sensitivity for similar false positive rates than the first model (AUC = 0.87), making it preferable for use cases requiring higher true positive rates (e.g., medical diagnostics).

Second Model (AUC = 0.90): Better suited for scenarios requiring high sensitivity, such as: Disease screening: Missing a positive case has severe consequences.

Bar chart for metrics comparison



Analysis after evaluation of FNN models:

The Improved FNN Model has a much higher recall (91%) compared to the original model (86%), meaning it's better at identifying people who are at risk of having a stroke. So, the Improved FNN Model is the better choice for my objective to catch most people who actually have a stroke.

- The Improved FNN Model shows a significant improvement across all key metrics: accuracy, precision, recall, F1 score, ROC AUC, and MCC.

- The improvements are likely due to the regularization techniques, batch normalization, more extensive hyperparameter tuning, and training enhancements like early stopping, learning rate scheduling, and model checkpointing in the second model.

Model Performance for Stroke Prediction

The improvements in the model's performance, as seen in the evaluation metrics, have important implications for stroke prediction.
Higher Accuracy (84%) • The improved model correctly predicts the presence or absence of a stroke in 84% of cases, which is a significant improvement from the initial 78%. • Impact on stroke prediction: A higher accuracy means the model is better at identifying both stroke and non-stroke cases. This is important in a medical context because it reduces the likelihood of misdiagnosing a patient.

Increased Precision (80%) • Precision refers to the percentage of true positive stroke predictions out of all the cases predicted as positive (stroke). The improved model's precision has increased by a tiny bit, meaning fewer false positives. • Impact on stroke prediction: Higher precision means that the model is less likely to incorrectly classify healthy individuals as having a stroke. In medical practice, this is crucial because misdiagnosing a healthy person could lead to unnecessary treatments, tests, or anxiety for the patient. Both models are almost the same here, with the improved model being just a tiny bit better. This means the positive predictions are accurate in both models.

Improved Recall (91%) • Recall measures the percentage of actual stroke cases that the model correctly identifies. The improved model has a recall rate of 91%, meaning it correctly identifies 91% of all stroke cases. • Impact on stroke prediction: A higher recall is critical for stroke prediction because it means the model is much better at catching the majority of actual stroke cases. In a healthcare setting, this reduces the risk of missing a stroke diagnosis, which could lead to delayed treatment or worse patient outcomes. Early detection of strokes is vital for timely intervention and better recovery chances.

Better F1 Score (85%) • The F1 score is the harmonic mean of precision and recall. The improved model has a higher F1 score, reflecting a better balance between precision and recall. • Impact on stroke prediction: A higher F1 score suggests that the model performs well in both identifying stroke cases and avoiding false positives. For stroke prediction, this means the model is both sensitive (detecting most strokes) and specific (avoiding unnecessary false alarms).

Improved ROC AUC (90%) The improved model is slightly better at distinguishing between strokes and non-strokes. • The improved model has a higher ROC AUC, indicating that it has a better ability to discriminate between stroke and non-stroke patients. • Impact on stroke prediction: A higher ROC AUC means the model is better at distinguishing between stroke and non-stroke cases across all thresholds. This is important for ensuring that the model can be used effectively in different clinical scenarios, where the trade-off between sensitivity and specificity might need to be adjusted.

Improved Matthews Correlation Coefficient (MCC) (70%) • The MCC is a balanced measure that takes into account true positives, true negatives, false positives, and false negatives. The improved model has a higher MCC meaning it's a bit better overall at handling both positive and negative cases also indicating a better overall performance, especially when dealing with imbalanced data (e.g., more non-stroke cases than stroke cases). • Impact on stroke prediction: A higher MCC means that the model is more reliable in its predictions, with fewer misclassifications overall. In stroke prediction, this is particularly valuable in ensuring that the model is not biased toward predicting one class (e.g., non-stroke) over the other, which could lead to missing critical stroke cases.
Overall Impact on Stroke Prediction:

- Better detection: The improved model is more likely to detect actual strokes (higher recall), which is crucial for early intervention.

- Fewer false alarms: With reasonably good precision, the model is less likely to falsely diagnose someone as having a stroke, reducing unnecessary treatments or tests.

- More balanced predictions: The higher F1 score and MCC indicate that the model is making more balanced and reliable predictions, crucial for a real-world clinical setting where both false positives and false negatives can have serious consequences.

- Improved clinical decision-making: With these improvements, healthcare professionals can trust the model more, using it as a valuable tool for stroke risk assessment and early detection, ultimately leading to better patient outcomes.

Recommendations

We could target values closer to 0.8 MCC and 90% F1 score, but 85% F1 is already excellent in most practical applications. We can try other models to compare with FNN for stroke prediction specially comparing Recall value if catching majority of strokes is a priority.

Comparison of metrics for different Models

Metric	Improved Stacking Ensemble Model	Improved FNN Model
Accuracy	0.8673	0.8421
Precision	0.8333	0.7945
Recall	0.9174	0.9220
F1 Score	0.8734	0.8535
ROC AUC	0.9200	0.8985
MCC	0.7384	0.6933

The above highlights that the Improved Stacking Ensemble Model generally outperforms the Improved FNN Model across most metrics, with higher Accuracy, Precision, F1 Score, ROC AUC, and MCC. The Improved FNN Model, however, slightly edges out in Recall.

Hyperparameters for Improved Stacking Ensemble Model

Parameter	Value
Test Size	0.3
Cross-Validation Folds	5
Randomized Search Iterations	15
Refined Best Score	0.8538
LightGBM Parameters	
Learning Rate	0.0598
Max Depth	6
Number of Estimators	150
CatBoost Parameters	
Learning Rate	0.1012
Depth	4
Iterations	50
HistGradientBoosting Parameters	
Learning Rate	0.2378
Max Iterations	150
Meta-Model (CatBoost) Parameters	
Learning Rate	0.1153
Depth	3
Iterations	100

Hyperparameters for Improved FNN Model

Parameter	Value
Test Size	0.3
Validation Split	0.2
Epochs	50
Patience	5
Batch Size	34
Learning Rate	0.005
Optimizer	adamax

Layer Details for Improved FNN Model

Layer	Neurons	Activation	Dropout Rate	Regularizer	Regularizer Value
Layer 1	384	tanh	0.3	L2	0.001
Layer 2	384	tanh	0.3	L2	0.001
Layer 3	32	tanh	0.3	L2	0.001
Output Layer	1	sigmoid	N/A	None	N/A

4.3 Ximena Michelle Castro Villalobos [6515820]

To enable comparison between the different models and versions of FNN the same performance metrics were used by the four members of the group: Accuracy, precision, recall, F1-score, MCC and ROC and AUC.

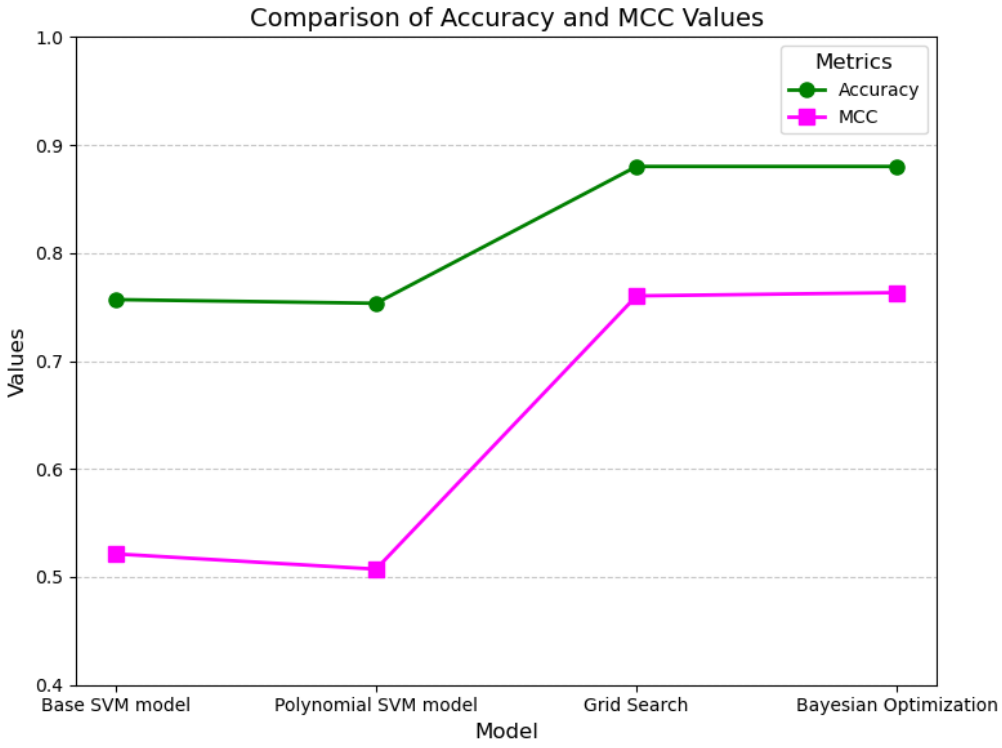
Support Vector Machine (unique model)
For the SVM model, I first computed a base version and then added different optimization techniques to compare and choose the one that would give me the best performance results, which were obtained by looking at the different performance metrics values.

Optimization technique	Performance metrics						
	Accuracy	Precision	Recall	F1-score	Support	MCC	AUC
Base model	0.7568493150684932	- Class 0: 0.81 - Class 1: 0.72 - Macro avg: 0.76	- Class 0: 0.67 - Class 1: 0.84 - Macro avg: 0.76	- Class 0: 0.73 - Class 1: 0.78 - Macro avg: 0.76	- Class 0: 146 - Class 1: 146 - Macro avg: 292	0.5213993951603676	0.763698630136 9862
Polynomial kernel	0.7534246575342466	- Class 0: 0.76 - Class 1: 0.74 - Macro avg: 0.75	- Class 0: 0.73 - Class 1: 0.77 - Macro avg: 0.75	- Class 0: 0.75 - Class 1: 0.76 - Macro avg: 0.75		0.5072778598240506	NA
Grid Search for hyperparameters	0.8801369863013698	- Class 0: 0.88 - Class 1: 0.88 - Macro avg: 0.88	- Class 0: 0.88 - Class 1: 0.88 - Macro avg: 0.88	- Class 0: 0.88 - Class 1: 0.88 - Macro avg: 0.88		0.7602918066411091	NA
Bayesian optimization	0.8801369863013698	- Class 0: 0.92 - Class 1: 0.85 - Macro avg: 0.88	- Class 0: 084 - Class 1: 0.92 - Macro avg: 0.88	- Class 0: 0.87 - Class 1: 0.89 - Macro avg: 0.88		0.7633058593186576	0.880136986301 3698

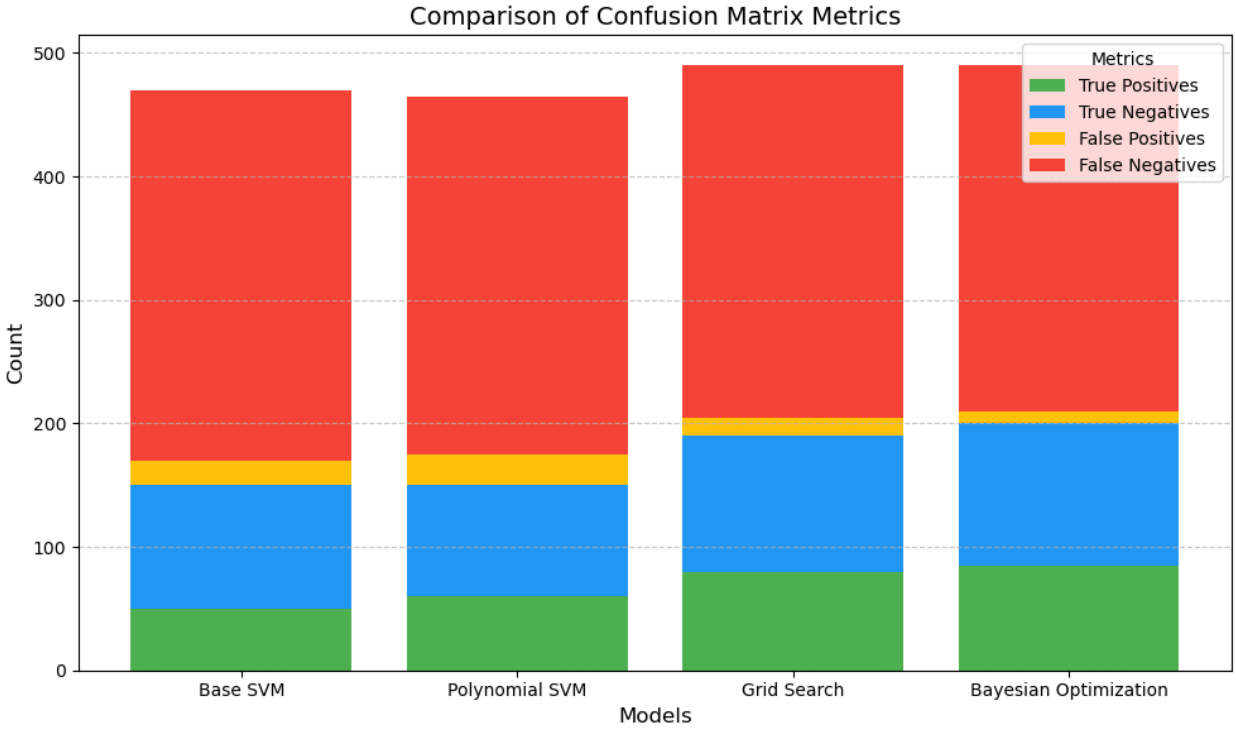
Best hyperparameters obtained with Bayesian optimization on trial 25:
f1 score value: 0.899372117615524.
hyperparameters:
'C': 37.31016016959667
‘gamma': 0.05076235484219706
kernel='rbf'

Best performance metrics for SVM: Bayesian optimization

- Accuracy (0.88) and MCC (0.763):
  These high values for accuracy and MCC can show a strong performance of the model with this optimization algorithm and an efficient ability to generalize the new data.
  The model correctly classified the classes 88% of the samples in the dataset. Accuracy can be a misleading metric for unbalanced datasets, unlike this one which was previously balanced with under-sampling and SMOTE.
  An MCC of 0.763 shows a strong correlation between predicted and actual values which indicates an accurate prediction for both classes.
  Both of these measures show that the model has an effective handling of the balanced data, avoiding bias, and maintaining consistency in predictions across classes.



The macro average value was used to compare the different optimization algorithms as it works better for balanced datasets and on the performance of the model for each class which is particularly useful for medical cases.



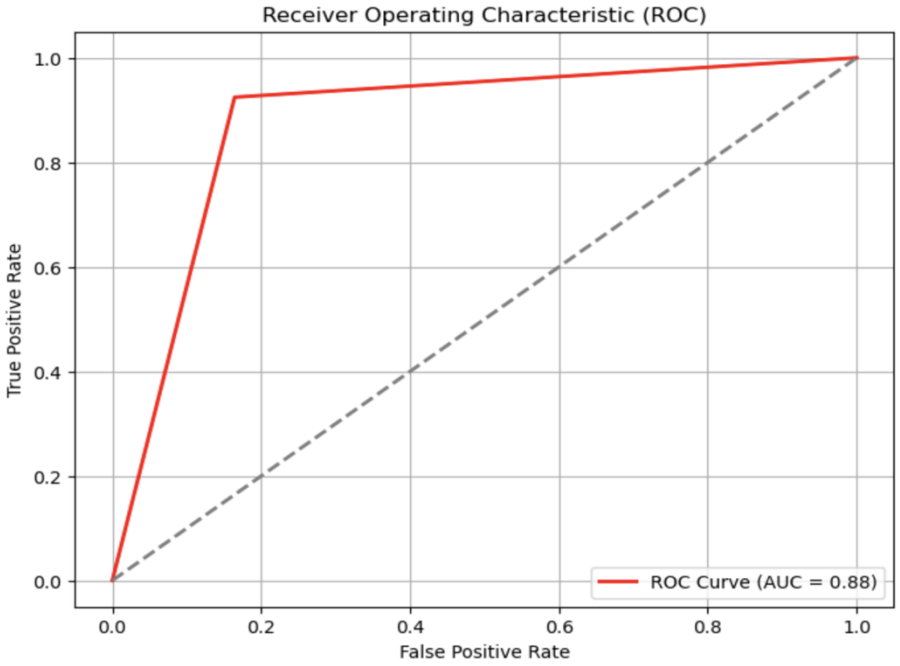
Besides accuracy, other metrics that are part from the confusion matrix are:

- Precision, Recall and F1-score:

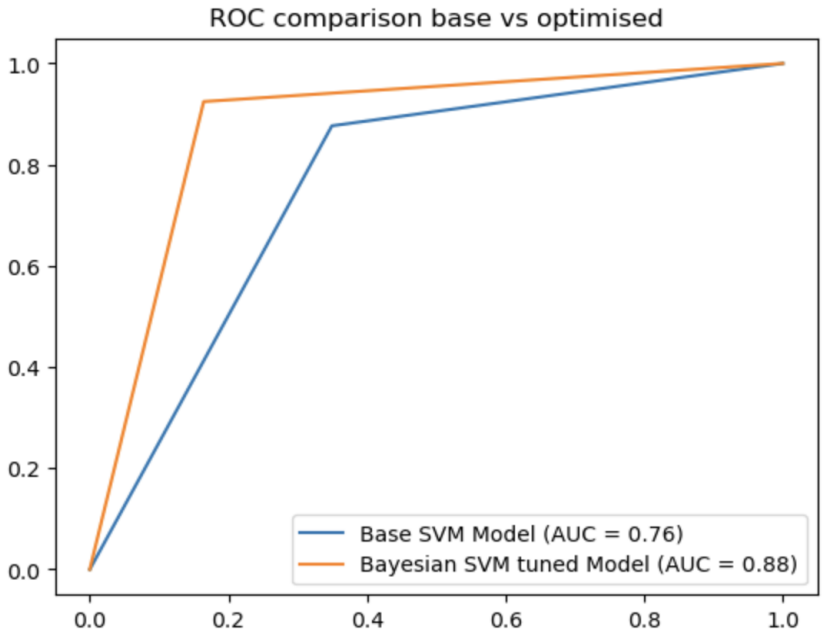
  The three of them have a macro avg of 0.88, which shows a consistent and effective performance of the model across both classes. It also indicates that the model handles the complexities of the data well, including any potential synthetic noise introduced by SMOTE.

  Also, a macro F1-score of 0.88 reflects that the model balances precision and recall well across both classes, making it reliable for both detecting positive cases and avoiding false alarms. Which is essential in medical applications such as stroke prediction

- ROC and AUC




```
AUC: 0.8801369863013698
```



The AUC of the Bayesian SVM tuned model 0.88 shows a good model performance as it indicates a strong ability to differentiate between classes. It also shows an improvement on the efficiency of the model as the base SVM only shows an AUC of 0.76.

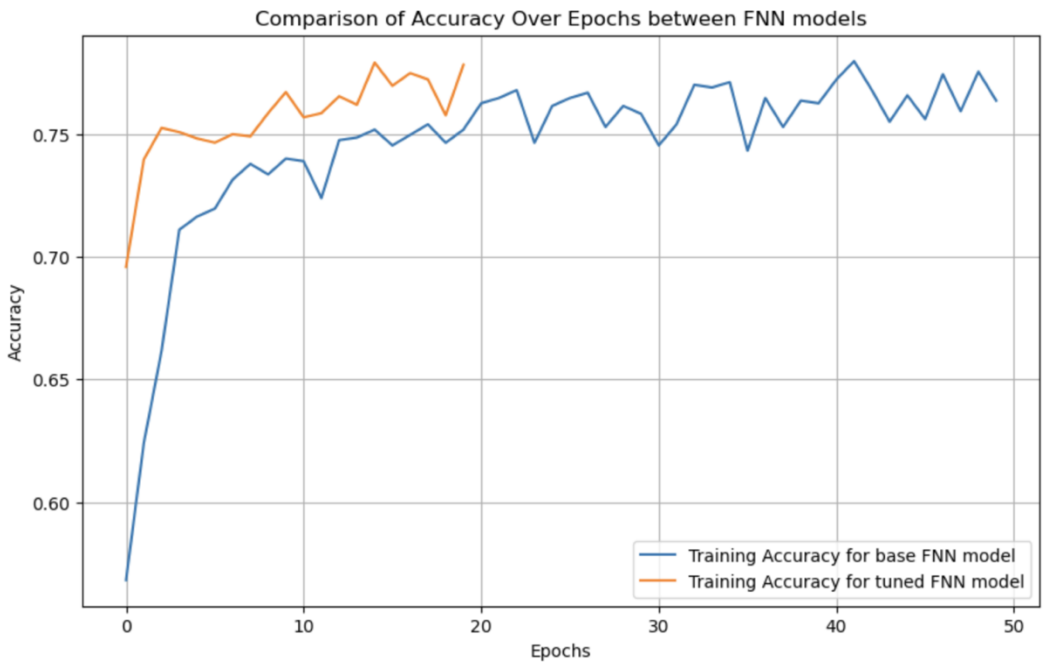
Feedforward Neural Network (shared model)

Best Parameters: {'activation': 'tanh', 'batch_size': 16, 'dropout_rate': 0.0, 'epochs': 20, 'neurons': 64, 'optimizer': 'adam'}

	Base FNN	Tuned FNN
Accuracy	0.7739726305007935	0.7945205479452054
Precision	• For class 0: 0.84 • For class 1: 0.73 • macro_avg: 0.78	• For class 0: 0.84 • For class 1: 0.76 • macro_avg: 0.80
Recall	• For class 0: 0.68 • For class 1: 0.87 • macro_avg: 0.77	• For class 0: 0.73 • For class 1: 0.86 • macro_avg: 0.79
F1 score	• For class 0: 0.75 • For class 1: 0.79 • macro_avg: 0.77	• For class 0: 0.78 • For class 1: 0.81 • macro_avg: 0.79
MCC	0.5583086534376924	0.5935694526523001
AUC	0.7739726027397261	0.7945205479452055
Test loss	0.4860832393169403	0.4860832393169403

- Accuracy:

  An accuracy of 0.7945 indicates the model is correctly predicting the outcome 79.45% of the time. As this dataset involved multiple classes and complex relationships, an accuracy of 79.45% indicates better-than-random performance and also an improvement from the base model. Accuracy is not the best evaluation metric in terms of unbalanced data as the model can only predict correctly the majority class. However, as this dataset has already been previously balanced this was not a problem.



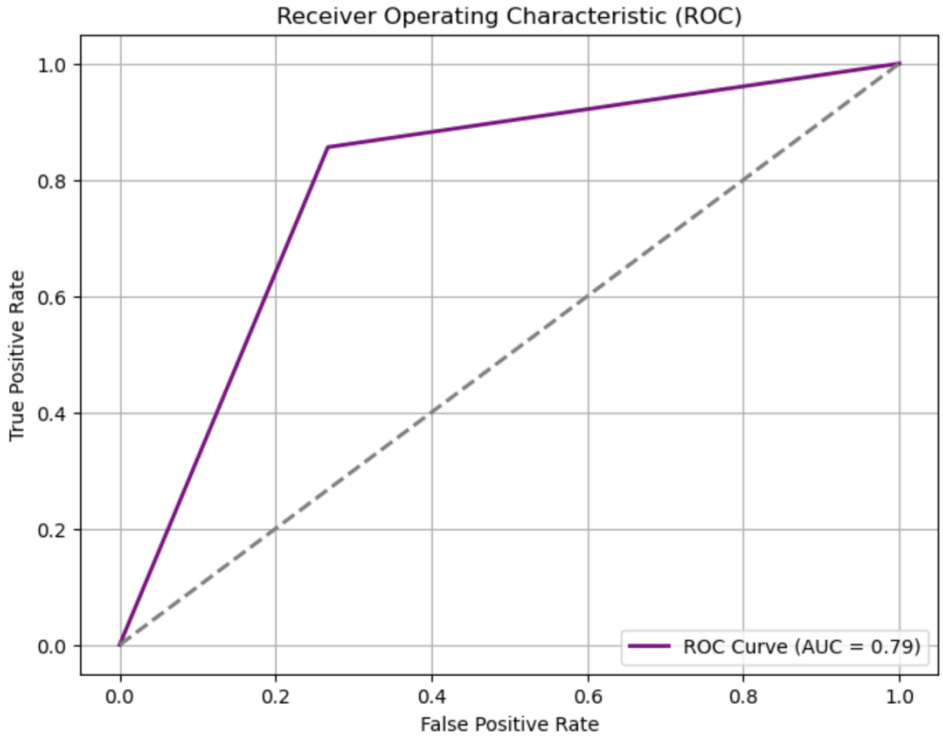
- Precision, recall and F1-score:
  A macro average value of precision of 0.80 shows robustness across both classes, this precision score shows that the model's predictions are generally accurate. When looking at the precision per class the model struggles more to predict correctly class 0 compared to class 1. This can be due to features of class 1 overlapping significantly with other classes, making it harder for the model to distinguish.
  In the case of recall, on average, the model correctly identifies 79**%** of the actual instances across both classes. Furthermore, only 14% are false negatives for class 1. The model does a better job of capturing class 1 compared to class 0, which is likely due to better learned patterns or a higher focus on this class
  For F1-score, a macro average of 0.79 indicates a good performance of the model across both classes. The minimal difference on scores between classes suggests the model is marginally better at handling class 1 (0.81). Furthermore, it shows the model being relatively balanced in its performance between the two classes.

- MCC: 0. 5935694526523001

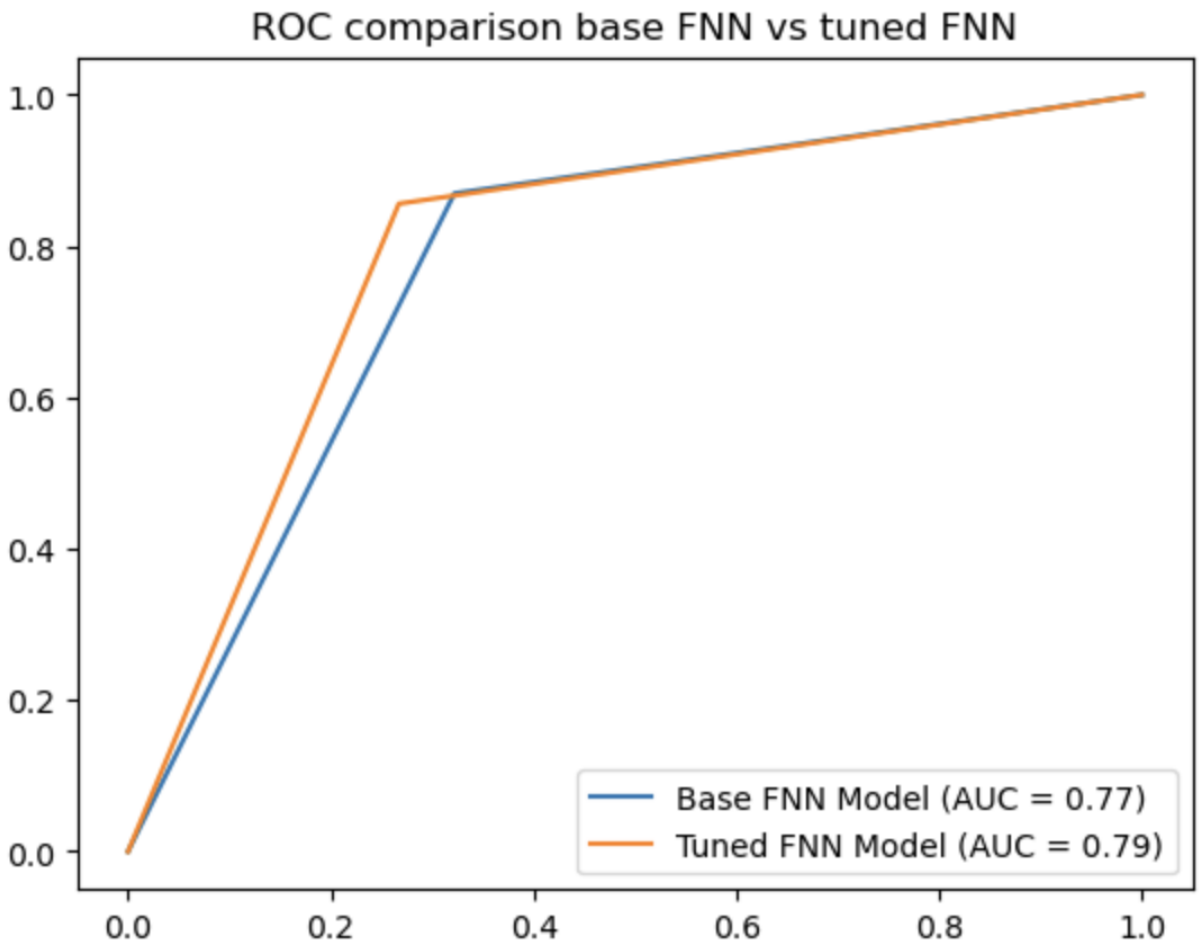
  For binary tasks, such as stroke target variable, MCC provided a more reliable assessment than accuracy or F1-score. MCC = 0.5935 indicates that the model's predictions are significantly better than random guessing and have a strong positive correlation with the true labels.

- ROC and AUC: 0.7945205479452055

  The ROC curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) for various threshold values.
  The high AUC score (0.7945) of the tuned FNN model is consistent with the macro precision (0.80), macro recall (0.79), and F1-scores for the individual classes. This indicates that the model is balanced in handling both precision and recall. It also shows that this FNN model would be reliable in real-world applications where distinguishing between two classes is critical, such as in a healthcare case for stroke prediction.



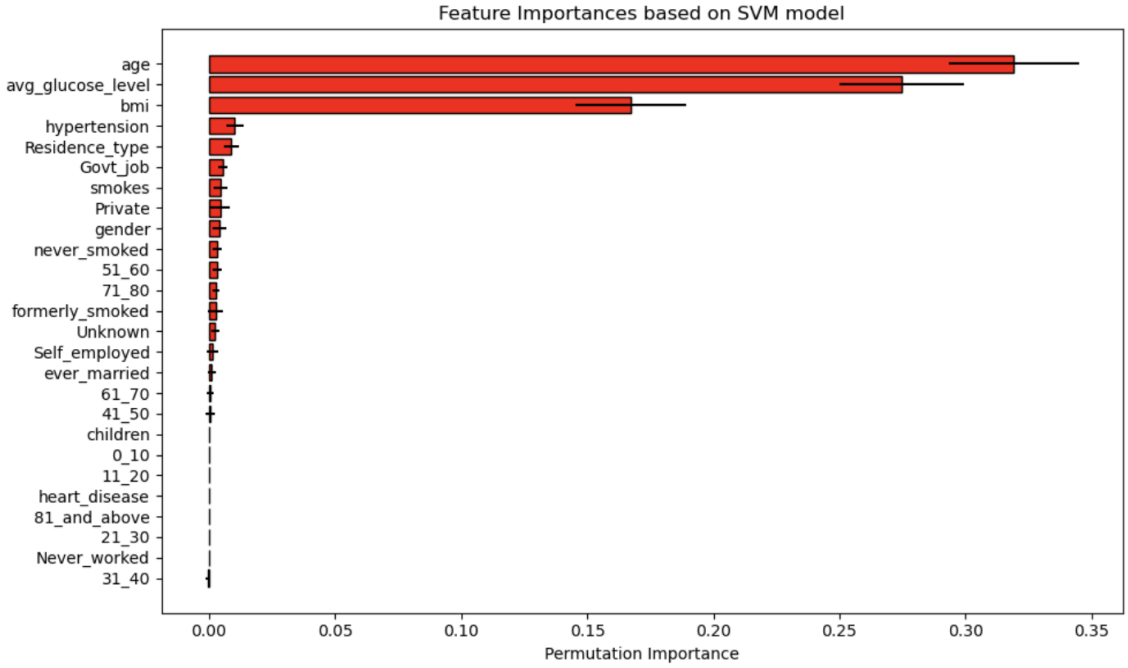
AUC: 0.7945205479452055



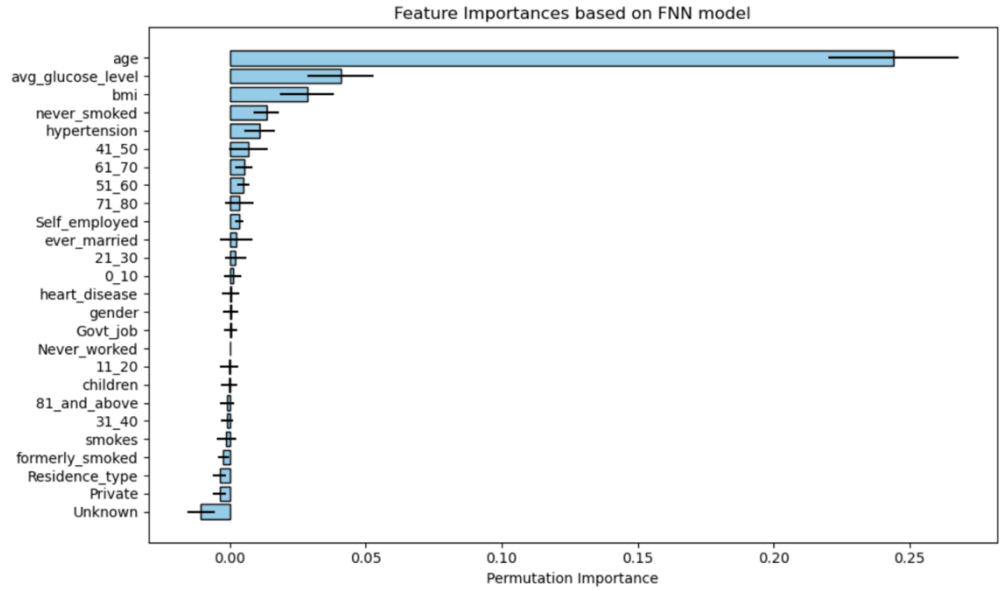
The tuned model is approximately **11.64 percentage points** better at distinguishing between the classes compared to the base FNN model. This is a significant improvement, especially for medical applications such as stroke prediction, since it would mean a reduction in false positives/negatives and therefore improve the decision-making process. Furthermore, this improvement shows that the base model might have been underfitting in comparison with the tuned one and demonstrates the impact of the hyperparameters changes. For example, the increase in neurons (64) enabled the model to capture more complex relationships in the data.

Feature importance

In the case of SVM with Bayesian optimization, the three most important features after age are avg_glucose_level, bmi and hypertension



For the tuned FNN model, the three most important features after age are avg_glucose_level, bmi and never_smoked.



For both cases “age” was not considered as it corresponds to the feature before hot encoding. Although both methods have similar feature importances they are not the same. This difference arises as features that contribute the most to defining the decision boundary between classes are more important for the SVM model. Whereas, in the case of FNN the feature importance is assigned focusing on how much they reduce the loss function across all layers, which is also impacted by the network’s architecture, activation function and weights. The complexity of each model can also be a reason for this difference on the feature importance as neural networks can capture interactions and dependencies between features that may not be apparent in simpler models like SVM.

4.4 C Ashwini Reddy [6886963]

eXtreme Gradient Boost Model (unique model)

```
XGBoost Model
Accuracy: 0.88
ROC AUC Score (Class 1 - Stroke): 0.93

Confusion Matrix:
[[117, 29]
 [7, 139]]

Metrics for Class 0 (No Stroke):
Precision: 0.94
Recall: 0.80
F1 Score: 0.87

Metrics for Class 1 (Stroke):
Precision: 0.83
Recall: 0.95
F1 Score: 0.89

MCC for Class 1 (Stroke): 0.76
```

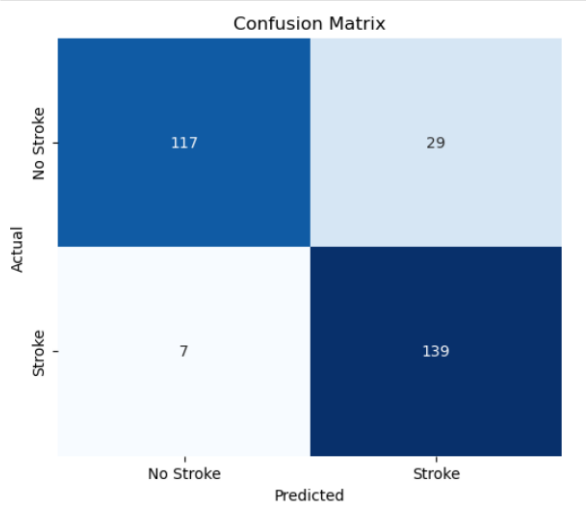
Strengths of the XGBoost Model for Medical Datasets (Stroke Prediction)

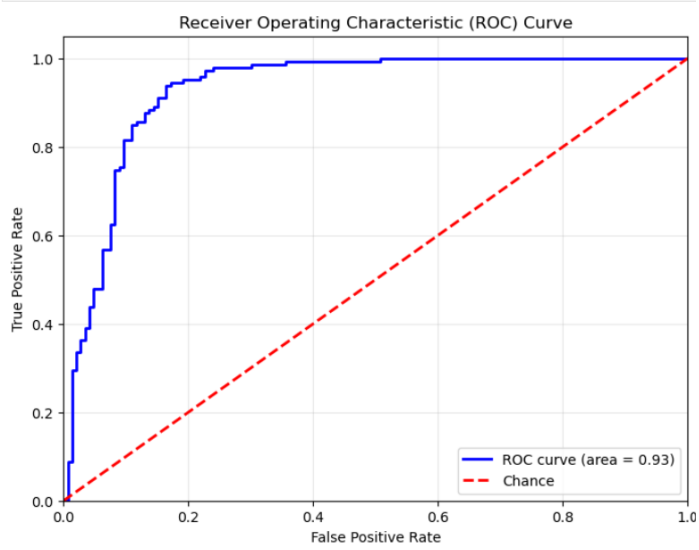
- High Accuracy and ROC AUC Score – Accuracy: 0.88 and ROC AUC: 0.93 are strong indicators that the model is performing well and is good at distinguishing between stroke and non-stroke cases.
- Balanced Precision and Recall for Class 1 (Stroke) – Precision for Class 1 (Stroke): 0.83 and Recall: 0.95 indicate that the model is both accurate in detecting strokes and good at identifying true stroke cases. High recall for Class 1 indicates that most of the actual stroke cases are being identified.

Weaknesses of the XGBoost Model for Medical Datasets (Stroke Prediction)

- False Positives for Class 0 (No Stroke) – The false positive rate is higher for Class 0 While precision for non-stroke is high (0.94), the model could be overpredicting non-stroke cases as strokes.
- Overfitting Risk – XGBoost is highly flexible but there's a risk of overfitting to noise in the medical data. Regularization parameters and careful validation are essential to avoid this.

Graph for ROC AUC and Confusion matrices:






```
Confusion Matrix:
True Negative (Class 0 - No Stroke): 117
False Positive (Class 0 - No Stroke): 29
False Negative (Class 1 - Stroke): 7
True Positive (Class 1 - Stroke): 139
```

Confusion Matrix:

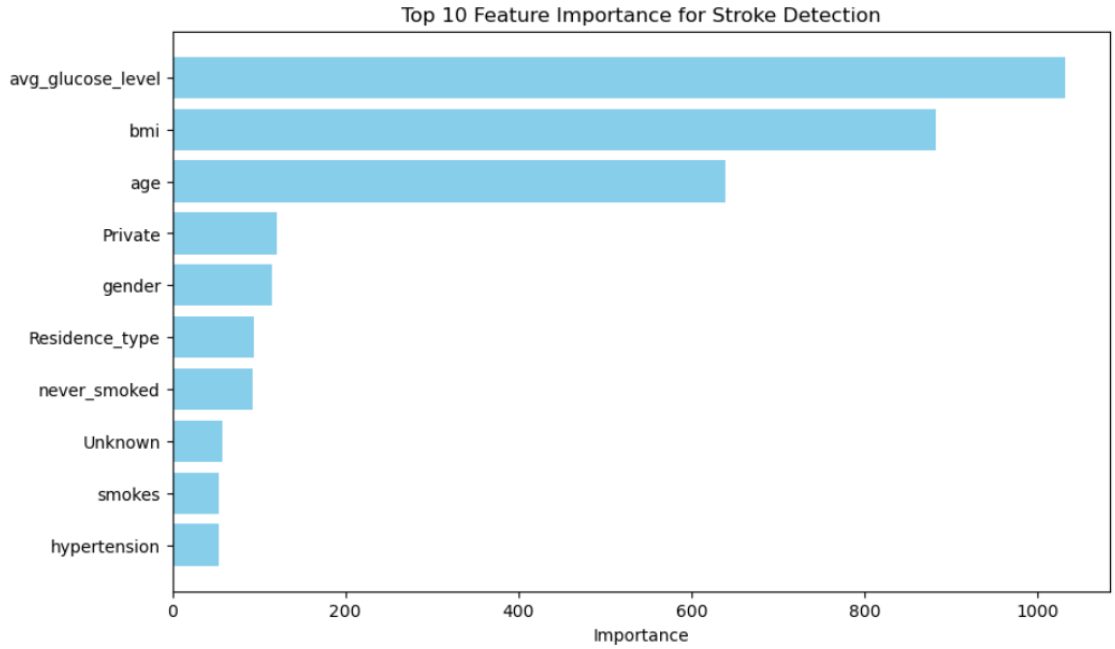
- The model has performed well in correctly identifying both positive and negative cases, but there is still a concern about false positives and false negatives.
- False Positives (29) can lead to unnecessary treatments, which may be costly or invasive.
- False Negatives (7) are even more concerning because patients who have the disease are missed, potentially delaying treatment.

ROC AUC Curve:

- AUC of 0.93 is a strong indicator that our model performs well, and in the context of medical data, this would suggest that the model is a very useful tool for

diagnosis, though care should still be taken to balance the trade-offs between false positives and false negatives.

Graph Obtained for Feature selection:



The top 10 feature importance bar chart visually displays which features are most critical for detecting strokes, based on the frequency with which they are used to split the data during model training.

Feedforward Neural Network (shared model)

```
Accuracy: 0.81

Metrics for Class 0 (No Stroke):
Precision: 0.92
Recall: 0.68
F1 Score: 0.78

Metrics for Class 1 (Stroke):
Precision: 0.74
Recall: 0.93
F1 Score: 0.82

MCC for Class 1 (Stroke): 0.64
AUC-ROC for Class 1 (Stroke): 0.88

Confusion Matrix:
True Negative (Class 0): 153
False Positive (Class 0): 71
False Negative (Class 1): 14
True Positive (Class 1): 199
```

Strengths of the FNN Model for Medical Datasets (Stroke Prediction):

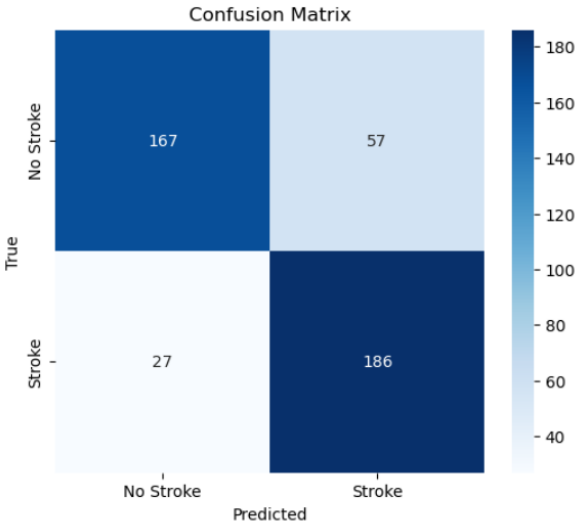
- Good Recall for Stroke (Class 1): 93% recall means the model is very good at identifying stroke cases.
- Reasonable Accuracy (81%): Overall, the model performs decently.

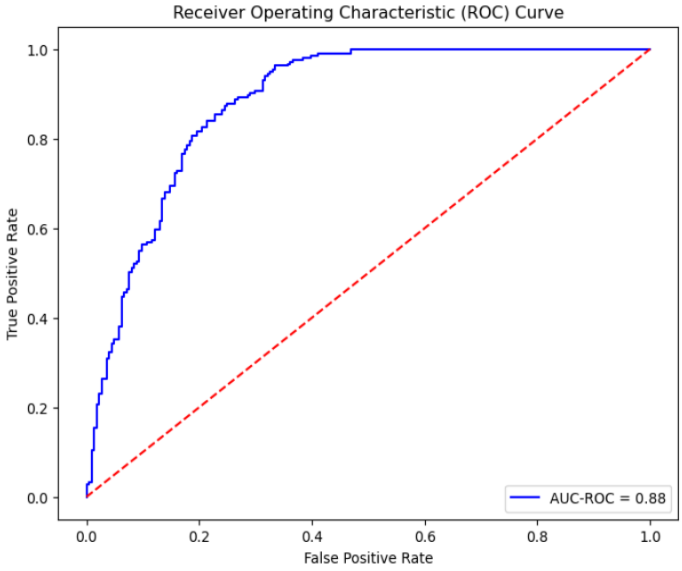
Weaknesses of the the FNN Model for Medical Datasets (Stroke Prediction):

- Low Precision for Stroke (Class 1): Precision of 74% means the model falsely predicts stroke cases quite often.
- Low Recall for No Stroke (Class 0): 68% recall indicates many "No Stroke" cases are missed.

Graph for ROC AUC and Confusion matrices

```
Confusion Matrix:
[[167  57]
 [ 27 186]]
```





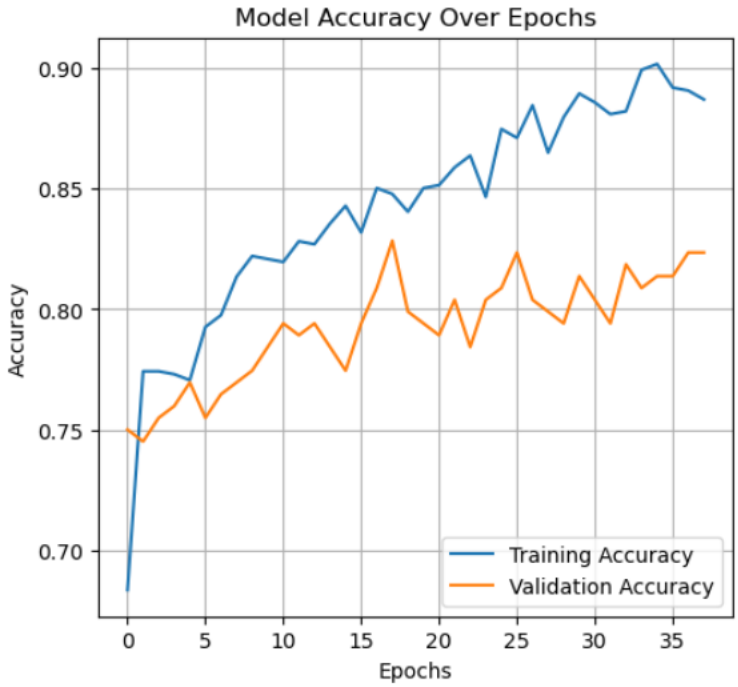
Confusion Matrix

- The model performs well in identifying both true positives (186 cases) and true negatives (167 cases).
- However, it has 57 false positives and 27 false negatives.
- The false negatives are particularly critical in a medical context, as they mean the model is missing some patients who have the disease. Depending on the disease, this could have serious consequences.

ROC AUC Curve

- An AUC score of 0.88 shows that our model is doing a very good job in distinguishing between the positive and negative classes, with a high probability of ranking positive examples above negative ones.

Accuracy Vs Epochs



We can observe that the model starts to overfit after a certain number of epochs, causing the training accuracy to grow faster than the validation accuracy. The slight convergence at the end is due to early stopping, regularization, or simply the model starting to generalize better again after some overfitting.

5. Discussion, Conclusion, and Recommendations

5.1 Discussions and Comparisons

Metrics used:

- Accuracy: Basic metric and useful in case of balanced data but is not enough to tell if a model is working well so we have used other models.
- Precision: As we are doing medical diagnoses the cost of false positive is high and it's important to use precision, but it ignores false negative.
- Recall: Failing to detect a stroke can have severe consequences so, recall ensures that as many positive cases are identified as possible.
- F1-Score: It's a single matric that takes both recall and precision into account.
- ROC AUC curve: It helps to discriminate between the positive and negative classes, regardless of the decision threshold.
- MCC (Matthews Correlation Coefficient): It is a balanced measure considering all the four values - True Positives, False Positives, True Negatives, False Negatives.

FNN Model comparison with other teammates

Models	Accuracy	Precision	Recall	F1 score	MCC	Confusion Matrix	ROC AUC
Ashwini	0.81	0.74	0.93	0.82	0.64	[[153 71] [14 199]]	0.88
Fizza	0.89	0.86	0.93	0.89	0.79	[[124 22 [9 137]]	0.94
Sampada	0.78	0.75	0.85	0.80	0.57	[[158 61 33 185]]	0.78
Ximena	0.79	0.80	0.79	0.79	0.59	[[107 39 [21 125]]	0.79

Best combination after hyperparameter tuning for different FNN models

Models	Hidden Layers	No of Neurons (per layer)	Activation	Dropout Rate	Test Size	Validation Split	Epochs	Batch size	Learning rate	optimizer
Ashwini	2	64,128	ReLU, sigmoid	0.2	0.3	0.2	40	34	0.001	Adam
Fizza	3	384	Tanh, sigmoid	0.3	0.3	0.2	50	34	0.005	adamax
Sampada	2	32	ReLU	0.2, 0.4	0.3	-	50	32	0.00097	Adam
Ximena	2	64	Tanh	0.0	0.2	-	20	16	0.001	Adam

From the above two tables we can come to conclusion on which parameter contributes to increasing the effectiveness and usefulness of the model in our dataset.

- Hidden layers: increasing the hidden layer to 3 improves accuracy but also risks overfitting of the data, to balance this we should normalize dropout and batch size.
- Number of neurons: To balance the complexity of a model we can use intermediate values like 128-256.
- Activation: ReLU was commonly used but Tanh seems to work well.
- Dropout rate: Higher rate improves generalization but avoid higher than 0.4 to retain sufficient learning capacity.
- Learning rate: Higher learning rate shows better performance.
- Batch size: Smaller batches work well but also slow the training.

Comparison between the shared FNN models

Fizza's model outperforms all other FNN models in terms accuracy (0.89), F1 Score (0.89), MCC (0.79), and ROC AUC (0.94), indicating great predictive power balanced classification. Ashwini's model has a high recall (0.93) but have lower accuracy of 0.81. Ximena's and Sampada's models also have relatively lower accuracies of 0.79 and 0.78 respectively as well as ROC AUC value of 0.79 and 0.78 respectively. Sampada's model outperforms slightly in recall (0.85) compared to Ximena (0.79).

In terms of hyperparameter tuning, Fizza's model uses three hidden layers with 384 neurons, Tanh and sigmoid activation functions, a dropout of 0.3, and an Adamax optimizer. Ashwini's model uses two hidden layers with 64 and 128 neurons, ReLU and sigmoid activations, and a dropout rate of 0.2, delivering strong recall. Sampada's model uses two hidden layers with 32 neurons for each layer, ReLU activation, a variable dropout rate (0.2, 0.4) and Adam optimizer. Ximena's model includes two hidden layers with 64 neurons, Tanh activation, no dropout rate, and an Adam optimizer

Unique models comparison

Model	Accuracy	Precision		F1 Score		MCC	Recall		Confusion matrix	ROC AUC
		0	1	0	1		0	1		
XGBoost (Ashwini)	0.88	0.94	0.83	0.87	0.89	0.76	0.80	0.95	[[117 29] [7 139]]	0.93
Ensemble method (Fizza)	0.89	-	0.86	-	0.89	0.79	-	0.93	[[124 22] [9 137]]	0.941
SVM (Ximena)	0.88	0.92	0.86	0.87	0.89	0.763	0.84	0.92	[[122 24] [11 135]]	0.88
Random Forest (Sampada)	0.88	0.94	0.84	0.88	0.89	0.77	0.82	0.95	[[179 40] [11 207]]	0.96

Comparison between the unique models

From the table above, it can be seen that the Ensemble technique has the highest accuracy (0.89), followed by XGBoost, SVM, and Random Forest (0.88 each). Random Forest has the highest ROC AUC score of 0.96, demonstrating that it can distinguish between the two class most effectively. XGBoost and Ensemble method also performs well in this metric with the ROC AUC score of 0.93 and 0.941 respectively. For precision, XGBoost (0.94 for class 0) and SVM (0.86 for class 1) outperforms, with Ensemble also performing well with 0.86 for class 1. Random Forest has the highest F1 score (0.88 for class 0, 0.89 for class 1), with MCC score of 0.77, indicating balanced classification. Random Forest achieves the highest recall (0.95 for class 1), meaning it detects 95% of actual stroke cases. This is critical in stroke prediction as missing true positives (strokes) can be life-threatening. The second-best is XGBoost (0.95), closely tied. According to confusion matrix, XGBoost minimizes the number of false negatives (7) while the

Ensemble technique has the lowest number of false positives (22). Overall, Random Forest and XGBoost are best suited for stroke prediction.

5.2 Conclusion and Recommendation

The machine learning models used for stroke prediction meets its primary objective of identifying individuals at high risk of experiencing a stroke based on medical and demographic factors. By integrating the models into healthcare systems, healthcare professionals can have a dependable tool for early intervention, timely medical attention, and personalized prevention intiatives. This proactive strategy can significantly improve stroke prevention and reduce stroke incidence. Key insights from the modeling process revealed that features such residence type, smoking, and gender, initially assumed as critical contributors, were not among the top factors affecting stroke. The analysis emphasizes the importance of data-driven approaches in identifying actual risk factors. Random Forest outperformed amongst the unique models with the highest ROC AUC score (0.96), balanced F1 score (0.89 from stroke cases), and reliable (0.95), making it the most effective for stroke prediction. For neural network models, Fizza's FNN configuration achieved the highest accuracy (0.89) and demonstrated excellent overall performance.

In order to improve prediction, it is recommended to prioritize Random Forest or XGBoost models when avoiding false negatives is critical as both have the lowest false negative rates and ensure reliable identification of true stroke cases. However, if computational efficiency and simplicity are critical, SVM can be considered because it provides competitive performance with fewer hyperparameters and lower computational overhead. Moreover, the model should allow for personalized risk assessments based on medical histories, demographics, and lifestyle factors, while real-time monitoring via wearable devices or mobile application can improve the process by tracking vital parameters and providing timely alerts for inteventions. Further improvements can be made by refining the feature importance analysis to better capture subtle stroke predictors, which will enhance the model's accuracy. Combining standard models with neural networks to develop ensemble techniques can boost robustness and adaptability. Regular updates with real-world patient data are required to keep the model relevant and effective for diverse populations. These recommendations, if implemented, will optimize stroke prediction, support proactive healthcare, and eventually improve patient outcomes and healthcare efficiency.

6. Author Contribution Statement

Section	Team member
Introduction	Fizza and Ashwini
Objectives	Fizza and Ashwini
Literature review	Ashwini
Visualization for introduction	Fizza and Ximena
Preprocessing report section	Ximena
Preprocessing code	Ximena and Sampada
Preprocessing visualization	Sampada
Report structure, finalization and formatting	Sampada

Ashwini- 25%
Ximena- 25%
Sampada- 25%
Fizza- 25%

Signatures:

7. References

GeeksforGeeks, 2024. *Random Forest Algorithm in Machine Learning.* [Online]
Available at: https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/
[Accessed 28 December 2024].

GeeksforGeeks, 2024. *Feedforward neural network.* [Online]
Available at: https://www.geeksforgeeks.org/feedforward-neural-network/
[Accessed 28 December 2024].

Wang, C.-C. W., Kuo, P.-H. & Chen, G.-Y., 2022. *Machine Learning Prediction of Turning Precision Using Optimized XGBoost Model.* [Online]
Available at: https://www.mdpi.com/2076-3417/12/15/7739
[Accessed 29 December 2024].

Gupta, V., 2017. *Understanding Feedforward Neural Networks.* [Online]
Available at: https://learnopencv.com/understanding-feedforward-neural-networks/
[Accessed 29 December 2024].

GeeksforGeeks, 2024. *Feedfoward Neural Network.* [Online]
Available at: https://www.geeksforgeeks.org/feedforward-neural-network/
[Accessed 29 December 2024].

Ohiri, E., 2024. *Feedforward neural networks: everything you need to know.* [Online]
Available at: https://www.cudocompute.com/blog/feedforward-neural-networks-everything-you-need-to-know
[Accessed 29 December 2024].

Organization, W. H., 2022. *World Stroke Day 2022.* [Online]
Available at: https://www.who.int/srilanka/news/detail/29-10-2022-world-stroke-day-2022
[Accessed 19 December 2024].

Feigin, V. L., Norrving, B. & Mensah, G. A., 2021. Global burden of stroke. *Circulation Research,* 120(3), pp. 439-448.

Benjamin, E. J., Muntner, P., Alonso, A. & al, e., 2019. Heart Disease and Stroke Statistics—2019 Update: A Report From the American Heart Association. *Circulation,* 139(10), pp. e56-e528.

Anon., 2018. Roth, G. A.; Johnson, C.; Abajobir, A.; et al. *Journal of the American College of Cardiology,* 70(1), pp. 1-25.

Sacco, R. L., Kasner, S. E., Broderick, J. P. & al, e., 2013. An updated definition of stroke for the 21st century: A statement for healthcare professionals from the American Heart Association/American Stroke Association. *Stroke,* 44(7), pp. 2064-2089.

Ovbiagele, B., Goldstein, L. B., Higashida, R. T. & al, e., 2015. Forecasting the future of stroke in the United States. *Stroke,* 44(8), pp. 2361-2375.

Topol, E. J., 2019. *Deep Medicine: How Artificial Intelligence Can Make Healthcare Human Again.* s.l.:Basic Books.

Chahine Y, M. M. M. B. D. Á. J. B. P. A. N., 2023. Machine learning and the conundrum of stroke risk prediction. *Arrhythmia & Electrophysiology Review,* Volume 12, p. e07.

P.Divya, K. P. B. D. V. K. K. C. R. G., 2025. Hybrid machine learning approach for early stroke prediction in elderly people. *Communications on Applied Nonlinear Analysis,* 32(2s), p. 241.

8. Appendix

Meeting Logs

Meeting No.	Date of Meeting	What did you discuss during the meeting?	The work you have done during the meeting	Who attended?
1	30 Oct 2024	Industries and datasets	Studied the coursework question, looked into different datasets from different industries/topics, and selected two possible data sets	Everyone
2	28 Nov 2024	Selected Dataset	Selected a dataset, discussed possible problem statement, looked into the selected dataset, created	Everyone
3	10 Dec 2024	Data preprocessing, project report discussion, model discussion	Discussed data preprocessing methods, how to balance the data and what changes needs to be made with the dataset, discussed the introduction part of the report, discussed what models can be used, noted questions that needed clarification	Everyone
4	11 Dec 2024	Data preprocessing	SMOTE to create synthetic data for better sampling, discussed what data records could be dropped.	Everyone
5	13 Dec 2024	Complete data preprocessing coding	Completed data processing, also worked on structure and uniformity of the data set	Sampada, Ximena
6	16 Dec 2024	Model Discussion	Decided on the unique and common model that we're going to do for the coursework	Everyone
7	17 Dec 2024	Report preparation - introduction	Discussed problem statements, models chosen, what format to choose.	Ashwini, Fizza
8	20 Dec 2024	Report Introduction	Formatted the document and added required references	Ashwini, Fizza
9	23 Dec 2024	Data pre-processing visualization and report	Discussed and completed the data exploration and visualization, discussed the data pre-processing part of the report	Ximena, Sampada

10	26 Dec 2024	Report, and evaluation metrics discussion	Discussed the report progress till date and things to elaborate and include in report, decided the evaluation metrics to be used	Everyone
11	30 Dec 2024	Model performance discussion and further steps	Discussed how far we've completed modeling and how are our models performing, also discussed further steps regarding the coursework	Everyone
12	1 Jan 2025	Report Finalization	Combined everyone's section in the final report, formatted and completed report for the coursework.	Everyone