NJIT

New Jersey Institute of Technology

**Final-Term Project Report**

# Supervised Data Mining (Binary Classification)

**NAME:** Sai Teja Reddy Bhumireddy
**NJIT UCID:** sb2744
**Email Address:** sb2744@njit.edu
**Professor:** Yasser Abduallah
CS 634 - 104 Data Mining

## Contents

Abstract: 3

Introduction 3

Project Workflow 3

Data Collection 3

Data Cleaning and Preparation 3

Model Training 10

Function for calculation of metrics using confusion matrix. 11

Finding Best parameters for models using GridSearchCV 11

Function to create LSTM model. 12

Model Training with Kfold. 12

Model Evaluation. 13

Calculating average metrics for each model after all folds 18

Computing ROC curve and ROC area for each model. 19

Results: 20

Conclusion 21

Recommendations: 21

Steps to run the Program: 22

STEP 1: Cloning the Repository 22

STEP 2: Create Virtual Environment. 22

STEP 3: Running the Notebook Locally 23

Alternative for STEP 3: Running the Notebook on Google Colab 23

References and Links: 23

## Abstract:

This project aims to predict stroke occurrence utilizing a dataset derived from a healthcare study, comprising 5,110 instances with various health indicators and personal characteristics. These features include age, gender, occupation, living arrangement, average glucose level, body mass index (BMI), and smoking status. Employing Random Forest, a deep learning model like LSTM, and a third algorithm selected from a predefined list such as Support Vector Machines, the objective is to classify individuals based on their stroke risk. The models' performances are assessed using accuracy metrics obtained through 10-fold cross-validation. The project aims to identify significant predictors of stroke and assess the effectiveness of different classification algorithms in forecasting health outcomes.

## Introduction

Stroke remains a significant global health concern, often leading to death or long-term disability. Early identification of individuals at high risk is crucial for effective intervention and prevention strategies. This project utilizes a healthcare dataset comprising vital attributes including medical history, biometric data, and lifestyle factors essential for assessing stroke risk. Leveraging machine learning techniques on this dataset offers a promising avenue for stroke prediction. The study aims to compare three classification algorithms: Random Forest, a deep learning algorithm from the LSTM family, and a classical machine learning algorithm like Support Vector Machines. Special attention is given to their predictive accuracy and their ability to handle imbalanced datasets commonly encountered in medical data, where stroke events are relatively rare. This endeavor contributes to the broader integration of predictive analytics in healthcare, fostering early diagnosis and personalized medicine approaches.

## Project Workflow

### Data Collection

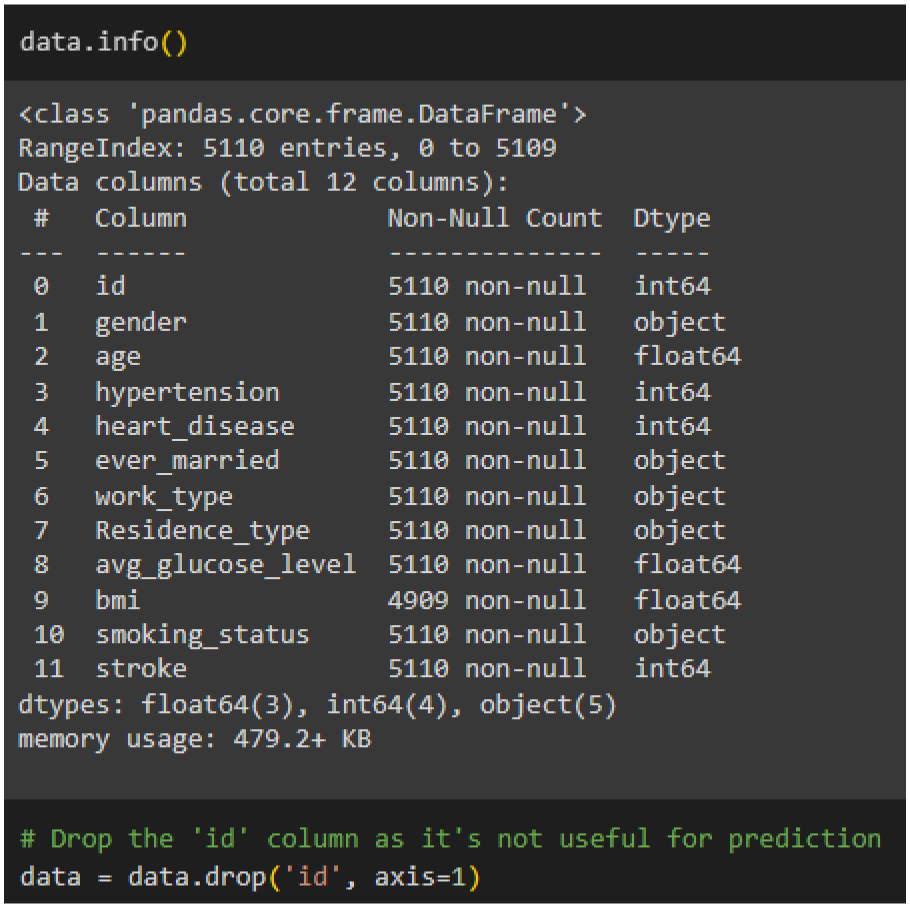
The dataset used in this project was sourced from the Stroke Prediction Dataset on Kaggle.com, encompassing information on 5,110 individuals. It comprises diverse attributes such as gender, age, hypertension, heart disease, marital status, occupation, living arrangement, average glucose level, body mass index (BMI), smoking status, and stroke occurrence.

```
data = pd.read_csv('C:/Users/kanna/Desktop/Sai Teja Reddy Bhumireddy_Final Term Project/Stroke Data.csv')
data.head()
```

```
      id  gender   age  hypertension  heart_disease  ever_married      work_type  Residence_type  avg_glucose_level   bmi   smoking_status  stroke
0   9046    Male  67.0             0              1           Yes        Private           Urban             228.69  36.6  formerly smoked       1
1  51676  Female  61.0             0              0           Yes  Self-employed           Rural             202.21   NaN     never smoked       1
2  31112    Male  80.0             0              1           Yes        Private           Rural             105.92  32.5     never smoked       1
3  60182  Female  49.0             0              0           Yes        Private           Urban             171.23  34.4           smokes       1
4   1665  Female  79.0             1              0           Yes  Self-employed           Rural             174.12  24.0     never smoked       1
```

### Data Cleaning and Preparation

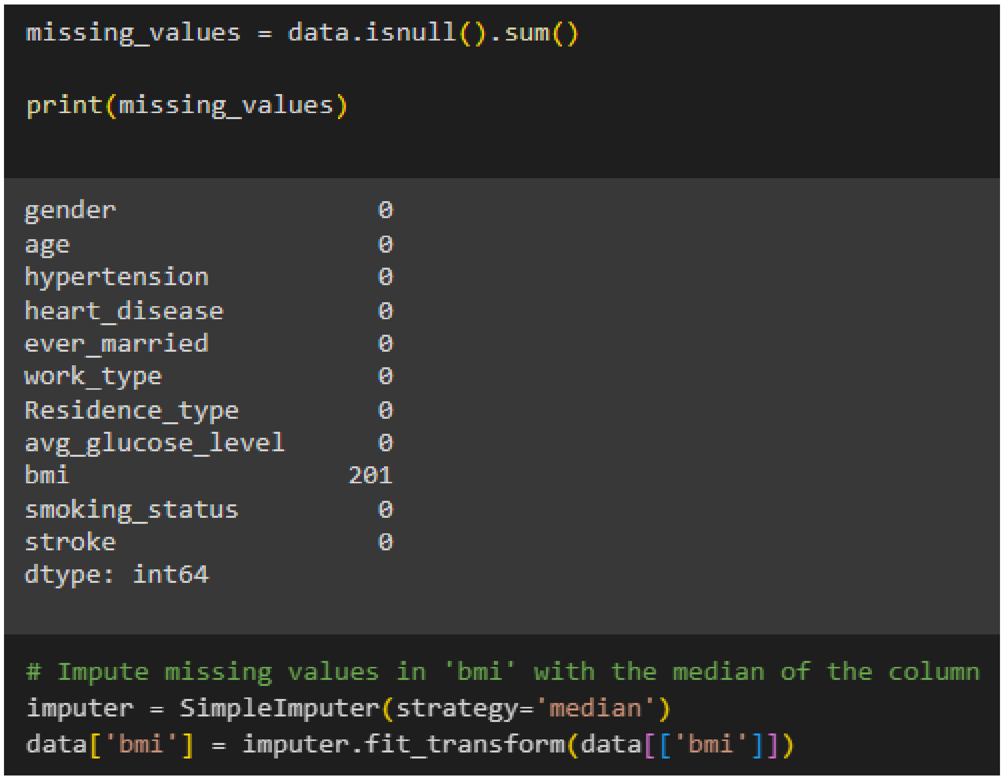
The dataset underwent preprocessing to handle missing values, primarily focusing on the BMI attribute. Categorical variables were adequately encoded to enable seamless analysis. Additionally, data normalization was conducted on continuous variables like age and average glucose level to standardize their ranges, thereby improving the model's performance.



```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 5110 non-null   int64  
 1   gender             5110 non-null   object 
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64  
 4   heart_disease      5110 non-null   int64  
 5   ever_married       5110 non-null   object 
 6   work_type          5110 non-null   object 
 7   Residence_type     5110 non-null   object 
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object 
 11  stroke             5110 non-null   int64  
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

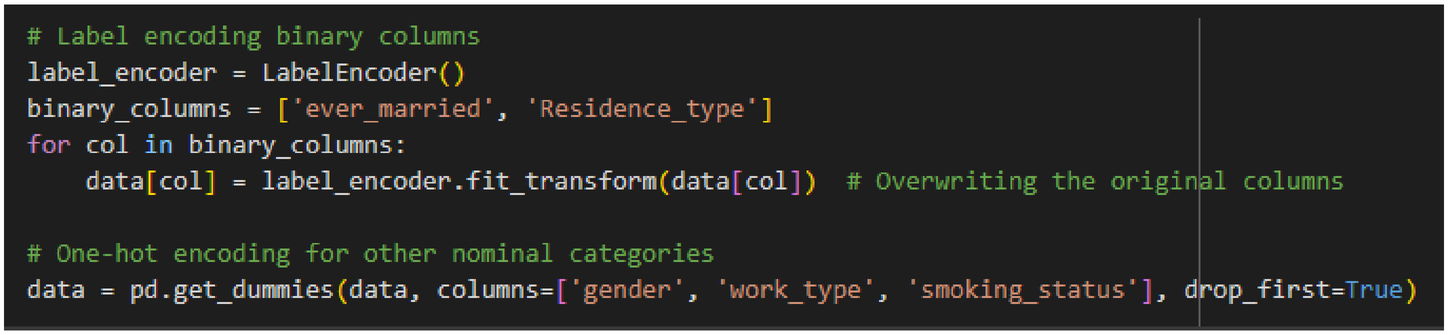
```python
# Drop the 'id' column as it's not useful for prediction
data = data.drop('id', axis=1)
```



```python
missing_values = data.isnull().sum()

print(missing_values)
```

```
gender                 0
age                    0
hypertension           0
heart_disease          0
ever_married           0
work_type              0
Residence_type         0
avg_glucose_level      0
bmi                  201
smoking_status         0
stroke                 0
dtype: int64
```

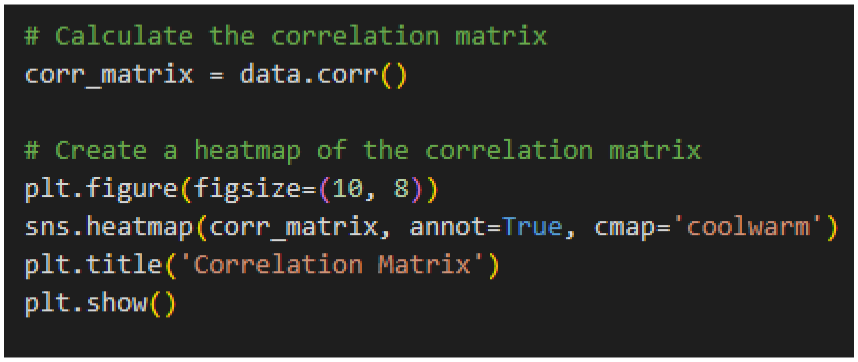
```python
# Impute missing values in 'bmi' with the median of the column
imputer = SimpleImputer(strategy='median')
data['bmi'] = imputer.fit_transform(data[['bmi']])
```



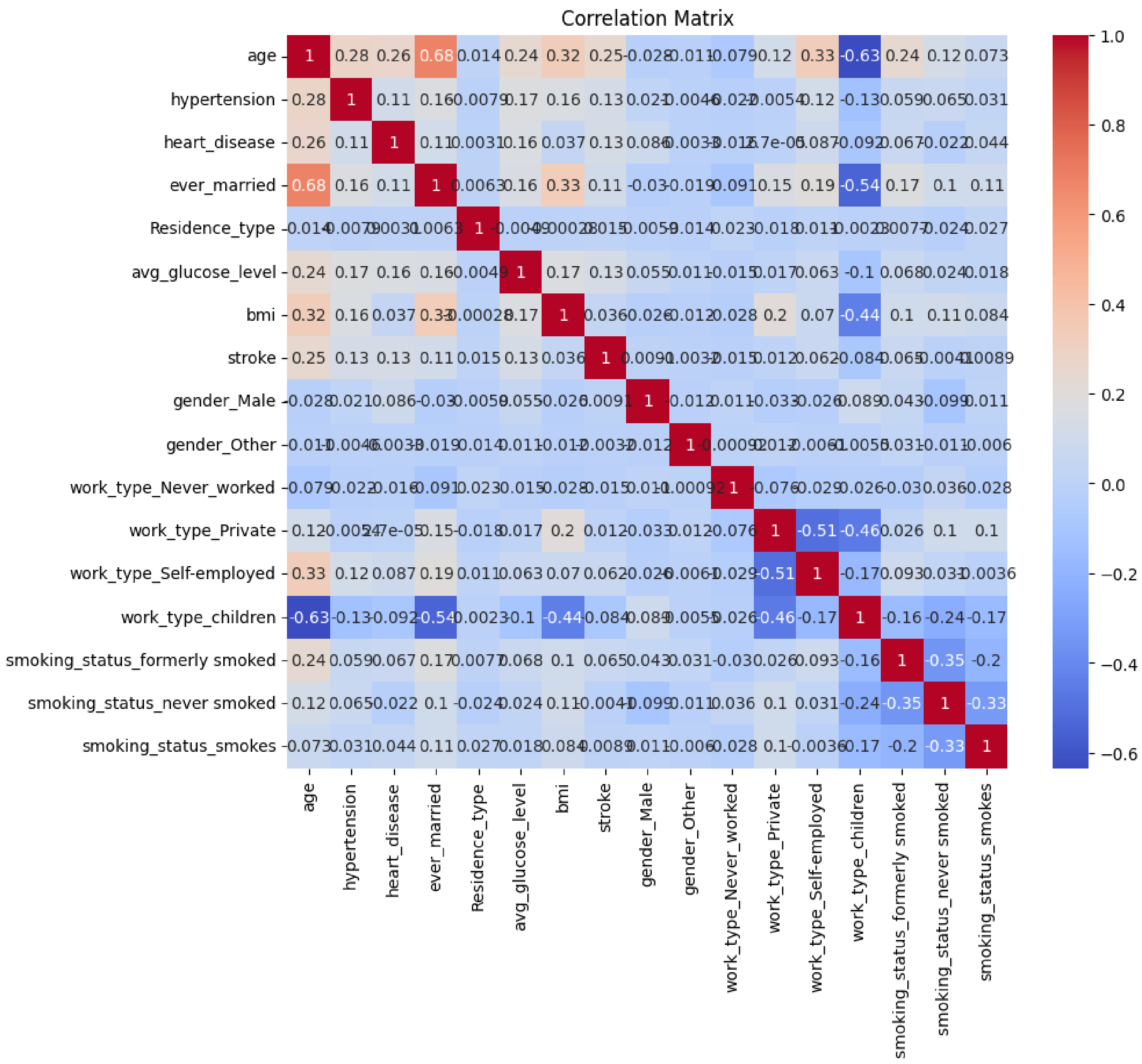
```python
# Label encoding binary columns
label_encoder = LabelEncoder()
binary_columns = ['ever_married', 'Residence_type']
for col in binary_columns:
    data[col] = label_encoder.fit_transform(data[col])  # Overwriting the original columns

# One-hot encoding for other nominal categories
data = pd.get_dummies(data, columns=['gender', 'work_type', 'smoking_status'], drop_first=True)
```

## Exploratory Data Analysis (EDA)

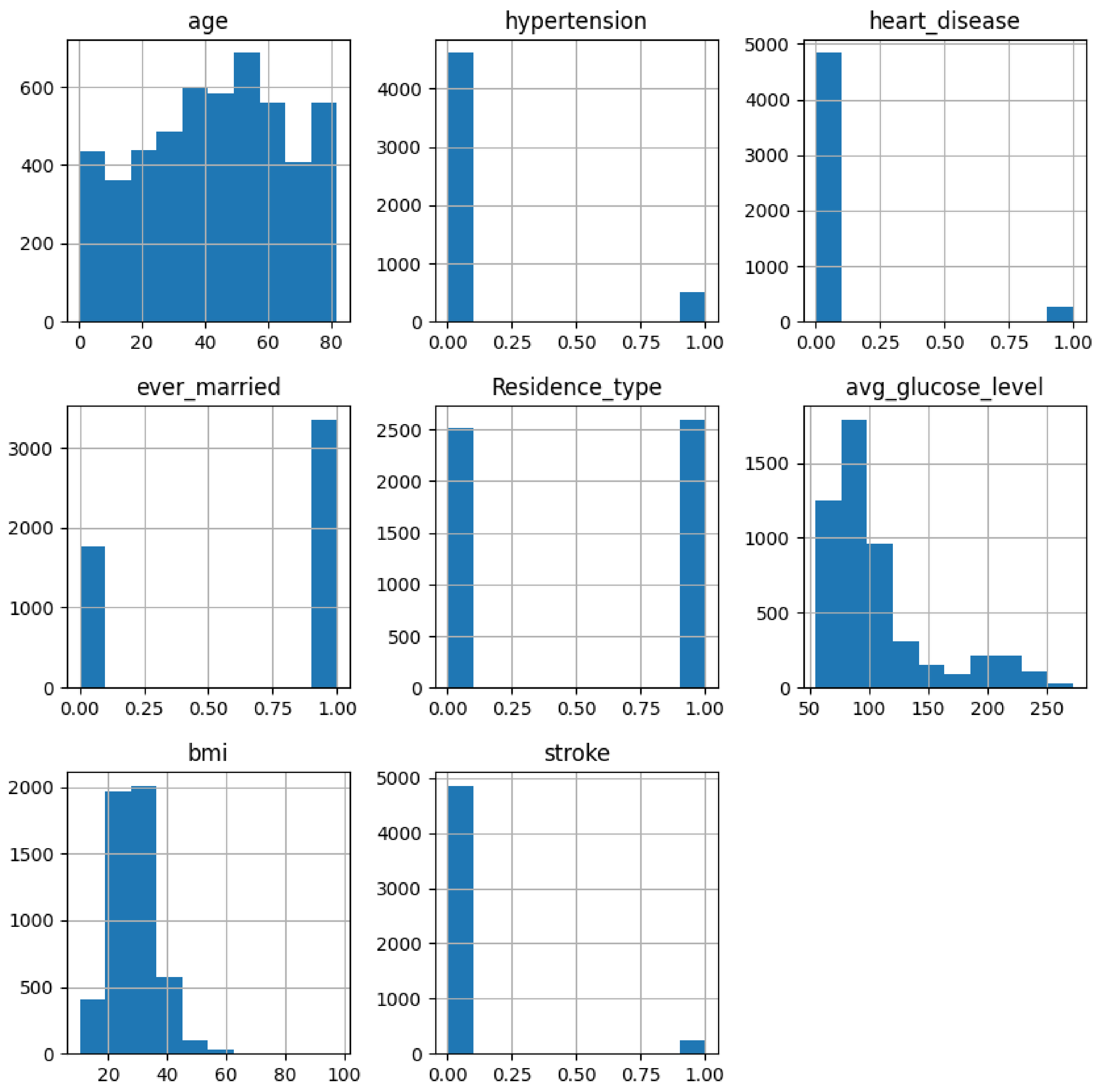
An initial analysis was performed to grasp the distribution and interrelationships among variables. Visualizations including histograms, box plots, and scatter plots were employed to detect patterns and anomalies in the data, guiding the feature selection procedure.



```python
# Calculate the correlation matrix
corr_matrix = data.corr()

# Create a heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



```
data.hist(figsize=(10, 10))
plt.show()
```



**Age Distribution:** The initial chart displays a relatively uniform distribution of ages, with a slight rise observed around the 50-60 age range. This suggests that the dataset encompasses individuals spanning a broad age range, less likely to be from a specific demographic group (such as school students or retirees) and more likely to represent a general population.

**Hypertension:** The distribution reveals that a significant majority of individuals in the dataset do not have hypertension, with only a small fraction exhibiting this condition.

**Heart Disease:** Likewise, the chart depicting heart disease indicates that only a small proportion of individuals have this condition compared to those who do not.

**Marital Status (Ever Married):** The chart implies that the majority of individuals in the dataset have been married at least once.

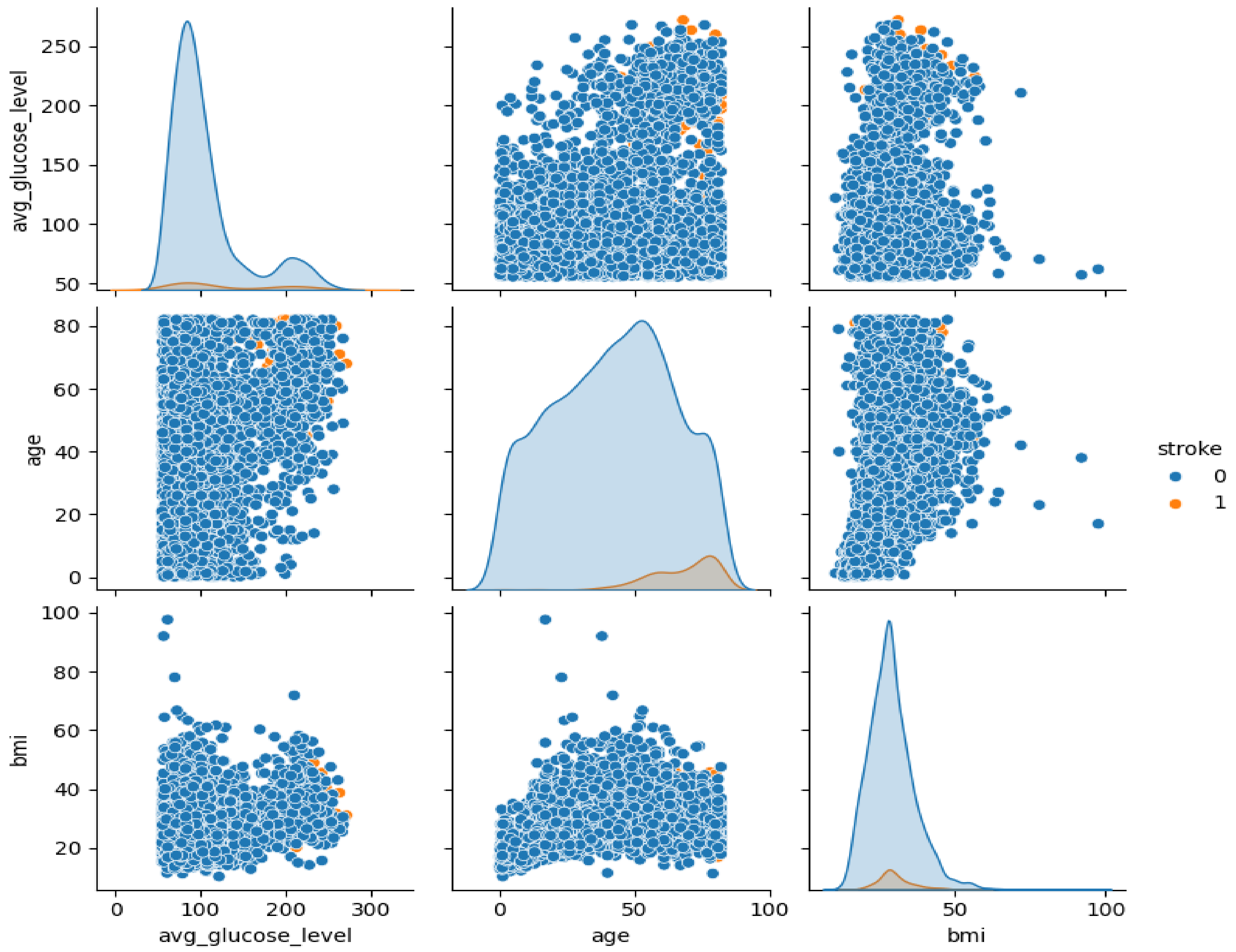
**Residence Type:** The data appears to be nearly evenly divided between two types of residences, potentially urban and rural.

**Average Glucose Level:** The histogram illustrates a right-skewed distribution, indicating that the majority of individuals have glucose levels on the lower side, with a long tail extending towards higher glucose levels. The peak of the distribution is observed around the 75-100 mg/dL range, which is within the normal range.

**BMI:** Similarly, this histogram shows a right-skewed distribution, where the majority of individuals fall within the BMI range considered normal or overweight (around 25-30), with fewer individuals in the underweight and obese categories.

**Stroke:** Like hypertension and heart disease, the vast majority of individuals have not experienced a stroke.

```
sns.pairplot(data[['avg_glucose_level','age','bmi','stroke']], hue='stroke')
plt.show()
```



**Average Glucose Level:** The distribution of average glucose levels is skewed to the right, with a peak in the normal range, yet some individuals exhibit extremely high levels. When juxtaposed with instances of strokes, higher glucose levels seem to correlate with a higher incidence of strokes, as indicated by the presence of orange dots.

**Age:** The age distribution spans broadly with a slight skew to the right, indicating the inclusion of elderly individuals in the population. The scatter plots demonstrate a discernible trend: the probability of stroke rises with age.

**BMI:** The distribution of BMI is likewise right-skewed, with the majority of data points falling within the range considered normal or overweight. Although the relationship between BMI and stroke is not as evident from this plot, instances of strokes are observable across all BMI values.

**Relationship Between Variables:**

**Age and Average Glucose Level:** The scatter plot depicting age against average glucose level does not reveal any distinct pattern or correlation. This lack of discernible trend suggests that within this dataset, there is no clear relationship between age and glucose level.

**Age and BMI:** Similarly, the scatter plot depicting age against BMI does not show a clear correlation.

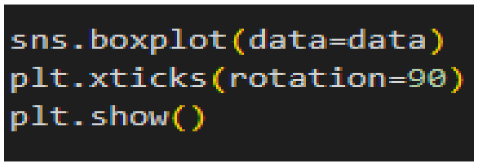
**Average Glucose Level and BMI:** The scatter plot does not exhibit a clear correlation between average glucose levels and BMI.

**Stroke Occurrences:**

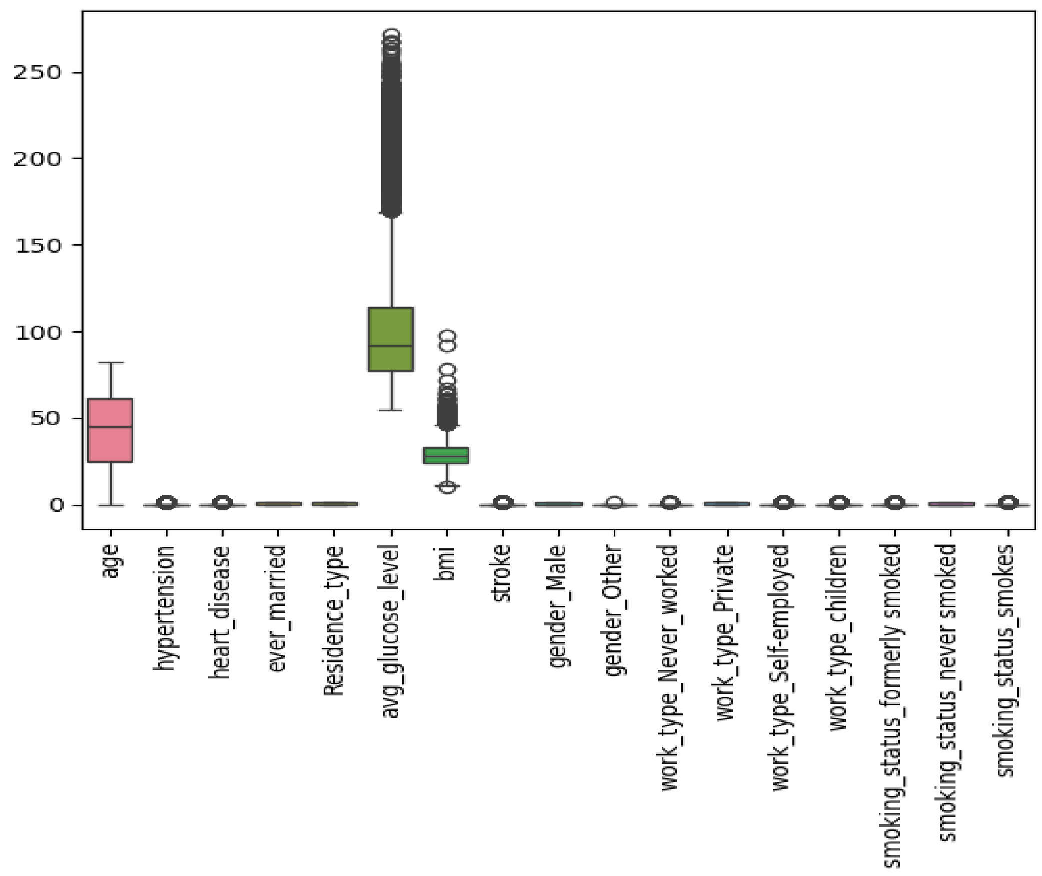
**Strokes and Average Glucose Level:** A conspicuous cluster of strokes (depicted as orange dots) is observable at higher glucose levels.

**Strokes and Age:** Strokes are more common in older age groups.

**Strokes and BMI:** Strokes are scattered across the range of BMI, indicating that within this dataset, BMI is not a clear-cut predictor for strokes.



```
sns.boxplot(data=data)
plt.xticks(rotation=90)
plt.show()
```



## Model Training

Three models were trained:

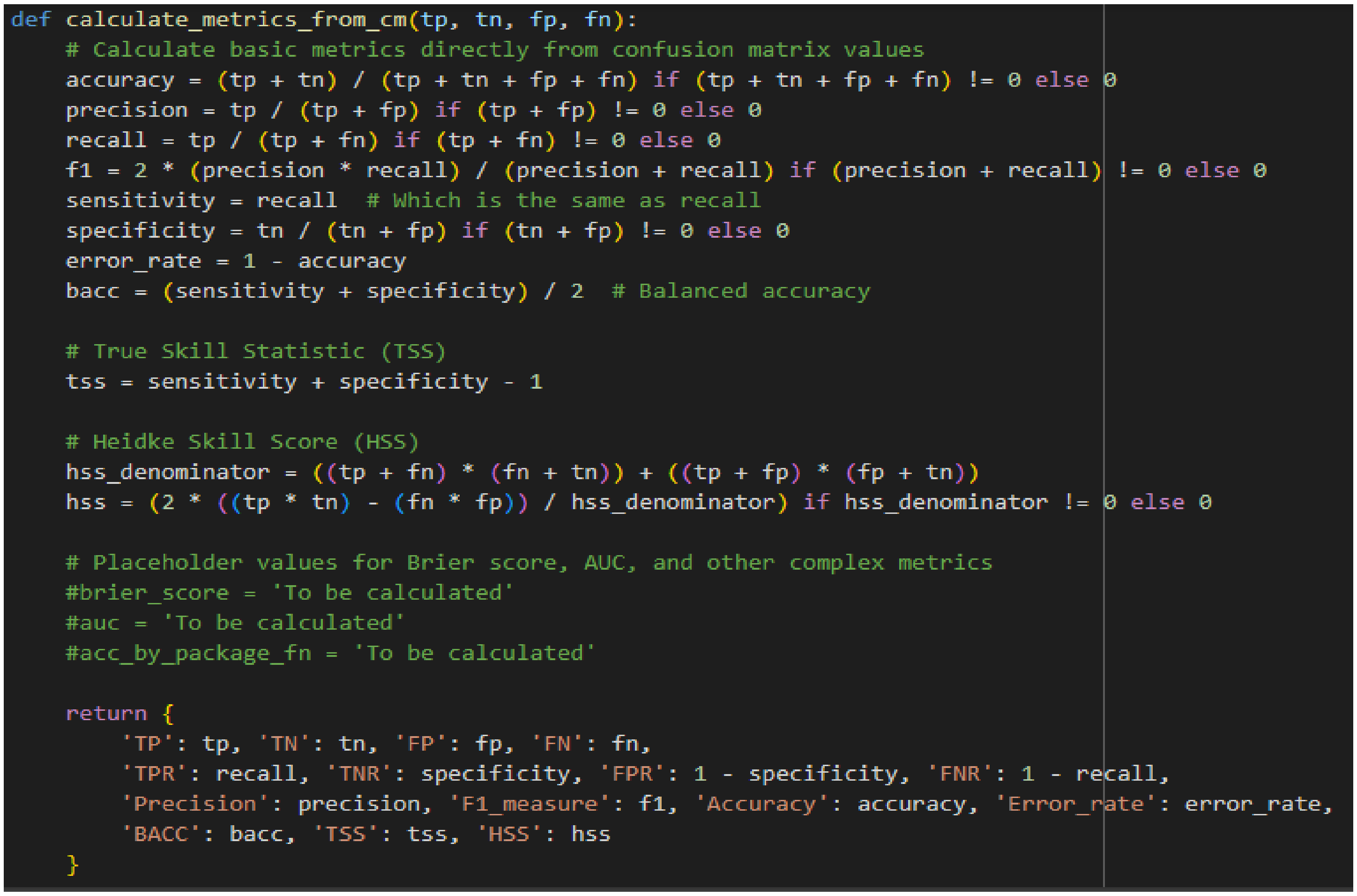
1. **Random Forest:** A reliable ensemble technique renowned for its ability to counter overfitting and achieve high accuracy in classification tasks.

2. **Deep Learning Model (e.g., LSTM):** Selected for its capability to model sequences, which proves advantageous for data where temporal patterns could impact the outcome.

3. **Support Vector Machines (SVM):** Chosen for its proficiency in handling high-dimensional spaces and its capacity to model non-linear decision boundaries using kernel functions.

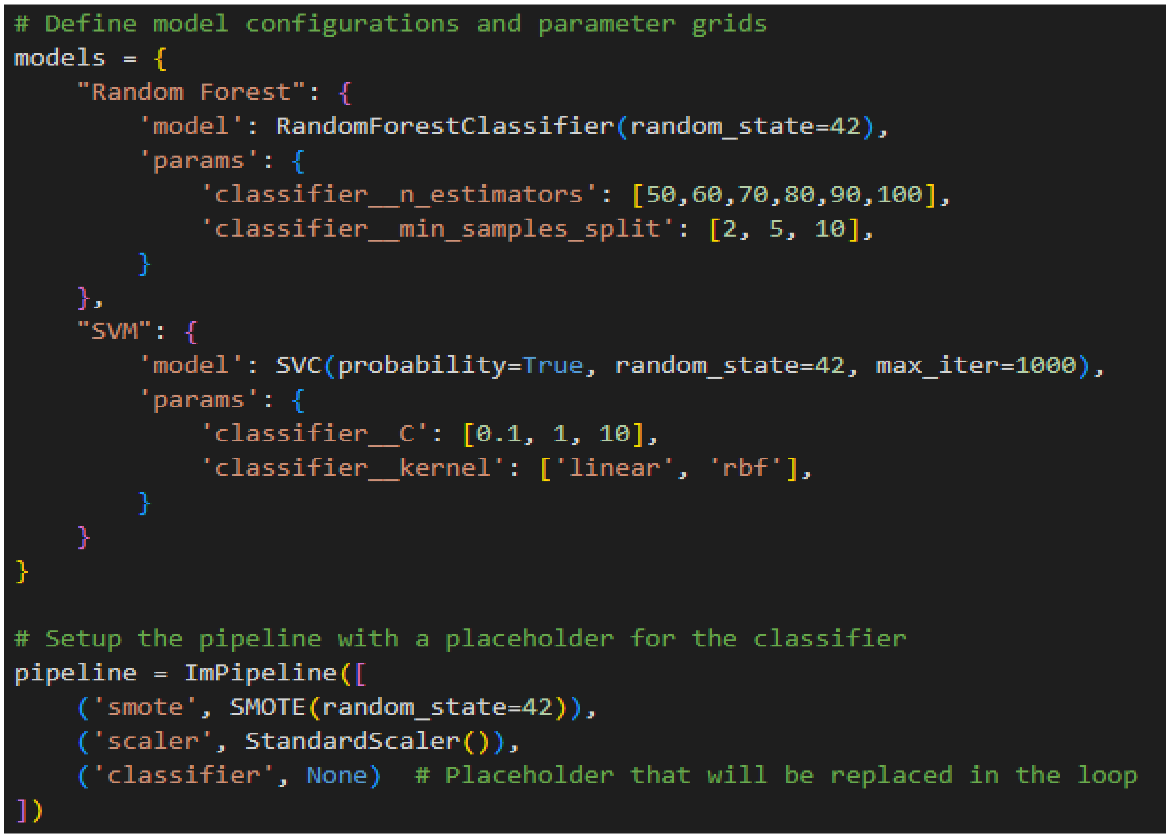
Each model was subjected to 10-fold cross-validation to ensure the robustness of the findings and to mitigate the model's performance variability.

**Function for calculation of metrics using confusion matrix.**

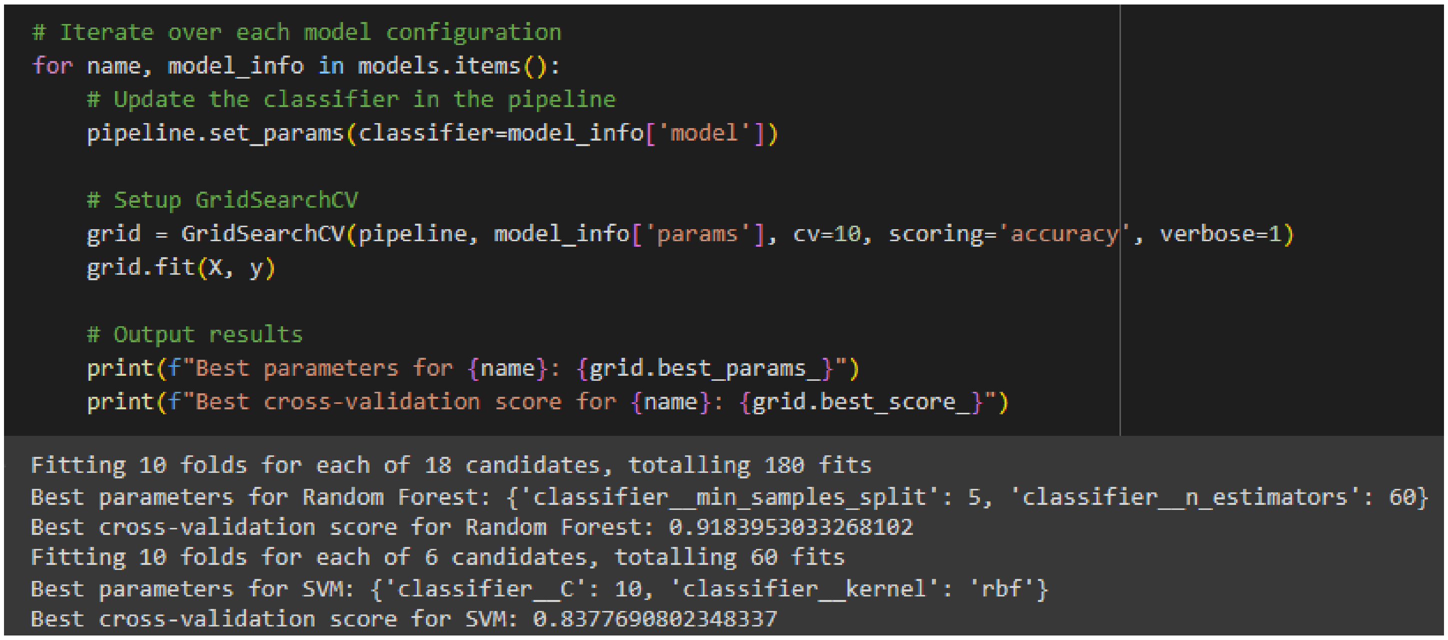


```python
def calculate_metrics_from_cm(tp, tn, fp, fn):
    # Calculate basic metrics directly from confusion matrix values
    accuracy = (tp + tn) / (tp + tn + fp + fn) if (tp + tn + fp + fn) != 0 else 0
    precision = tp / (tp + fp) if (tp + fp) != 0 else 0
    recall = tp / (tp + fn) if (tp + fn) != 0 else 0
    f1 = 2 * (precision * recall) / (precision + recall) if (precision + recall) != 0 else 0
    sensitivity = recall  # Which is the same as recall
    specificity = tn / (tn + fp) if (tn + fp) != 0 else 0
    error_rate = 1 - accuracy
    bacc = (sensitivity + specificity) / 2  # Balanced accuracy

    # True Skill Statistic (TSS)
    tss = sensitivity + specificity - 1

    # Heidke Skill Score (HSS)
    hss_denominator = ((tp + fn) * (fn + tn)) + ((tp + fp) * (fp + tn))
    hss = (2 * ((tp * tn) - (fn * fp)) / hss_denominator) if hss_denominator != 0 else 0

    # Placeholder values for Brier score, AUC, and other complex metrics
    #brier_score = 'To be calculated'
    #auc = 'To be calculated'
    #acc_by_package_fn = 'To be calculated'

    return {
        'TP': tp, 'TN': tn, 'FP': fp, 'FN': fn,
        'TPR': recall, 'TNR': specificity, 'FPR': 1 - specificity, 'FNR': 1 - recall,
        'Precision': precision, 'F1_measure': f1, 'Accuracy': accuracy, 'Error_rate': error_rate,
        'BACC': bacc, 'TSS': tss, 'HSS': hss
    }
```

**Finding Best parameters for models using GridSearchCV.**



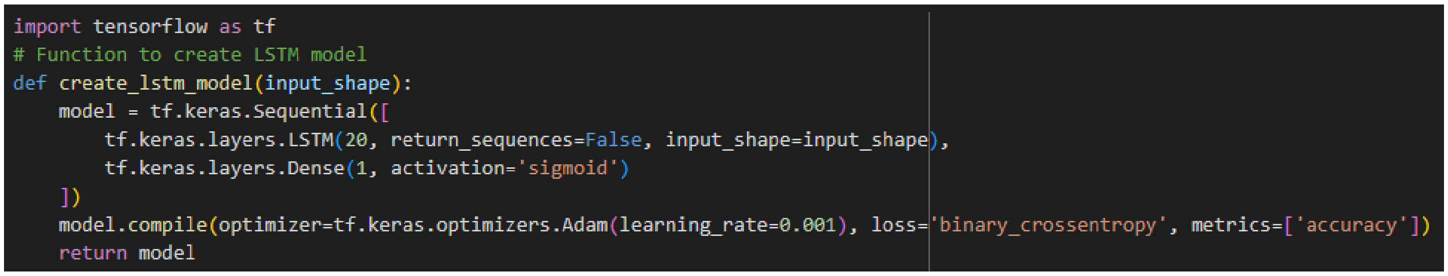
```python
# Define model configurations and parameter grids
models = {
    "Random Forest": {
        'model': RandomForestClassifier(random_state=42),
        'params': {
            'classifier__n_estimators': [50,60,70,80,90,100],
            'classifier__min_samples_split': [2, 5, 10],
        }
    },
    "SVM": {
        'model': SVC(probability=True, random_state=42, max_iter=1000),
        'params': {
            'classifier__C': [0.1, 1, 10],
            'classifier__kernel': ['linear', 'rbf'],
        }
    }
}

# Setup the pipeline with a placeholder for the classifier
pipeline = ImPipeline([
    ('smote', SMOTE(random_state=42)),
    ('scaler', StandardScaler()),
    ('classifier', None)  # Placeholder that will be replaced in the loop
])
```



```python
# Iterate over each model configuration
for name, model_info in models.items():
    # Update the classifier in the pipeline
    pipeline.set_params(classifier=model_info['model'])

    # Setup GridSearchCV
    grid = GridSearchCV(pipeline, model_info['params'], cv=10, scoring='accuracy', verbose=1)
    grid.fit(X, y)

    # Output results
    print(f"Best parameters for {name}: {grid.best_params_}")
    print(f"Best cross-validation score for {name}: {grid.best_score_}")
```

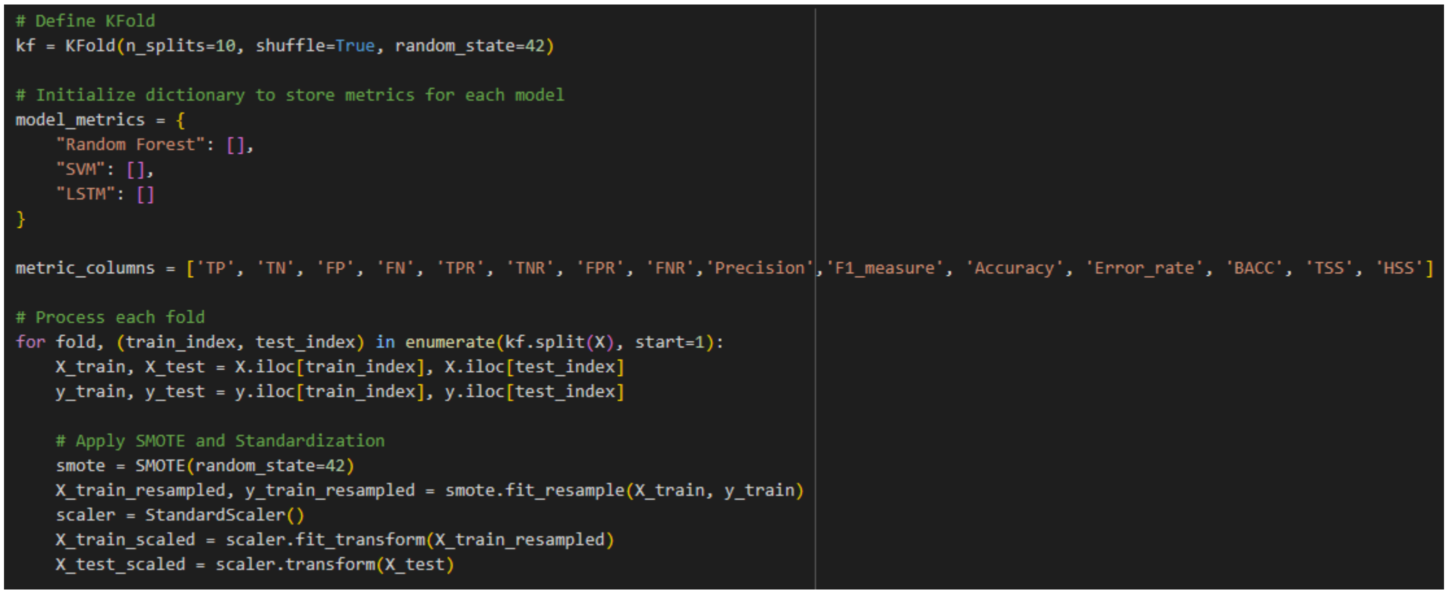
```
Fitting 10 folds for each of 18 candidates, totalling 180 fits
Best parameters for Random Forest: {'classifier__min_samples_split': 5, 'classifier__n_estimators': 60}
Best cross-validation score for Random Forest: 0.9183953033268102
Fitting 10 folds for each of 6 candidates, totalling 60 fits
Best parameters for SVM: {'classifier__C': 10, 'classifier__kernel': 'rbf'}
Best cross-validation score for SVM: 0.8377690802348337
```

## Function to create LSTM model.

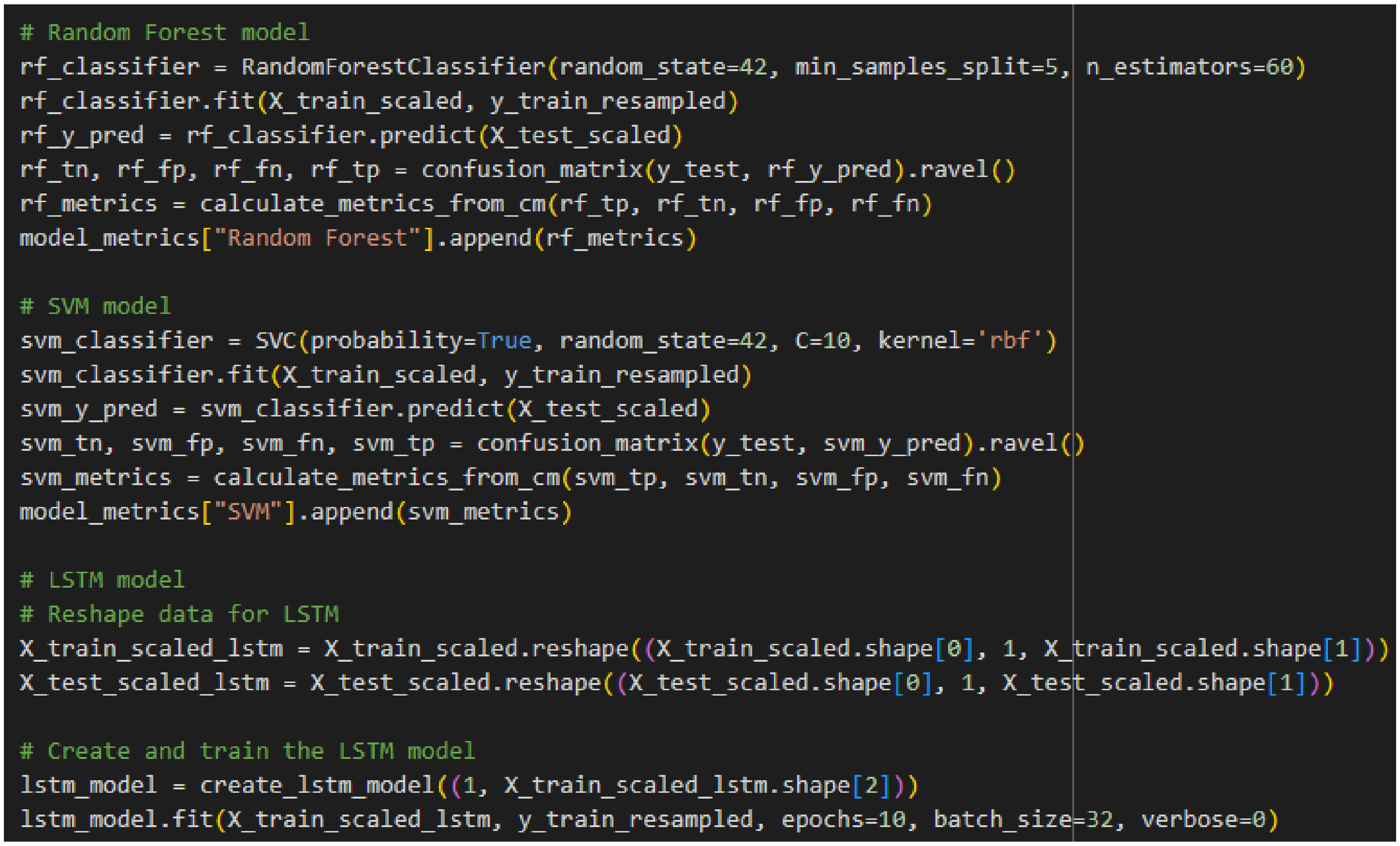


```python
import tensorflow as tf
# Function to create LSTM model
def create_lstm_model(input_shape):
    model = tf.keras.Sequential([
        tf.keras.layers.LSTM(20, return_sequences=False, input_shape=input_shape),
        tf.keras.layers.Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

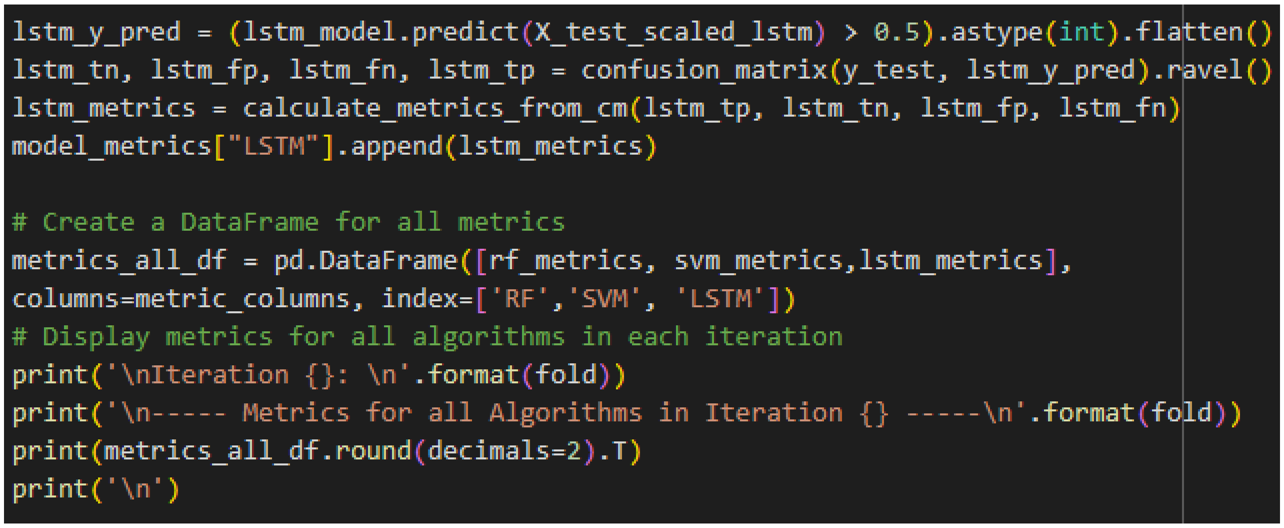
## Model Training with Kfold.



```python
# Define KFold
kf = KFold(n_splits=10, shuffle=True, random_state=42)

# Initialize dictionary to store metrics for each model
model_metrics = {
    "Random Forest": [],
    "SVM": [],
    "LSTM": []
}

metric_columns = ['TP', 'TN', 'FP', 'FN', 'TPR', 'TNR', 'FPR', 'FNR','Precision','F1_measure', 'Accuracy', 'Error_rate', 'BACC', 'TSS', 'HSS']

# Process each fold
for fold, (train_index, test_index) in enumerate(kf.split(X), start=1):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Apply SMOTE and Standardization
    smote = SMOTE(random_state=42)
    X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train_resampled)
    X_test_scaled = scaler.transform(X_test)
```



```python
# Random Forest model
rf_classifier = RandomForestClassifier(random_state=42, min_samples_split=5, n_estimators=60)
rf_classifier.fit(X_train_scaled, y_train_resampled)
rf_y_pred = rf_classifier.predict(X_test_scaled)
rf_tn, rf_fp, rf_fn, rf_tp = confusion_matrix(y_test, rf_y_pred).ravel()
rf_metrics = calculate_metrics_from_cm(rf_tp, rf_tn, rf_fp, rf_fn)
model_metrics["Random Forest"].append(rf_metrics)

# SVM model
svm_classifier = SVC(probability=True, random_state=42, C=10, kernel='rbf')
svm_classifier.fit(X_train_scaled, y_train_resampled)
svm_y_pred = svm_classifier.predict(X_test_scaled)
svm_tn, svm_fp, svm_fn, svm_tp = confusion_matrix(y_test, svm_y_pred).ravel()
svm_metrics = calculate_metrics_from_cm(svm_tp, svm_tn, svm_fp, svm_fn)
model_metrics["SVM"].append(svm_metrics)

# LSTM model
# Reshape data for LSTM
X_train_scaled_lstm = X_train_scaled.reshape((X_train_scaled.shape[0], 1, X_train_scaled.shape[1]))
X_test_scaled_lstm = X_test_scaled.reshape((X_test_scaled.shape[0], 1, X_test_scaled.shape[1]))

# Create and train the LSTM model
lstm_model = create_lstm_model((1, X_train_scaled_lstm.shape[2]))
lstm_model.fit(X_train_scaled_lstm, y_train_resampled, epochs=10, batch_size=32, verbose=0)
```



```python
lstm_y_pred = (lstm_model.predict(X_test_scaled_lstm) > 0.5).astype(int).flatten()
lstm_tn, lstm_fp, lstm_fn, lstm_tp = confusion_matrix(y_test, lstm_y_pred).ravel()
lstm_metrics = calculate_metrics_from_cm(lstm_tp, lstm_tn, lstm_fp, lstm_fn)
model_metrics["LSTM"].append(lstm_metrics)

# Create a DataFrame for all metrics
metrics_all_df = pd.DataFrame([rf_metrics, svm_metrics,lstm_metrics],
columns=metric_columns, index=['RF','SVM', 'LSTM'])
# Display metrics for all algorithms in each iteration
print('\nIteration {}: \n'.format(fold))
print('\n----- Metrics for all Algorithms in Iteration {} -----\n'.format(fold))
print(metrics_all_df.round(decimals=2).T)
print('\n')
```

## Model Evaluation.

The evaluation of the models involved utilizing metrics derived from the confusion matrix, such as True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN). Additional metrics, including the False Positive Rate (FPR), False Negative Rate (FNR), True Skill Statistics (TSS), and Heidke Skill Score (HSS), were also computed manually. This approach was undertaken to enhance comprehension and ensure accurate assessment of the models.

```
Iteration 1:


----- Metrics for all Algorithms in Iteration 1 -----

                RF     SVM    LSTM
TP            2.00    5.00    9.00
TN          467.00  430.00  431.00
FP           16.00   53.00   52.00
FN           26.00   23.00   19.00
TPR           0.07    0.18    0.32
TNR           0.97    0.89    0.89
FPR           0.03    0.11    0.11
FNR           0.93    0.82    0.68
Precision     0.11    0.09    0.15
F1_measure    0.09    0.12    0.20
Accuracy      0.92    0.85    0.86
Error_rate    0.08    0.15    0.14
BACC          0.52    0.53    0.61
TSS           0.04    0.07    0.21
HSS           0.05    0.05    0.14


16/16 [==============================] - 0s 2ms/step

Iteration 2:


----- Metrics for all Algorithms in Iteration 2 -----

                RF     SVM    LSTM
TP            7.00    8.00    7.00
TN          454.00  432.00  435.00
FP           23.00   45.00   42.00
FN           27.00   26.00   27.00
TPR           0.21    0.24    0.21
TNR           0.95    0.91    0.91
FPR           0.05    0.09    0.09
FNR           0.79    0.76    0.79
Precision     0.23    0.15    0.14
F1_measure    0.22    0.18    0.17
Accuracy      0.90    0.86    0.86
Error_rate    0.10    0.14    0.14
BACC          0.58    0.57    0.56
TSS           0.16    0.14    0.12
HSS           0.17    0.11    0.10


16/16 [==============================] - 1s 3ms/step

Iteration 3:


----- Metrics for all Algorithms in Iteration 3 -----

                RF     SVM    LSTM
TP            4.00   11.00    9.00
TN          467.00  440.00  439.00
FP           17.00   44.00   45.00
```

```
FN           23.00   16.00   18.00
TPR           0.15    0.41    0.33
TNR           0.96    0.91    0.91
FPR           0.04    0.09    0.09
FNR           0.85    0.59    0.67
Precision     0.19    0.20    0.17
F1_measure    0.17    0.27    0.22
Accuracy      0.92    0.88    0.88
Error_rate    0.08    0.12    0.12
BACC          0.56    0.66    0.62
TSS           0.11    0.32    0.24
HSS           0.13    0.21    0.16


16/16 [==============================] - 1s 2ms/step

Iteration 4:


----- Metrics for all Algorithms in Iteration 4 -----

                RF     SVM    LSTM
TP            2.00    4.00    7.00
TN          464.00  434.00  432.00
FP           22.00   52.00   54.00
FN           23.00   21.00   18.00
TPR           0.08    0.16    0.28
TNR           0.95    0.89    0.89
FPR           0.05    0.11    0.11
FNR           0.92    0.84    0.72
Precision     0.08    0.07    0.11
F1_measure    0.08    0.10    0.16
Accuracy      0.91    0.86    0.86
Error_rate    0.09    0.14    0.14
BACC          0.52    0.53    0.58
TSS           0.03    0.05    0.17
HSS           0.04    0.03    0.10


16/16 [==============================] - 1s 3ms/step

Iteration 5:


----- Metrics for all Algorithms in Iteration 5 -----

                RF     SVM    LSTM
TP            2.00    4.00    5.00
TN          480.00  434.00  443.00
FP           16.00   62.00   53.00
FN           13.00   11.00   10.00
TPR           0.13    0.27    0.33
TNR           0.97    0.88    0.89
FPR           0.03    0.12    0.11
FNR           0.87    0.73    0.67
Precision     0.11    0.06    0.09
F1_measure    0.12    0.10    0.14
Accuracy      0.94    0.86    0.88
Error_rate    0.06    0.14    0.12
```

```
BACC           0.55    0.57    0.61
TSS            0.10    0.14    0.23
HSS            0.09    0.05    0.09


16/16 [==============================] - 1s 2ms/step

Iteration 6:


----- Metrics for all Algorithms in Iteration 6 -----

                 RF     SVM    LSTM
TP             2.00    3.00    5.00
TN           465.00  436.00  440.00
FP            22.00   51.00   47.00
FN            22.00   21.00   19.00
TPR            0.08    0.12    0.21
TNR            0.95    0.90    0.90
FPR            0.05    0.10    0.10
FNR            0.92    0.88    0.79
Precision      0.08    0.06    0.10
F1_measure     0.08    0.08    0.13
Accuracy       0.91    0.86    0.87
Error_rate     0.09    0.14    0.13
BACC           0.52    0.51    0.56
TSS            0.04    0.02    0.11
HSS            0.04    0.01    0.07


16/16 [==============================] - 1s 3ms/step

Iteration 7:


----- Metrics for all Algorithms in Iteration 7 -----

                 RF     SVM    LSTM
TP             3.00    6.00    8.00
TN           475.00  433.00  422.00
FP            16.00   58.00   69.00
FN            17.00   14.00   12.00
TPR            0.15    0.30    0.40
TNR            0.97    0.88    0.86
FPR            0.03    0.12    0.14
FNR            0.85    0.70    0.60
Precision      0.16    0.09    0.10
F1_measure     0.15    0.14    0.16
Accuracy       0.94    0.86    0.84
Error_rate     0.06    0.14    0.16
BACC           0.56    0.59    0.63
TSS            0.12    0.18    0.26
HSS            0.12    0.09    0.11


16/16 [==============================] - 1s 2ms/step

Iteration 8:
```

```
----- Metrics for all Algorithms in Iteration 8 -----

                RF     SVM    LSTM
TP            7.00    9.00   10.00
TN          461.00  430.00  426.00
FP           17.00   48.00   52.00
FN           26.00   24.00   23.00
TPR           0.21    0.27    0.30
TNR           0.96    0.90    0.89
FPR           0.04    0.10    0.11
FNR           0.79    0.73    0.70
Precision     0.29    0.16    0.16
F1_measure    0.25    0.20    0.21
Accuracy      0.92    0.86    0.85
Error_rate    0.08    0.14    0.15
BACC          0.59    0.59    0.60
TSS           0.18    0.17    0.19
HSS           0.20    0.13    0.14


16/16 [==============================] - 1s 2ms/step

Iteration 9:


----- Metrics for all Algorithms in Iteration 9 -----

                RF     SVM    LSTM
TP            3.00    4.00    6.00
TN          473.00  441.00  430.00
FP           18.00   50.00   61.00
FN           17.00   16.00   14.00
TPR           0.15    0.20    0.30
TNR           0.96    0.90    0.88
FPR           0.04    0.10    0.12
FNR           0.85    0.80    0.70
Precision     0.14    0.07    0.09
F1_measure    0.15    0.11    0.14
Accuracy      0.93    0.87    0.85
Error_rate    0.07    0.13    0.15
BACC          0.56    0.55    0.59
TSS           0.11    0.10    0.18
HSS           0.11    0.05    0.08


16/16 [==============================] - 0s 2ms/step

Iteration 10:


----- Metrics for all Algorithms in Iteration 10 -----

                RF     SVM    LSTM
TP            3.00    5.00    6.00
TN          470.00  439.00  439.00
FP           18.00   49.00   49.00
FN           20.00   18.00   17.00
TPR           0.13    0.22    0.26
```

```
TNR            0.96    0.90    0.90
FPR            0.04    0.10    0.10
FNR            0.87    0.78    0.74
Precision      0.14    0.09    0.11
F1_measure     0.14    0.13    0.15
Accuracy       0.93    0.87    0.87
Error_rate     0.07    0.13    0.13
BACC           0.55    0.56    0.58
TSS            0.09    0.12    0.16
HSS            0.10    0.07    0.10
```

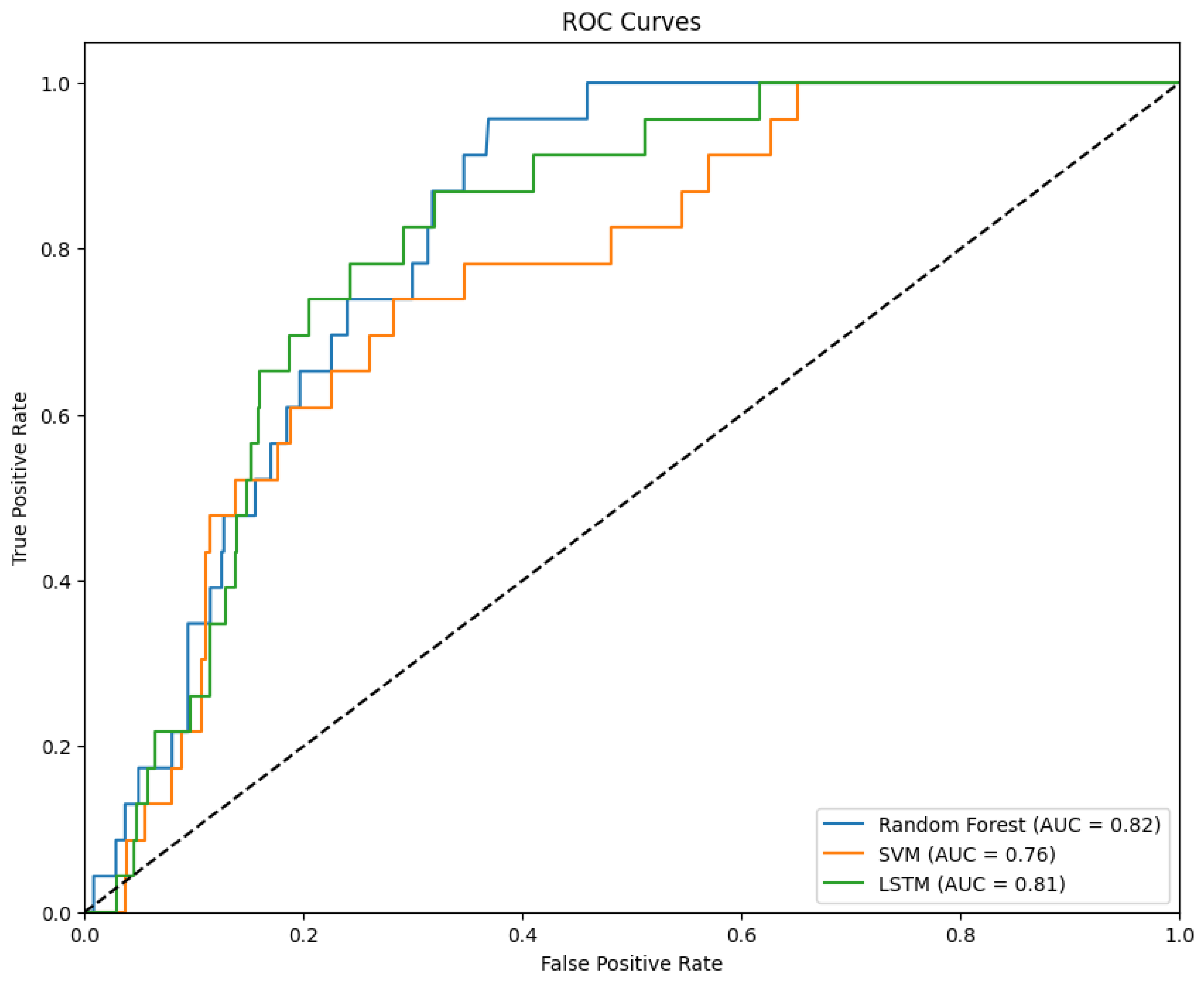
## Calculating average metrics for each model after all folds

```
# Calculate average metrics for each model after all folds
average_metrics = {}
for model_name, metrics_list in model_metrics.items():
    average_metrics[model_name] = pd.DataFrame(metrics_list).mean()

# Create a DataFrame from the average metrics dictionary
average_metrics_df = pd.DataFrame(average_metrics)

# Transpose DataFrame to display models as rows and metrics as columns
average_metrics_df = average_metrics_df.T

# Print the formatted DataFrame
print("Average Metrics per Model across All Folds:")
print(average_metrics_df.round(2))
```

```
Average Metrics per Model across All Folds:
                TP     TN    FP    FN   TPR   TNR   FPR   FNR  Precision  \
Random Forest  3.5  467.6  18.5  21.4  0.14  0.96  0.04  0.86       0.15   
SVM            5.9  434.9  51.2  19.0  0.24  0.89  0.11  0.76       0.10   
LSTM           7.2  433.7  52.4  17.7  0.29  0.89  0.11  0.71       0.12   

               F1_measure  Accuracy  Error_rate  BACC   TSS   HSS  
Random Forest        0.14      0.92        0.08  0.55  0.10  0.10  
SVM                  0.14      0.86        0.14  0.57  0.13  0.08  
LSTM                 0.17      0.86        0.14  0.59  0.19  0.11  
```

## Computing ROC curve and ROC area for each model.

```
# Predict probabilities
rf_y_score = rf_classifier.predict_proba(X_test_scaled)[:, 1]
svm_y_score = svm_classifier.predict_proba(X_test_scaled)[:, 1]
lstm_y_score = lstm_model.predict(X_test_scaled_lstm).ravel()

# Compute ROC curve and ROC area for each model
fpr_rf, tpr_rf, _ = roc_curve(y_test, rf_y_score)
auc_rf = auc(fpr_rf, tpr_rf)

fpr_svm, tpr_svm, _ = roc_curve(y_test, svm_y_score)
auc_svm = auc(fpr_svm, tpr_svm)

fpr_lstm, tpr_lstm, _ = roc_curve(y_test, lstm_y_score)
auc_lstm = auc(fpr_lstm, tpr_lstm)

# Plot all ROC curves
plt.figure(figsize=(10, 8))
plt.plot(fpr_rf, tpr_rf, label='Random Forest (AUC = %0.2f)' % auc_rf)
plt.plot(fpr_svm, tpr_svm, label='SVM (AUC = %0.2f)' % auc_svm)
plt.plot(fpr_lstm, tpr_lstm, label='LSTM (AUC = %0.2f)' % auc_lstm)

plt.plot([0, 1], [0, 1], 'k--')  # Random chance line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves')
plt.legend(loc="lower right")
plt.show()
```



## Results:

The performance of three classification models—Random Forest, Support Vector Machines (SVM), and Long Short-Term Memory (LSTM) networks—was evaluated across all folds using various measures. The results demonstrate the following average metrics per model:

- **Random Forest**: Demonstrated a relatively balanced performance, achieving the highest accuracy of 92%. It attained a True Positive Rate (TPR) of 0.14 and a True Negative Rate (TNR) of 0.96, signifying a robust ability to accurately identify non-stroke cases. However, it also exhibited a relatively high False Negative Rate (FNR) of 0.86, indicating that while it seldom misclassified non-stroke cases as strokes, it more frequently failed to identify stroke cases.

- **Support Vector Machines (SVM)**: Exhibited a higher True Positive Rate (0.24) than Random Forest, indicating superior identification of stroke cases. However, this came with a notable trade-off in precision and resulted in a lower overall accuracy of 86%. Additionally, this model displayed a higher False Positive Rate (FPR) of 0.11, implying a greater number of non-stroke cases being incorrectly classified as strokes.

- **LSTM**: Attained the highest True Positive Rate (0.29) among the three models, indicating superior identification of stroke cases. However, akin to the SVM, it exhibited a high False Positive Rate (0.11) and an overall accuracy comparable to SVM at 86%.

Regarding the ROC curve, the Random Forest model attained the highest Area Under the Curve (AUC) at 0.82, closely trailed by the LSTM model at 0.81, and the SVM with an AUC of 0.76. The ROC graph demonstrates the trade-off between the True Positive Rate and the False Positive Rate for each classifier, with Random Forest exhibiting a slightly superior balance between these rates compared to the other models.

## Conclusion

The conducted study revealed that the Random Forest model outperformed the other two models in accuracy, error rate, and balanced accuracy (BACC), despite its cautious approach in predicting actual stroke events (low True Positive Rate, TPR). Its conservative nature is further highlighted by its highest True Negative Rate (TNR), indicating its proficiency in accurately identifying true negatives, a crucial aspect in medical diagnoses where false alarms can have significant consequences.

The LSTM model demonstrated potential in identifying true stroke cases (highest TPR and True Skill Statistics, TSS), suggesting that deep learning techniques could capture intricate patterns in the data. However, its precision and error rate matched those of the SVM, indicating room for improvement.

While the Random Forest exhibited the best overall performance across most metrics, the selection of the optimal model may also hinge on the clinical context. For instance, in scenarios where missing a stroke diagnosis carries critical consequences, a model with a higher TPR like LSTM might be preferred, even at the expense of a higher False Positive Rate (FPR). Conversely, for large-scale screening where the cost of false positives is substantial, the Random Forest model may be more suitable due to its higher precision and lower FPR.

The findings suggest that a combination of models or further fine-tuning could be beneficial. Future research avenues could explore model ensembles, feature engineering, and data augmentation to enhance TPR without compromising other metrics. Additionally, it's imperative to consider the clinical implications of each metric and involve healthcare professionals in model selection to ensure alignment with patient care objectives.

### Recommendations:

**Model Ensembling:** Leveraging a blend of predictions from multiple models could harness their unique strengths. For instance, an ensemble consisting of both Random Forest and LSTM models might bolster overall prediction accuracy while also maintaining a heightened true positive rate.

**Feature Engineering:** Further exploration of the dataset to create new features or refine existing ones could augment the predictive capabilities of the models. Expertise in healthcare domain could be instrumental in identifying pertinent features for incorporation.

**Cost-sensitive Learning:** In healthcare scenarios where the costs associated with false negatives and false positives differ, utilizing cost-sensitive learning methods could provide enhanced practical utility by minimizing errors that result in higher costs.

**Deep Learning Model Adjustments:** Exploring different architectures and configurations of deep learning models, such as CNNs for feature extraction or several types of RNNs, might yield better results, particularly in capturing complex patterns in the data.

**Longitudinal Data Analysis:** If temporal data is available, using methods that consider the time sequence of medical events could improve predictive performance, as strokes can be associated with long-term trends in health indicators.

**Clinical Trial:** Conducting a prospective clinical trial with the developed models could validate their effectiveness and utility in a real-world clinical setting.

**Expert Involvement:** Continuously involving medical experts in the loop could ensure that the models are aligned with clinical knowledge and practice, and that the outcomes are interpretable and actionable in a healthcare environment.

## Steps to run the Program:

Minimum requirements to run the program:

- **CPU**: Intel Core i3 or equivalent AMD
- **RAM**: 4GB (8GB recommended for smoother performance)
- **Storage**: 256GB SSD (for faster read/write speeds)
- **OS**: Windows 10, macOS, or a modern Linux distribution
- **Software**: Latest version of Python, Jupyter Notebook or Visual Studio Code

### STEP 1: Cloning the Repository

Clone the repository, follow these steps:

1. Open the terminal on your machine.
2. Change the current working directory to the location where you want the cloned directory.
3. Type the following command and press Enter:
4. `git clone`

   https://github.com/Bhumireddy2001/Sai-Teja-Reddy-Bhumireddy-_-Final-Term-Project.git
5. Navigate to the cloned repository:
6. cd stroke-prediction

### STEP 2: Create Virtual Environment.

Create a conda environment after opening the repository. It is recommended to do this project in a virtual environment to avoid conflicts with other Python packages you may have installed.

```
conda create -n myenv python -y
conda activate myenv
```

or

```
python -m venv env
source env/bin/activate
```

Note: On Windows, use `env\Scripts\activate`

### STEP 3: Running the Notebook Locally

Run the notebook locally, follow these steps:

1. Ensure you have Jupyter Notebook installed, if not, install it using pip:
2. pip install notebook.
3. Install the required libraries:
4. `pip install -r requirements.txt`
5. Start the Jupyter Notebook server:

   "Jupyter notebook"
6. The command will open a new tab in your default web browser. Navigate to the .ipynb file and open it.
7. Run the cells in the Jupyter Notebook to execute the code.

### Alternative for STEP 3: Running the Notebook on Google Colab

You can also run the notebook on Google Colab without needing to install anything on your local machine:

1. Open the Google Colab website: Google Colab
2. Sign in with your Google account if you are not already logged in.
3. Go to File > Open notebook.
4. Select the GitHub tab and enter the URL of the repository.
5. Open the .ipynb file from the list.
6. You can now run the notebook cells one by one.

Remember to save a copy of the notebook to your Google Drive if you make changes and wish to keep them.

Note: Please make sure you have installed the required packages before running the code. If any package is missing, please install it from the requirements.txt file.Conclusion

## References and Links:

**GitHub Repository link:** https://github.com/Bhumireddy2001/Sai-Teja-Reddy-Bhumireddy-_-Final-Term-Project.git

**References:**

1. https://stackoverflow.com/questions/
2. Lectures Notes