Name : Manaswi Kabadi
Class : D15C
Roll no. : 27
Batch : B

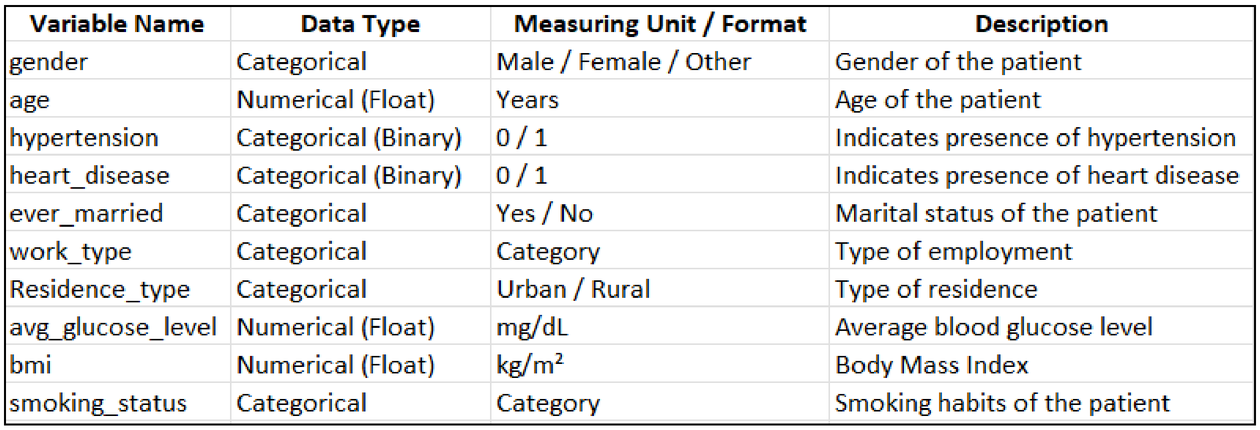
EXP - 3

Aim : Apply Decision Tree and Random Forest for classification tasks

Dataset Source : https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset

Dataset Description :
The **Stroke Prediction Dataset** is a real-world medical dataset that contains information related to demographic, lifestyle, and clinical factors that may influence the occurrence of a stroke. Stroke is a serious health condition caused by an interruption of blood supply to the brain, and early prediction plays an important role in preventive healthcare.
The dataset consists of approximately 5,111 records, where each record represents an individual patient. The dataset is provided in CSV (Comma Separated Values) format and is structured in a tabular form. It contains both numerical and categorical attributes, making it suitable for classification-based machine learning algorithms such as Decision Tree and Random Forest.

Dataset Attributes:



Variable Name	Data Type	Measuring Unit / Format	Description
gender	Categorical	Male / Female / Other	Gender of the patient
age	Numerical (Float)	Years	Age of the patient
hypertension	Categorical (Binary)	0 / 1	Indicates presence of hypertension
heart_disease	Categorical (Binary)	0 / 1	Indicates presence of heart disease
ever_married	Categorical	Yes / No	Marital status of the patient
work_type	Categorical	Category	Type of employment
Residence_type	Categorical	Urban / Rural	Type of residence
avg_glucose_level	Numerical (Float)	mg/dL	Average blood glucose level
bmi	Numerical (Float)	kg/m^2	Body Mass Index
smoking_status	Categorical	Category	Smoking habits of the patient

Target Variables Used in the Experiment
● **stroke** :A binary categorical variable representing whether a patient has experienced a stroke.

- 0 → No Stroke
- 1 → Stroke

This variable is used as the **target for both Decision Tree and Random Forest classification algorithms**.

Theory

A Decision Tree is a supervised machine learning algorithm used for classification and regression tasks. In classification problems, it predicts the class label by learning a set of decision rules derived from the input features. The model works by recursively splitting the dataset into smaller subsets based on feature values, forming a tree-like structure.
Each internal node of the tree represents a feature-based decision, each branch represents the outcome of that decision, and each leaf node represents a final class label. Decision Trees are easy to interpret and visually explain, which makes them particularly useful in medical decision-making systems.
In this experiment, the Decision Tree classifier is used to predict whether a person is likely to suffer a stroke (0 = No, 1 = Yes) based on medical and lifestyle attributes such as age, hypertension, glucose level, BMI, and smoking status.

Mathematical Formulation of the Algorithm :

Decision Trees split data by selecting the feature that best separates the classes at each node. This selection is based on **impurity measures**, commonly **Entropy** or **Gini Index**.

Entropy

Entropy measures the randomness or impurity in a dataset.

$$Entropy(S) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

Where:

- pi is the proportion of samples belonging to class i
- c is the number of classes

Information Gain

Information Gain measures the reduction in entropy after a dataset is split on a feature.

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

The feature with the **highest information gain** is selected for splitting.

Gini Index

Another commonly used impurity measure is the Gini Index:

$$Gini(S) = 1 - \sum_{i=1}^{c} p_i^2$$

The feature with the **lowest Gini Index** is chosen for the split.

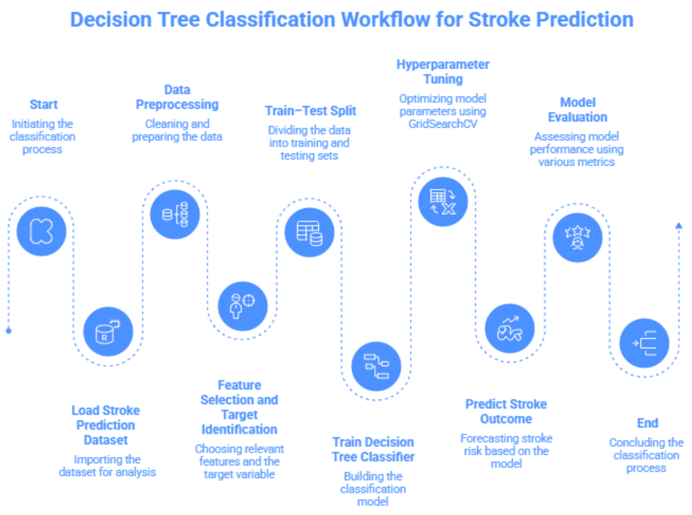
Algorithm Limitations :

- **Overfitting**
  Decision Trees tend to grow very deep and may learn noise from the training data.
  Condition of failure: When the tree becomes too complex, it performs well on training data but poorly on unseen data.
- **High Variance**
  Small changes in the dataset can lead to a completely different tree structure.
  Condition of failure: The model becomes unstable and sensitive to data variations.
- **Bias Toward Dominant Features**
  Decision Trees may favor features with more levels or unique values.
  Condition of failure: Important features with fewer categories may be ignored.

Methodology / Workflow :

Data Collection:
The Stroke Prediction dataset is loaded from the CSV file. It contains medical, demographic, and lifestyle-related attributes along with the target variable indicating stroke occurrence.
Data Preprocessing:
Categorical features such as gender, work type, residence type, and smoking status are converted into numerical form using encoding techniques. Missing values, particularly in the BMI attribute, are handled appropriately to ensure data completeness.
Feature Selection:
All relevant medical and lifestyle attributes are selected as independent variables, while the `stroke` attribute is chosen as the target variable.
Train–Test Split:
The dataset is divided into training and testing sets using an 80:20 ratio. This helps in evaluating the model’s performance on unseen data.

Model Training:
A Decision Tree classifier is trained on the training dataset. The model learns a set of decision rules by recursively splitting the data based on feature values.
Prediction:
The trained Decision Tree model is used to predict stroke occurrence for the test dataset.
Model Evaluation:
The performance of the Decision Tree classifier is evaluated using classification metrics such as accuracy, confusion matrix, precision, recall, and F1-score.


Decision Tree Classification Workflow for Stroke Prediction
Start
Initiating the classification process
Load Stroke Prediction Dataset
Importing the dataset for analysis
Data Preprocessing
Cleaning and preparing the data
Feature Selection and Target Identification
Choosing relevant features and the target variable
Train–Test Split
Dividing the data into training and testing sets
Train Decision Tree Classifier
Building the classification model
Hyperparameter Tuning
Optimizing model parameters using GridSearchCV
Predict Stroke Outcome
Forecasting stroke risk based on the model
Model Evaluation
Assessing model performance using various metrics
End
Concluding the classification process

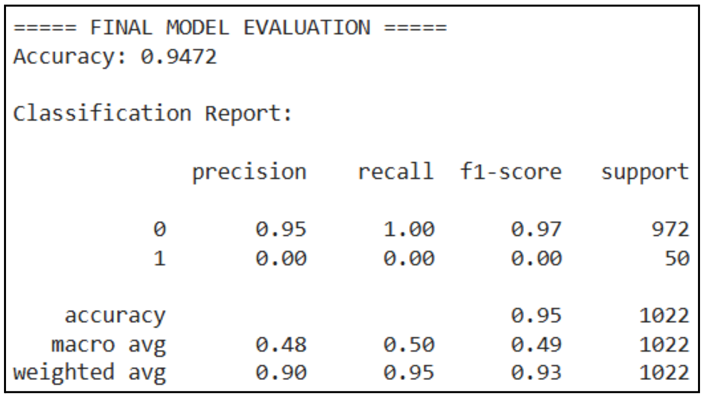
CODE :

```
# Decision Tree Classifier
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection
import train_test_split,
GridSearchCV
from sklearn.tree import
DecisionTreeClassifier, plot_tree
from sklearn.preprocessing import
LabelEncoder
from sklearn.metrics import
accuracy_score,
classification_report,
confusion_matrix,
ConfusionMatrixDisplay
# Load Dataset
df=
pd.read_csv("healthcare-dataset-st
roke-data.csv")
# Data Preprocessing
df.drop("id", axis=1,
inplace=True)
# Handle missing values (BMI)
df["bmi"] =
df["bmi"].fillna(df["bmi"].mean())
# Encode categorical variables
le = LabelEncoder()
for col in df.columns:
    if df[col].dtype == "object":
```

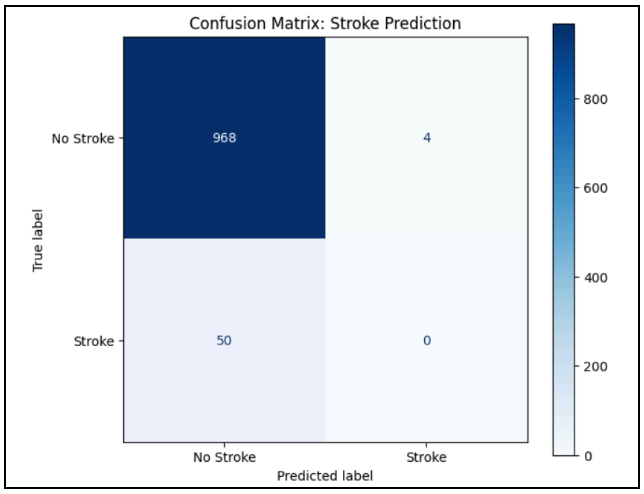
```
                        df[col]   =
le.fit_transform(df[col])
# Features & Target
X = df.drop("stroke", axis=1)
y = df["stroke"]
#Train-Test Split
X_train, X_test, y_train, y_test =
train_test_split(
           X,   y,   test_size=0.2,
random_state=42, stratify=y)
# Decision Tree Model
dtree                             =
DecisionTreeClassifier(random_stat
e=42)
# Hyperparameter Grid
param_grid = {
            "criterion":   ["gini",
"entropy"],
     "max_depth": [None, 5, 10, 15,
20],
      "min_samples_split":  [2,  10,
20],
       "min_samples_leaf":  [1,  5,
10]}

#Grid Search
grid_search = GridSearchCV(
    estimator=dtree,
    param_grid=param_grid,
    cv=5,
    n_jobs=-1,
    verbose=1)
grid_search.fit(X_train, y_train)
print("=====    BEST    PARAMETERS
=====")
print(grid_search.best_params_)
print(f"Best      Cross-Validation
Accuracy:
{grid_search.best_score_:.4f}")
```

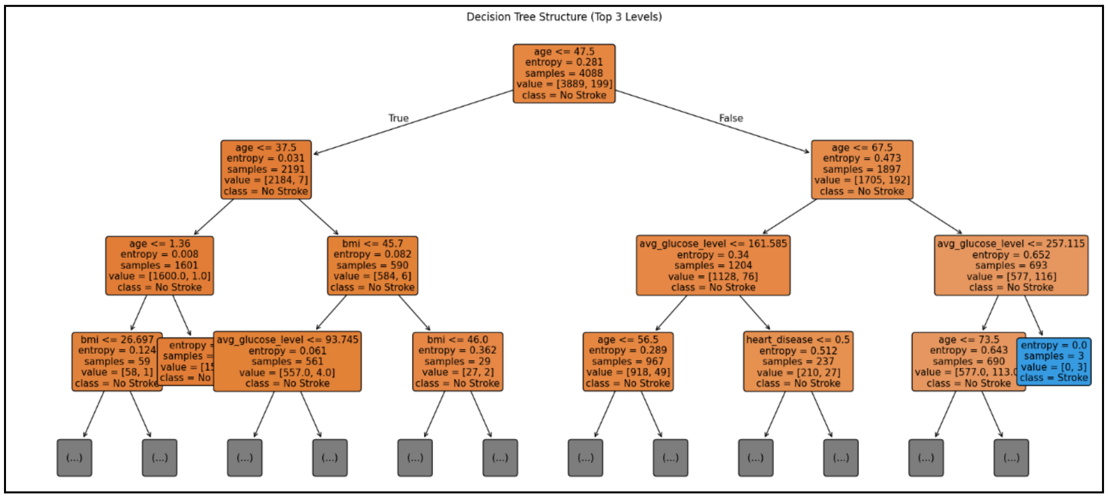
```
# Best Model
best_model                        =
grid_search.best_estimator_

#Prediction
y_pred                            =
best_model.predict(X_test)

# Evaluation
print("\n=====     FINAL     MODEL
EVALUATION =====")
print(f"Accuracy:
{accuracy_score(y_test,
y_pred):.4f}")
print("\nClassification
Report:\n")
print(classification_report(y_test
, y_pred))

#Confusion Matrix
cm    =    confusion_matrix(y_test,
y_pred)
disp = ConfusionMatrixDisplay(
    confusion_matrix=cm,
      display_labels=["No  Stroke",
"Stroke"])

fig, ax = plt.subplots(figsize=(7,
6))
disp.plot(cmap="Blues", ax=ax)
plt.title("Confusion       Matrix:
Stroke Prediction")
plt.show()
#Tree Visualization (Top 3 Levels)
plt.figure(figsize=(22, 10))
plot_tree(
    best_model,
    max_depth=3,
    feature_names=X.columns,
        class_names=["No  Stroke",
"Stroke"],
    filled=True,
    rounded=True,
    fontsize=11)
plt.title("Decision Tree Structure
(Top 3 Levels)")
plt.show()
# Feature Importance
importances                         =
best_model.feature_importances_
feature_importance_df               =
pd.DataFrame({
    "Feature": X.columns,
    "Importance": importances
}).sort_values(by="Importance",
ascending=False)
plt.figure(figsize=(10, 6))
sns.barplot(
    x="Importance",
    y="Feature",
    data=feature_importance_df,
    palette="viridis",
    hue="Feature",
    legend=False
)
plt.title("Feature  Importance  for
Stroke Prediction")
plt.show()
```

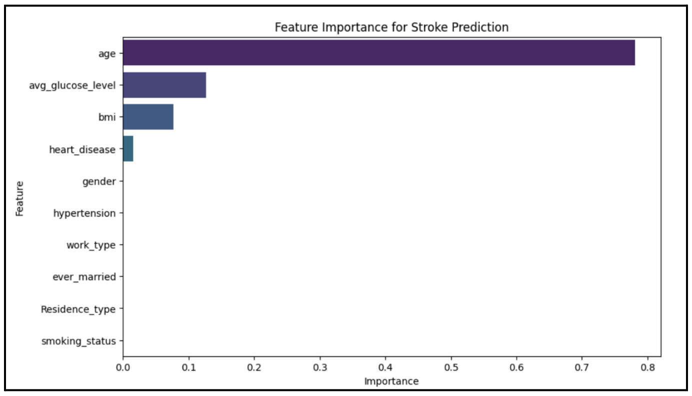
OUTPUT :



```
===== FINAL MODEL EVALUATION =====
Accuracy: 0.9472

Classification Report:

              precision    recall  f1-score   support

           0       0.95      1.00      0.97       972
           1       0.00      0.00      0.00        50

    accuracy                           0.95      1022
   macro avg       0.48      0.50      0.49      1022
weighted avg       0.90      0.95      0.93      1022
```







Performance Analysis

The Decision Tree classifier achieved an accuracy of 94.72% on the test dataset. The model performed very well in predicting non-stroke cases, with a recall of 1.00 for class 0. However, it failed to correctly identify stroke cases, resulting in zero precision and recall for class 1. This occurred due to the severe class imbalance in the dataset. Although the overall accuracy is high, the low macro-average F1-score (0.49) indicates poor minority class prediction.

Hyperparameter Tuning

Hyperparameter tuning using GridSearchCV evaluated 90 parameter combinations. The optimal parameters selected were criterion = entropy, max_depth = 5, min_samples_split = 10, and min_samples_leaf = 1, achieving a best cross-validation accuracy of 95.03%. Limiting tree depth helped reduce overfitting but did not improve stroke class prediction.

<u>RANDOM FOREST</u>

Theory

Random Forest is an ensemble learning algorithm that builds multiple Decision Trees and combines their predictions to produce a final output. Instead of relying on a single tree, Random Forest creates a collection (forest) of trees and uses majority voting for classification tasks. This approach significantly improves prediction accuracy and reduces overfitting.
Each tree in a Random Forest is trained on a random subset of the training data and considers a random subset of features at each split. This randomness ensures diversity among the trees, making the overall model more robust and stable.
In this experiment, Random Forest is applied to the Stroke Prediction dataset to classify whether an individual is at risk of stroke (0 = No Stroke, 1 = Stroke) using medical and lifestyle attributes.

Mathematical Formulation of the Algorithm :

Random Forest does not rely on a single mathematical equation; instead, it follows an ensemble-based procedure:

1. From the original training dataset, multiple bootstrap samples are created using random sampling with replacement.
2. For each bootstrap sample, a Decision Tree is trained.
3. At each split in a tree, only a random subset of features is considered.
4. Each tree predicts a class label.
5. The final prediction is obtained using majority voting:

$$\hat{y} = \mathrm{mode}(y_1, y_2, \ldots, y_n)$$

Where:

- y1,y2,…,yn are predictions from individual trees
- y^ is the final predicted class

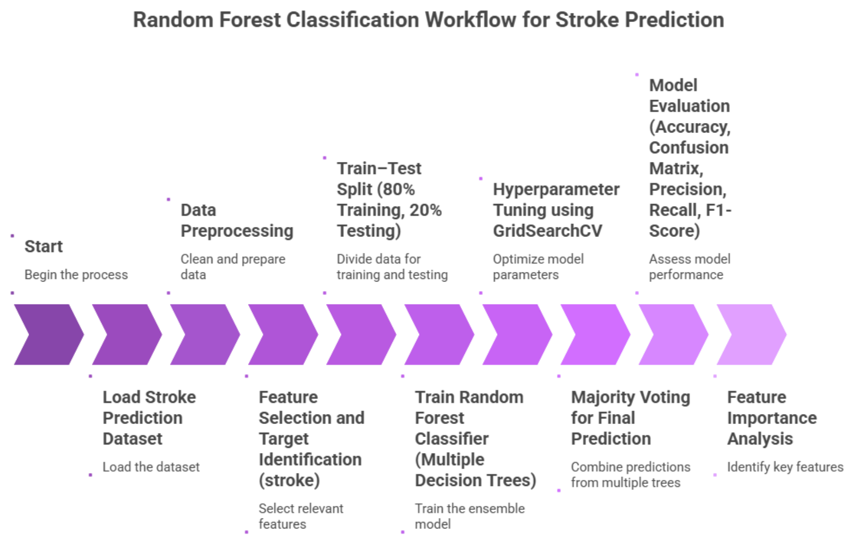
Algorithm Limitations :

- **Reduced Interpretability :** Since Random Forest consists of many trees, it is harder to interpret compared to a single Decision Tree.**Condition of failure:** Individual decision rules are not easily traceable.
- **Computational Cost :** Training many trees increases memory usage and computation time.**Condition of failure:** Performance may degrade on very large datasets or low-resource systems.
- **Bias in Imbalanced Datasets :** Random Forest can be biased toward the majority class. **Condition of failure:** Without proper handling, minority class predictions may be poor.

Methodology / Workflow :

1. **Data Collection:**
   The same Stroke Prediction dataset is used to maintain consistency and enable fair comparison with the Decision Tree model.
2. **Data Preprocessing:**
   Categorical attributes are encoded into numerical values, and missing values are handled. Since Random Forest is a tree-based model, feature scaling is not mandatory.
3. **Feature Selection:**
   All relevant attributes are selected as input features, and the `stroke` column is used as the target variable.

4. **Train–Test Split:**
 The dataset is split into 80% training data and 20% testing data.
5. **Model Training:**
 A Random Forest classifier is trained by constructing multiple Decision Trees using bootstrap samples of the training data. At each split, a random subset of features is considered to ensure diversity among trees.
6. **Prediction:**
 Each tree in the forest predicts a class label, and the final prediction is determined using majority voting.
7. **Model Evaluation:**
 The Random Forest classifier is evaluated using accuracy, confusion matrix, precision, recall, and F1-score. Feature importance scores are also analyzed to identify key factors influencing stroke prediction.


Random Forest Classification Workflow for Stroke Prediction
Start
Begin the process
Load Stroke Prediction Dataset
Load the dataset
Data Preprocessing
Clean and prepare data
Feature Selection and Target Identification (stroke)
Select relevant features
Train–Test Split (80% Training, 20% Testing)
Divide data for training and testing
Train Random Forest Classifier (Multiple Decision Trees)
Train the ensemble model
Hyperparameter Tuning using GridSearchCV
Optimize model parameters
Majority Voting for Final Prediction
Combine predictions from multiple trees
Model Evaluation (Accuracy, Confusion Matrix, Precision, Recall, F1-Score)
Assess model performance
Feature Importance Analysis
Identify key features

CODE :

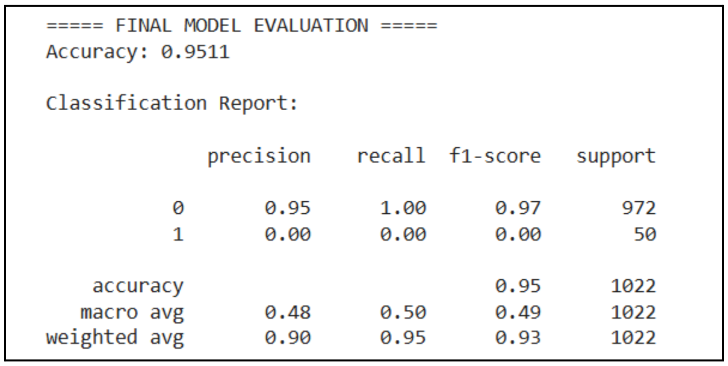
```
# Random Forest Classifier
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from        sklearn.model_selection
import              train_test_split,
GridSearchCV
from    sklearn.ensemble    import
RandomForestClassifier
from  sklearn.preprocessing  import
LabelEncoder
from      sklearn.metrics     import
accuracy_score,
classification_report,
confusion_matrix,
ConfusionMatrixDisplay
# Load Dataset
df                                  =
pd.read_csv("healthcare-dataset-st
roke-data.csv")
```

```
# Data Preprocessing
df.drop("id",              axis=1,
inplace=True)

# Handle missing BMI values
df["bmi"]                         =
df["bmi"].fillna(df["bmi"].mean())
# Encode categorical variables
le = LabelEncoder()
for col in df.columns:
    if df[col].dtype == "object":
                        df[col]   =
le.fit_transform(df[col])
# Features & Target
X = df.drop("stroke", axis=1)
y = df["stroke"]
# Train-Test Split
X_train, X_test, y_train, y_test =
train_test_split(
           X,   y,   test_size=0.2,
random_state=42, stratify=y)
# Random Forest Model
rf                                =
RandomForestClassifier(random_stat
e=42)
# Hyperparameter Grid
param_grid = {
    "n_estimators": [100, 200],
    "max_depth": [None, 10, 20],
    "min_samples_split": [2, 10],
    "min_samples_leaf": [1, 5]}
#Grid Search
grid_search = GridSearchCV(
    estimator=rf,
    param_grid=param_grid,
    cv=5,
    n_jobs=-1,
    verbose=1)
grid_search.fit(X_train, y_train)
```

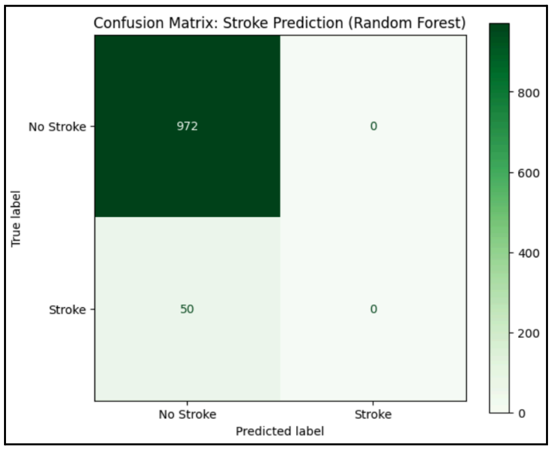
```
print("=====    BEST    PARAMETERS
=====")
print(grid_search.best_params_)
print(f"Best      Cross-Validation
Accuracy:
{grid_search.best_score_:.4f}")
# Best Model
best_model                        =
grid_search.best_estimator_
# Prediction
y_pred                            =
best_model.predict(X_test)
# Evaluation
print("\n=====     FINAL     MODEL
EVALUATION =====")
print(f"Accuracy:
{accuracy_score(y_test,
y_pred):.4f}")
print("\nClassification
Report:\n")
print(classification_report(y_test
, y_pred, zero_division=0))
# Confusion Matrix
cm    =    confusion_matrix(y_test,
y_pred)
disp = ConfusionMatrixDisplay(
    confusion_matrix=cm,
      display_labels=["No  Stroke",
"Stroke"])
fig, ax = plt.subplots(figsize=(7,
6))
disp.plot(cmap="Greens", ax=ax)
plt.title("Confusion       Matrix:
Stroke     Prediction      (Random
Forest)")
plt.show()
# Feature Importance
importances                       =
best_model.feature_importances_
```

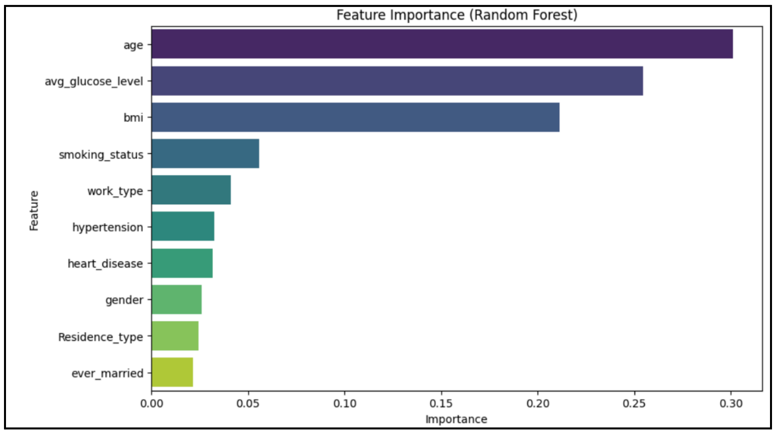
```
feature_importance_df =
pd.DataFrame({
    "Feature": X.columns,
    "Importance": importances
}).sort_values(by="Importance",
ascending=False)
plt.figure(figsize=(10, 6))
sns.barplot(
    x="Importance",
    y="Feature",
    data=feature_importance_df,
    palette="viridis",
    hue="Feature",
    legend=False)
plt.title("Feature Importance
(Random Forest)")
plt.show()
```

OUTPUT



```
===== FINAL MODEL EVALUATION =====
Accuracy: 0.9511

Classification Report:

              precision    recall  f1-score   support

           0       0.95      1.00      0.97       972
           1       0.00      0.00      0.00        50

    accuracy                           0.95      1022
   macro avg       0.48      0.50      0.49      1022
weighted avg       0.90      0.95      0.93      1022
```





Performance Analysis

The Random Forest classifier achieved a slightly higher accuracy of 95.11% and showed better stability during cross-validation (95.13%). Similar to the Decision Tree, it classified non-stroke cases accurately but failed to detect stroke cases. This confirms that ensemble learning alone does not resolve class imbalance issues. Despite higher accuracy, minority class prediction remains weak.

Hyperparameter Tuning

GridSearchCV evaluated 24 parameter combinations for Random Forest. The best parameters were n_estimators = 100, max_depth = None, min_samples_split = 2, and min_samples_leaf = 5, with a best cross-validation accuracy of 95.13%. Tuning improved model stability but did not address class imbalance.

Conclusion :

In this experiment, Decision Tree and Random Forest classifiers were implemented on the Stroke Prediction dataset to classify individuals at risk of stroke. Both models achieved high overall accuracy, mainly due to correct prediction of non-stroke cases. However, due to severe class imbalance, neither model was effective in identifying stroke cases. Random Forest showed slightly better stability and accuracy compared to Decision Tree. The experiment highlights that while tree-based models perform well for classification, additional techniques are required to handle imbalanced medical datasets effectively.