Data Mining: Final Project Report

Name: Aniket Khalate

UCID: ak3274

Email: ak3274@njit.edu

Professor: Dr. Yasser Abduallah

Topic: A Performance Comparison of Machine Learning Techniques in Diabetes Prediction

- ### Abstract

This project applies and contrasts three machine learning models Random Forest, Support Vector Machine (SVM), and Long Short Term Memory (LSTM) to predict diabetes risk using medical data. The analysis is based on standardized features such as glucose levels, blood pressure, BMI, and other key health indicators. Model performance is assessed using a wide range of evaluation metrics, cross-validation, and ROC analysis to identify which algorithm delivers the strongest predictive capability for diabetes detection.

- ### Introduction

Predicting diabetes is an important use case for machine learning within healthcare. This project centers on building and comparing several models to estimate diabetes risk using key medical indicators. The work incorporates both classic machine learning techniques Random Forest and SVM and a deep learning model, LSTM. Their effectiveness is examined through a broad set of evaluation metrics to provide a thorough performance comparison.

- **Project Overview**

  The project is divided into three main components:

  1. **Data Preprocessing and Feature Engineering**
     - Importing and preprocessing the Pima Indians Diabetes dataset.
     - Replacing missing values using median-based imputation.
     - Normalizing features with StandardScaler.
     - Splitting the data into stratified training and testing sets to preserve class balance.

  2. **Model Implementation and Optimization**

     The project employs three distinct models:
     - Random Forest Classifier.
       1. Hyperparameters optimized using GridSearchCV.
       2. Tuned parameters include n_estimators and min_samples_split.

     - Support Vector Machine (SVM)
       1. Implemented with a linear kernel.
       2. C parameter optimized via grid search.

     - LSTM Neural Network
       1. Consists of a single LSTM layer with 64 units.
       2. Followed by a dense output layer with sigmoid activation.
       3. Trained using the binary cross-entropy loss function.

  3. **Performance Analysis and Comparison**

     The models were assessed using multiple evaluation metrics under 10-fold cross-validation. The main performance highlights are as follows:
     - Random Forest

       a) Delivered high accuracy with well-balanced results.

       b) Achieved strong ROC-AUC values.

c) Showed efficient computational performance.

- Support Vector Machine (SVM)
  a) Produced competitive accuracy levels.
  b) Demonstrated solid generalization on unseen data.
  c) Incurred moderate computational cost.

- LSTM
  a) Achieved accuracy comparable to traditional algorithms.
  b) Required more computational resources.
  c) Shows potential for improved results on larger datasets.

- ROC Curve Analysis
  a) All models exhibited strong discriminative capabilities.
  b) ROC curves displayed clear separation from the random baseline.
  c) High AUC scores confirmed consistent and reliable model performance.

- **Core Concepts and Principles**
  1. Feature Standardization: All input features are standardized to promote consistent scaling and ensure fair model comparison.
  2. Cross-Validation: A 10 fold stratified cross-validation approach is used to provide a reliable and unbiased performance assessment.
  3. Hyperparameter Optimization: Grid search is applied to Random Forest and SVM models to identify the most effective parameter configurations.
  4. Performance Metrics: Model evaluation is carried out using a comprehensive set of metrics, including:
     - Accuracy, Precision, and F1-score
     - ROC-AUC score
     - True Skill Statistic (TSS)
     - Heidke Skill Score (HSS)
     - Balanced Accuracy (BACC)

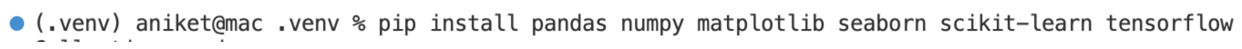
- **Technical Implementation Details**
  1. Data Processing Pipeline
     - Imputation of missing values.
     - Standardization of features.
     - Splitting the dataset into training and testing sets.

  2. Model Training Framework
     - Application of cross-validation for reliable model assessment.
     - Hyperparameter tuning for optimal performance.
     - Calculation of key performance metrics.

  3. Evaluation System
     - Detailed computation of evaluation metrics.
     - Generation of ROC curves.
     - Statistical analysis and comparison of model results.

- **Tutorial to run the .py file in your device**.
  1. Prerequisites
     - Ensure that the dataset CSV file is placed in the same directory as the .py file.
     - Maintain the original file names as provided in the ZIP folder or GitHub repository.
     - The dataset contains 400 records, so processing may take some time.

  2. Set up the Environments.
     - Ensure you have installed Python on your system. It is recommended to have ‘*Python 3.12.8*’ installed on the system.
     - If python is not installed, then go to python.org and install python version ‘*Python 3.12.8*’. It is important to select the "Add Python to PATH" option during the installation process.
     - Install the required Libraries by running:



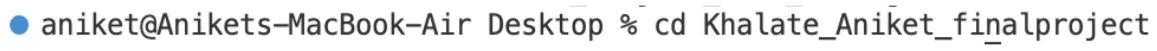
```
(.venv) aniket@mac .venv % pip install pandas numpy matplotlib seaborn scikit-learn tensorflow
```

3. Prepare the data files.
   Make sure you have the following CSV file in the same directory and with same naming convention as the script, “pima_diabetes”.

   Note: The Dataset is taken from Pima Indian Healthcare system.

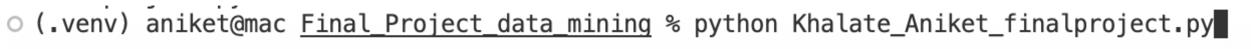
4. Run the program.
   - Check if the Python file is saved as, Khalate_Aniket_finalproject.py
   - Open a terminal or command prompt.
   - Navigate to the directory of folder containing the Python file and CSV files. The command to change the directory is: “cd *<path_of_folder_of_python_file>*”

     eg)

     ```
     aniket@Anikets-MacBook-Air Desktop % cd Khalate_Aniket_finalproject
     ```

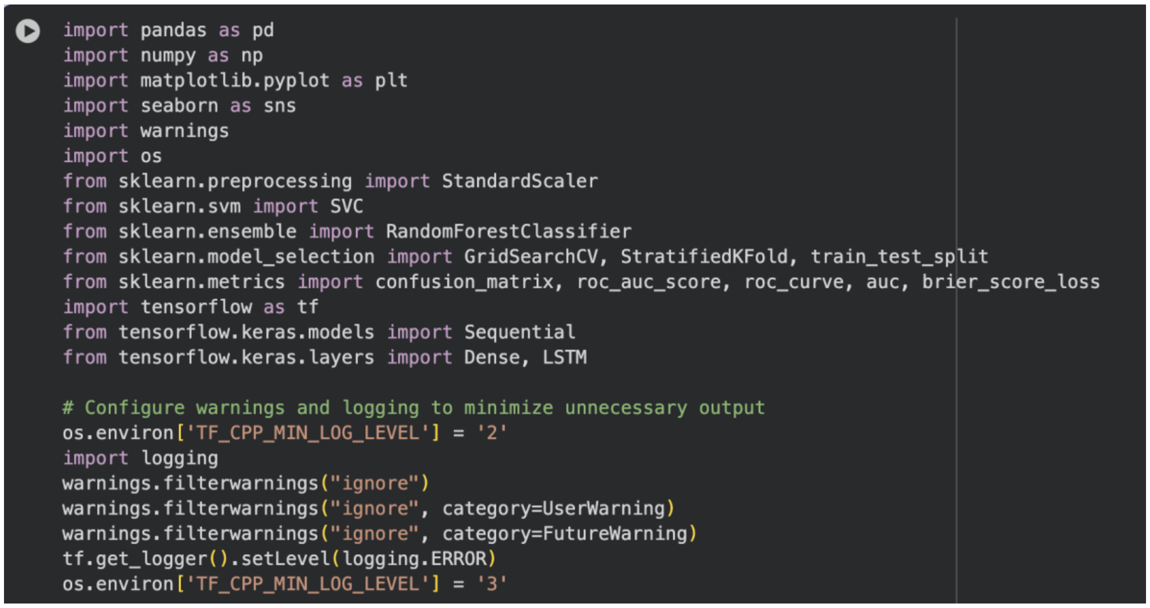
   - Run the python script using following command:
     “python *<python_file_name.py>*”

     ```
     (.venv) aniket@mac Final_Project_data_mining % python Khalate_Aniket_finalproject.py
     ```

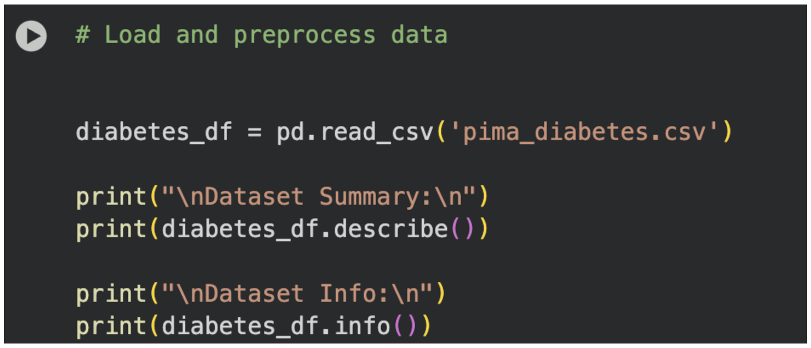
- **Code Implementation**
  1. Import statements
     - Basic Data Processing and Analysis: pandas, numpy.
     - Visualization Libraries: matplotlib, seaborn.
     - Warning Suppression
     - Scikit-learn Components: StandardScaler, SVC, RnadomForestClassifier, GridSearchCV, StratifiedKFold, train_test_split, confusion_matrix, roc_auc_score, roc_curve, auc, brier_score_loss.

- TensorFlow and Environment Settings: Sequential, Dense, LSTM.

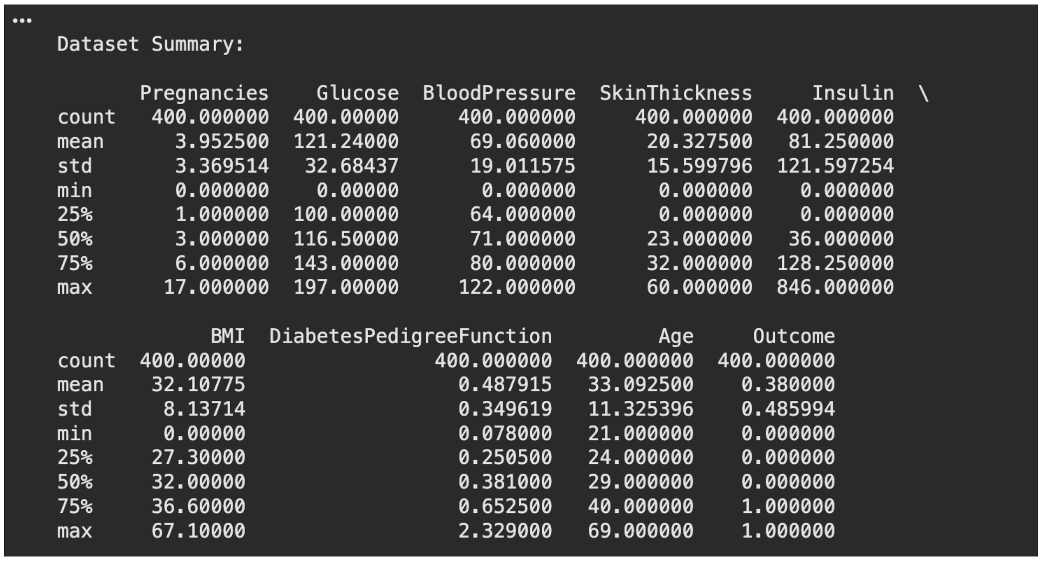


```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import os
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, StratifiedKFold, train_test_split
from sklearn.metrics import confusion_matrix, roc_auc_score, roc_curve, auc, brier_score_loss
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM

# Configure warnings and logging to minimize unnecessary output
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import logging
warnings.filterwarnings("ignore")
warnings.filterwarnings("ignore", category=UserWarning)
warnings.filterwarnings("ignore", category=FutureWarning)
tf.get_logger().setLevel(logging.ERROR)
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
```

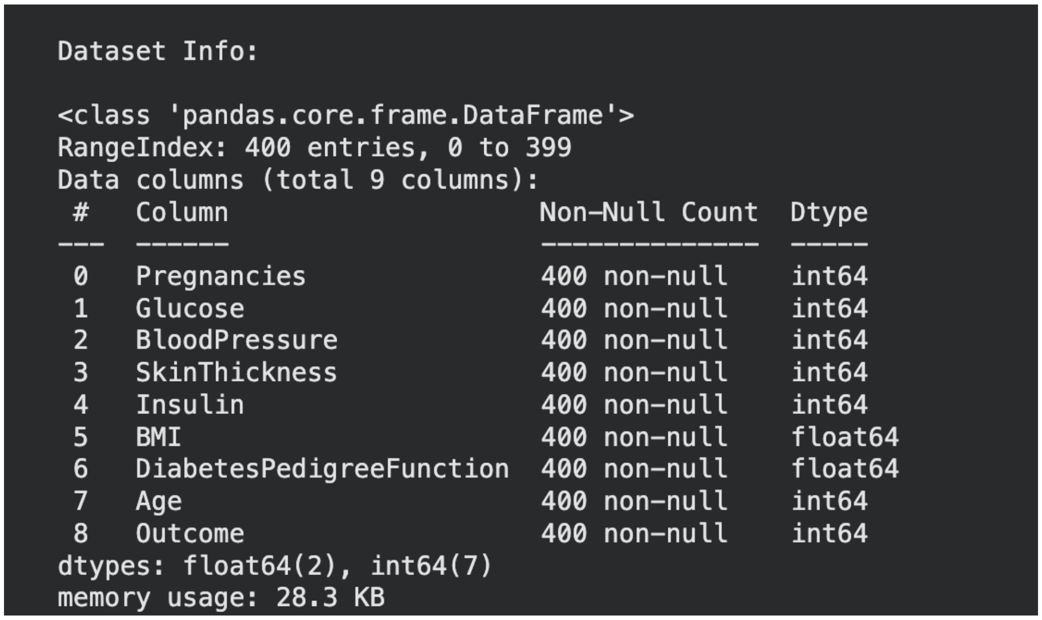
2. Data Loading and Preprocessing



```
# Load and preprocess data


diabetes_df = pd.read_csv('pima_diabetes.csv')

print("\nDataset Summary:\n")
print(diabetes_df.describe())

print("\nDataset Info:\n")
print(diabetes_df.info())
```

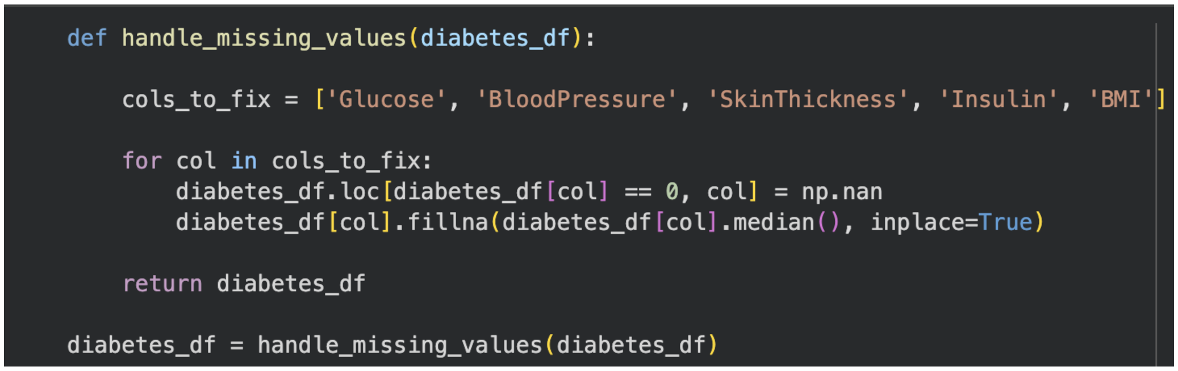


```
...
    Dataset Summary:

           Pregnancies    Glucose  BloodPressure  SkinThickness     Insulin  \
    count   400.000000  400.00000     400.000000     400.000000  400.000000
    mean      3.952500  121.24000      69.060000      20.327500   81.250000
    std       3.369514   32.68437      19.011575      15.599796  121.597254
    min       0.000000    0.00000       0.000000       0.000000    0.000000
    25%       1.000000  100.00000      64.000000       0.000000    0.000000
    50%       3.000000  116.50000      71.000000      23.000000   36.000000
    75%       6.000000  143.00000      80.000000      32.000000  128.250000
    max      17.000000  197.00000     122.000000      60.000000  846.000000

                 BMI  DiabetesPedigreeFunction         Age     Outcome
    count  400.00000                400.000000  400.000000  400.000000
    mean    32.10775                  0.487915   33.092500    0.380000
    std      8.13714                  0.349619   11.325396    0.485994
    min      0.00000                  0.078000   21.000000    0.000000
    25%     27.30000                  0.250500   24.000000    0.000000
    50%     32.00000                  0.381000   29.000000    0.000000
    75%     36.60000                  0.652500   40.000000    1.000000
    max     67.10000                  2.329000   69.000000    1.000000
```



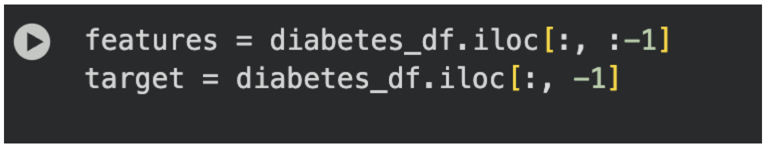
```
Dataset Info:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               400 non-null    int64  
 1   Glucose                   400 non-null    int64  
 2   BloodPressure             400 non-null    int64  
 3   SkinThickness             400 non-null    int64  
 4   Insulin                   400 non-null    int64  
 5   BMI                       400 non-null    float64
 6   DiabetesPedigreeFunction  400 non-null    float64
 7   Age                       400 non-null    int64  
 8   Outcome                   400 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 28.3 KB
```

3. Data Imputation: The below function imputes missing data in the diabetes dataset by treating incorrectly recorded zeros. Since measurements like glucose or blood pressure cannot realistically be zero for a living patient, those zeros are identified as missing and replaced with more reasonable estimates.



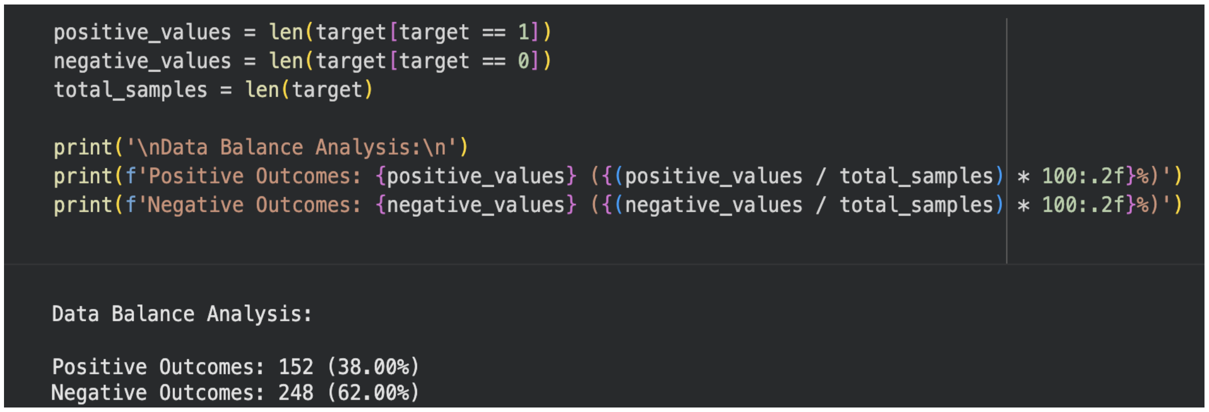
```
def handle_missing_values(diabetes_df):

    cols_to_fix = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']

    for col in cols_to_fix:
        diabetes_df.loc[diabetes_df[col] == 0, col] = np.nan
        diabetes_df[col].fillna(diabetes_df[col].median(), inplace=True)

    return diabetes_df

diabetes_df = handle_missing_values(diabetes_df)
```

4. Feature and Label Splitting: Divide the diabetes dataset into two parts
   - Features (X): The input variables used for prediction.
   - Target (y): The outcome the model is trying to predict.



```
features = diabetes_df.iloc[:, :-1]
target = diabetes_df.iloc[:, -1]
```

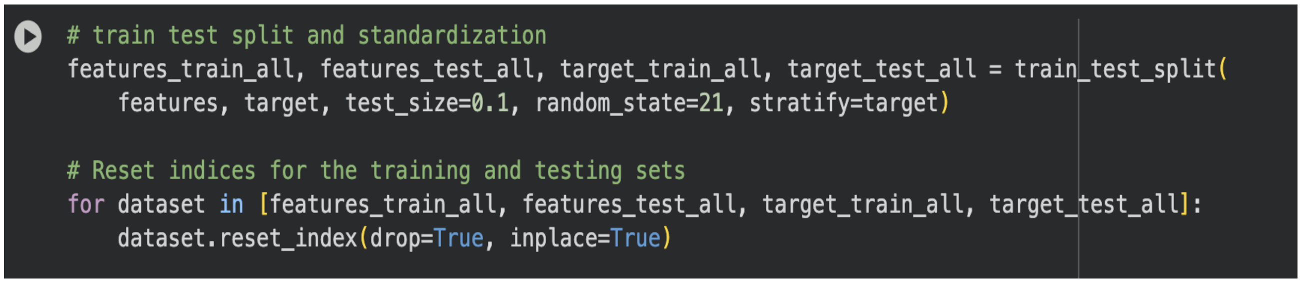
5. Data Balance Analysis: The below code checks how the diabetes dataset is distributed by:
   - Counting how many patients have diabetes (positive values)
   - Counting how many do not have diabetes (negative values)
   - Computing the percentage share of each group



```
positive_values = len(target[target == 1])
negative_values = len(target[target == 0])
total_samples = len(target)

print('\nData Balance Analysis:\n')
print(f'Positive Outcomes: {positive_values} ({(positive_values / total_samples) * 100:.2f}%)')
print(f'Negative Outcomes: {negative_values} ({(negative_values / total_samples) * 100:.2f}%)')
```

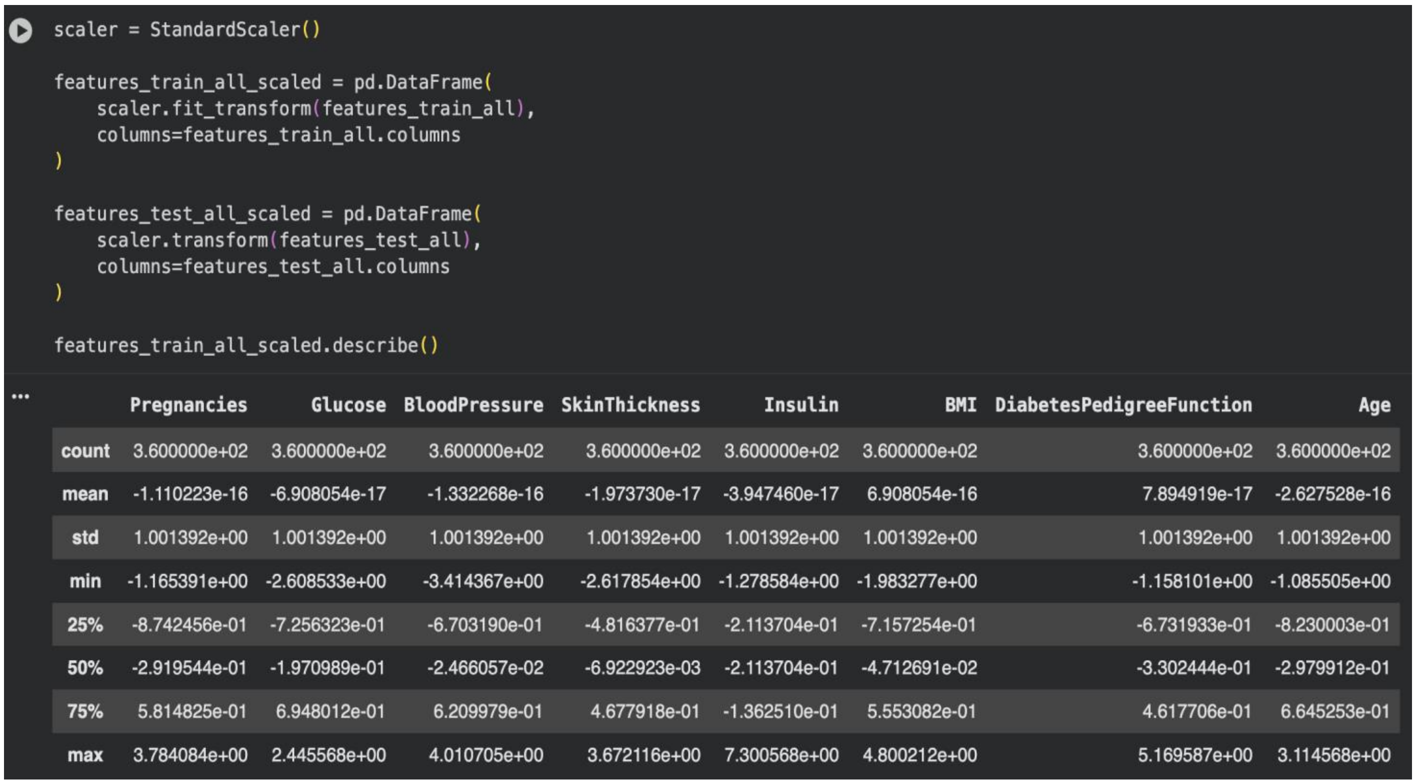
```
Data Balance Analysis:

Positive Outcomes: 152 (38.00%)
Negative Outcomes: 248 (62.00%)
```

6. Train Test Split: The code splits the dataset into training and testing set while also maintaining the distribution of the target variable.



```
# train test split and standardization
features_train_all, features_test_all, target_train_all, target_test_all = train_test_split(
    features, target, test_size=0.1, random_state=21, stratify=target)

# Reset indices for the training and testing sets
for dataset in [features_train_all, features_test_all, target_train_all, target_test_all]:
    dataset.reset_index(drop=True, inplace=True)
```

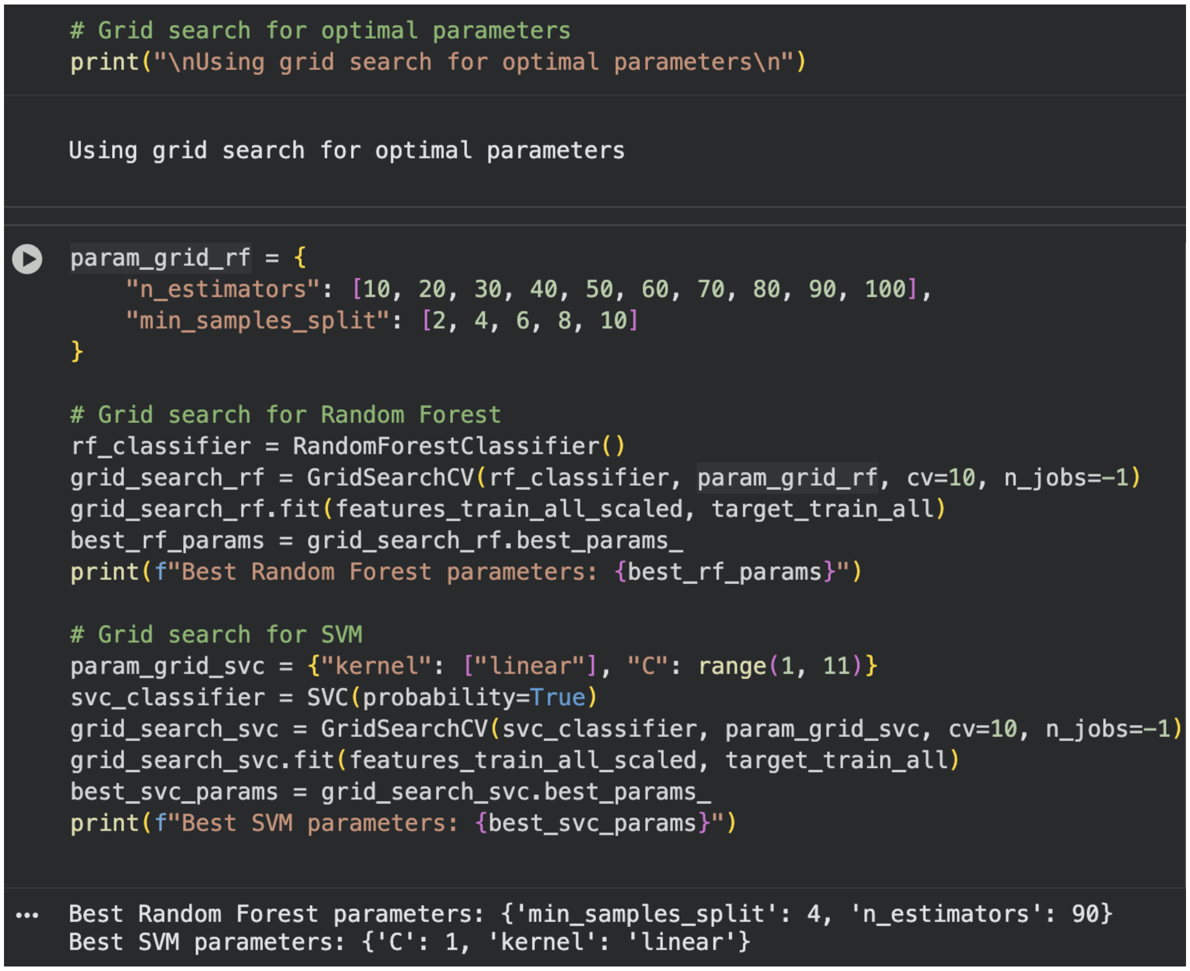
7. Feature Standardization: The below code normalizes the feature values by centering them around the mean and scaling them to have unit variance, which helps machine learning models work more effectively.



```
scaler = StandardScaler()

features_train_all_scaled = pd.DataFrame(
    scaler.fit_transform(features_train_all),
    columns=features_train_all.columns
)

features_test_all_scaled = pd.DataFrame(
    scaler.transform(features_test_all),
    columns=features_test_all.columns
)

features_train_all_scaled.describe()
```

```
        Pregnancies       Glucose  BloodPressure  SkinThickness       Insulin           BMI  DiabetesPedigreeFunction           Age
count  3.600000e+02  3.600000e+02   3.600000e+02   3.600000e+02  3.600000e+02  3.600000e+02              3.600000e+02  3.600000e+02
mean  -1.110223e-16 -6.908054e-17  -1.332268e-16  -1.973730e-17 -3.947460e-17  6.908054e-16              7.894919e-17 -2.627528e-16
std    1.001392e+00  1.001392e+00   1.001392e+00   1.001392e+00  1.001392e+00  1.001392e+00              1.001392e+00  1.001392e+00
min   -1.165391e+00 -2.608533e+00  -3.414367e+00  -2.617854e+00 -1.278584e+00 -1.983277e+00             -1.158101e+00 -1.085505e+00
25%   -8.742456e-01 -7.256323e-01  -6.703190e-01  -4.816377e-01 -2.113704e-01 -7.157254e-01             -6.731933e-01 -8.230003e-01
50%   -2.919544e-01 -1.970989e-01  -2.466057e-02  -6.922923e-03 -2.113704e-01 -4.712691e-02             -3.302444e-01 -2.979912e-01
75%    5.814825e-01  6.948012e-01   6.209979e-01   4.677918e-01 -1.362510e-01  5.553082e-01              4.617706e-01  6.645253e-01
max    3.784084e+00  2.445568e+00   4.010705e+00   3.672116e+00  7.300568e+00  4.800212e+00              5.169587e+00  3.114568e+00
```

8. Hyperparameter Tuning:
    - This step tunes the hyperparameters for the Random Forest and SVM models used in predicting diabetes.
    - The goal is to find the best performing parameter settings while keeping the computation reasonable.
    - GridSearchCV from scikit learn is used to run a thorough search across the chosen parameter grids.



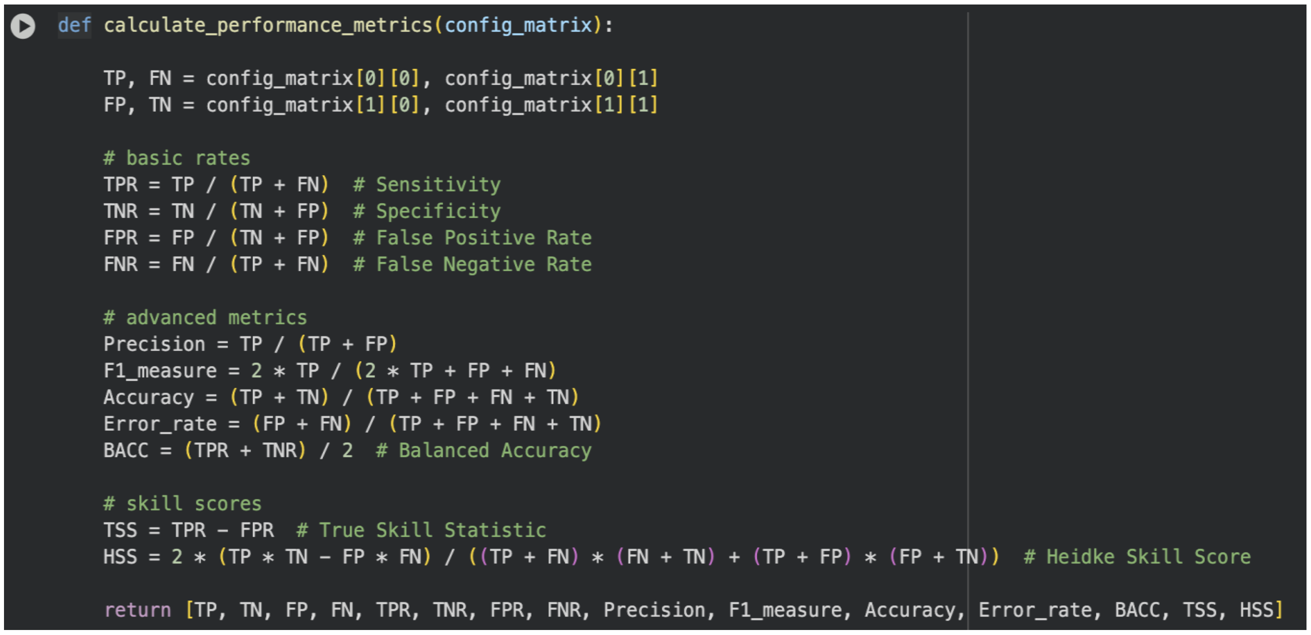
```
# Grid search for optimal parameters
print("\nUsing grid search for optimal parameters\n")
```

```
Using grid search for optimal parameters
```

```
param_grid_rf = {
    "n_estimators": [10, 20, 30, 40, 50, 60, 70, 80, 90, 100],
    "min_samples_split": [2, 4, 6, 8, 10]
}

# Grid search for Random Forest
rf_classifier = RandomForestClassifier()
grid_search_rf = GridSearchCV(rf_classifier, param_grid_rf, cv=10, n_jobs=-1)
grid_search_rf.fit(features_train_all_scaled, target_train_all)
best_rf_params = grid_search_rf.best_params_
print(f"Best Random Forest parameters: {best_rf_params}")

# Grid search for SVM
param_grid_svc = {"kernel": ["linear"], "C": range(1, 11)}
svc_classifier = SVC(probability=True)
grid_search_svc = GridSearchCV(svc_classifier, param_grid_svc, cv=10, n_jobs=-1)
grid_search_svc.fit(features_train_all_scaled, target_train_all)
best_svc_params = grid_search_svc.best_params_
print(f"Best SVM parameters: {best_svc_params}")
```

```
Best Random Forest parameters: {'min_samples_split': 4, 'n_estimators': 90}
Best SVM parameters: {'C': 1, 'kernel': 'linear'}
```

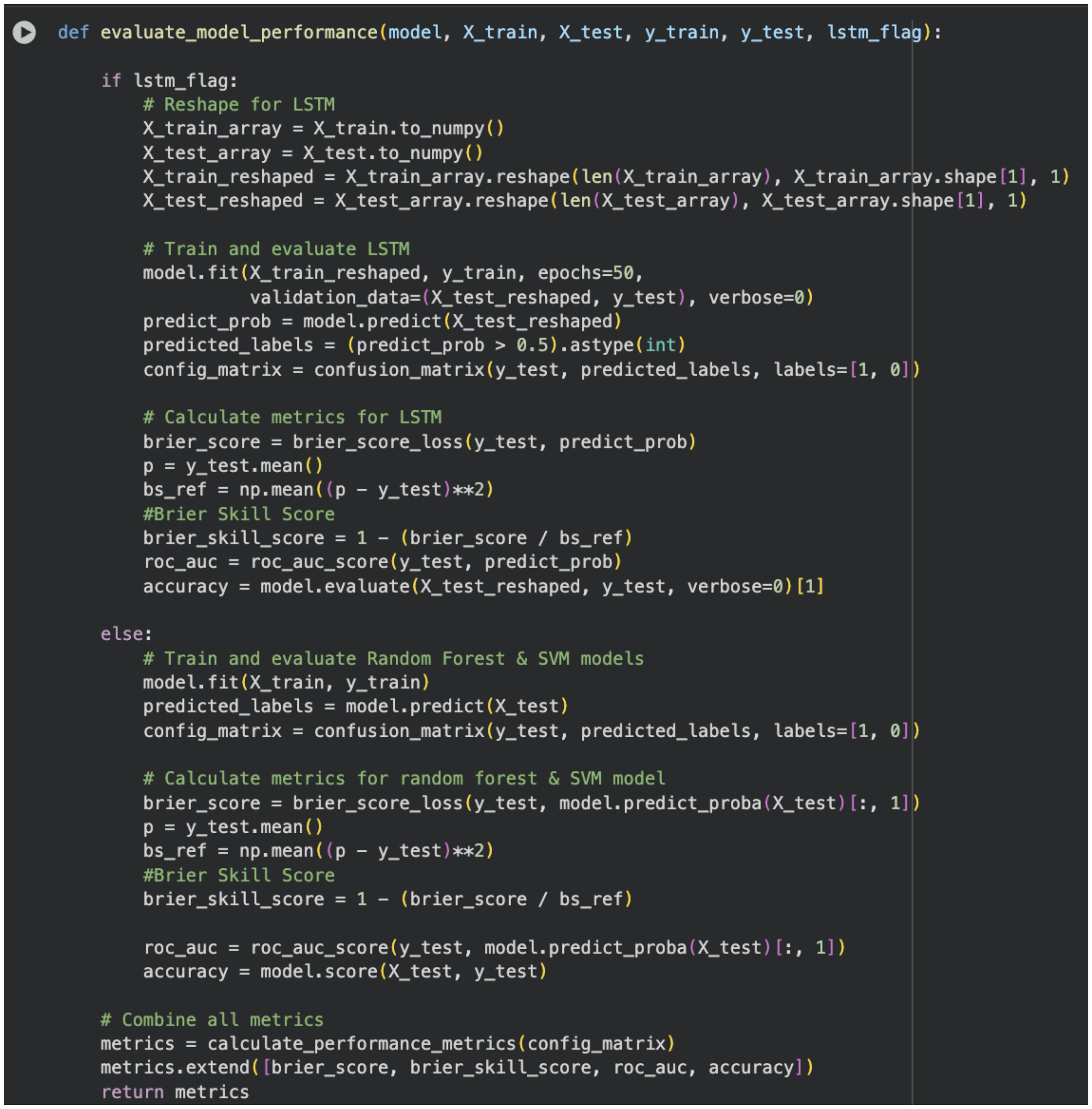
9. Classification Metrics Calculator: This function computes key performance metrics using a binary confusion matrix. It is meant for evaluating binary classifiers in machine learning tasks. The resulting metrics like accuracy, precision, recall, and several skill scores that give a overall view of how the model is performing and make it easier to compare different models.



```
def calculate_performance_metrics(config_matrix):

    TP, FN = config_matrix[0][0], config_matrix[0][1]
    FP, TN = config_matrix[1][0], config_matrix[1][1]

    # basic rates
    TPR = TP / (TP + FN)  # Sensitivity
    TNR = TN / (TN + FP)  # Specificity
    FPR = FP / (TN + FP)  # False Positive Rate
    FNR = FN / (TP + FN)  # False Negative Rate

    # advanced metrics
    Precision = TP / (TP + FP)
    F1_measure = 2 * TP / (2 * TP + FP + FN)
    Accuracy = (TP + TN) / (TP + FP + FN + TN)
    Error_rate = (FP + FN) / (TP + FP + FN + TN)
    BACC = (TPR + TNR) / 2  # Balanced Accuracy

    # skill scores
    TSS = TPR - FPR  # True Skill Statistic
    HSS = 2 * (TP * TN - FP * FN) / ((TP + FN) * (FN + TN) + (TP + FP) * (FP + TN))  # Heidke Skill Score

    return [TP, TN, FP, FN, TPR, TNR, FPR, FNR, Precision, F1_measure, Accuracy, Error_rate, BACC, TSS, HSS]
```

10. Model Evaluation Function:

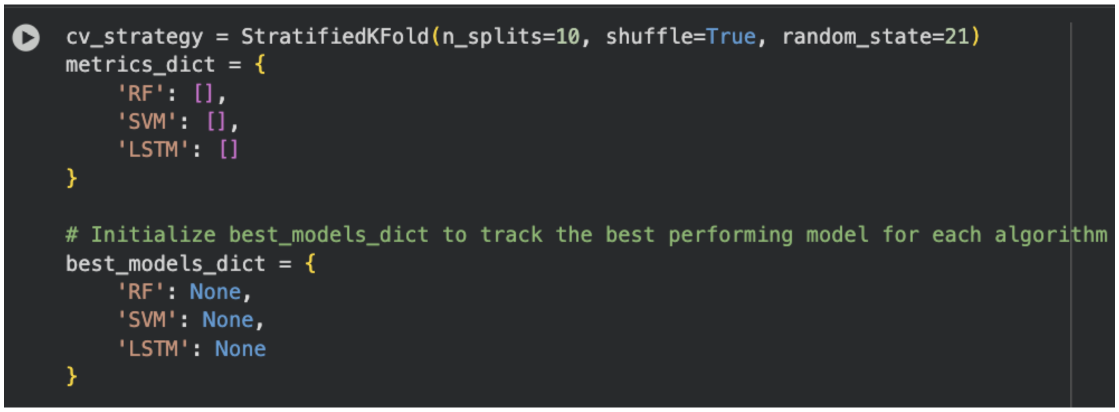
- This function trains a machine learning model and assesses the performance using a variety of metrics.
- It works with both traditional ML models and LSTM neural networks, applying the proper preprocessing and evaluation for each.
- Designed for binary classification, it provides metrics such as confusion matrix scores, ROC AUC, and the Brier score.



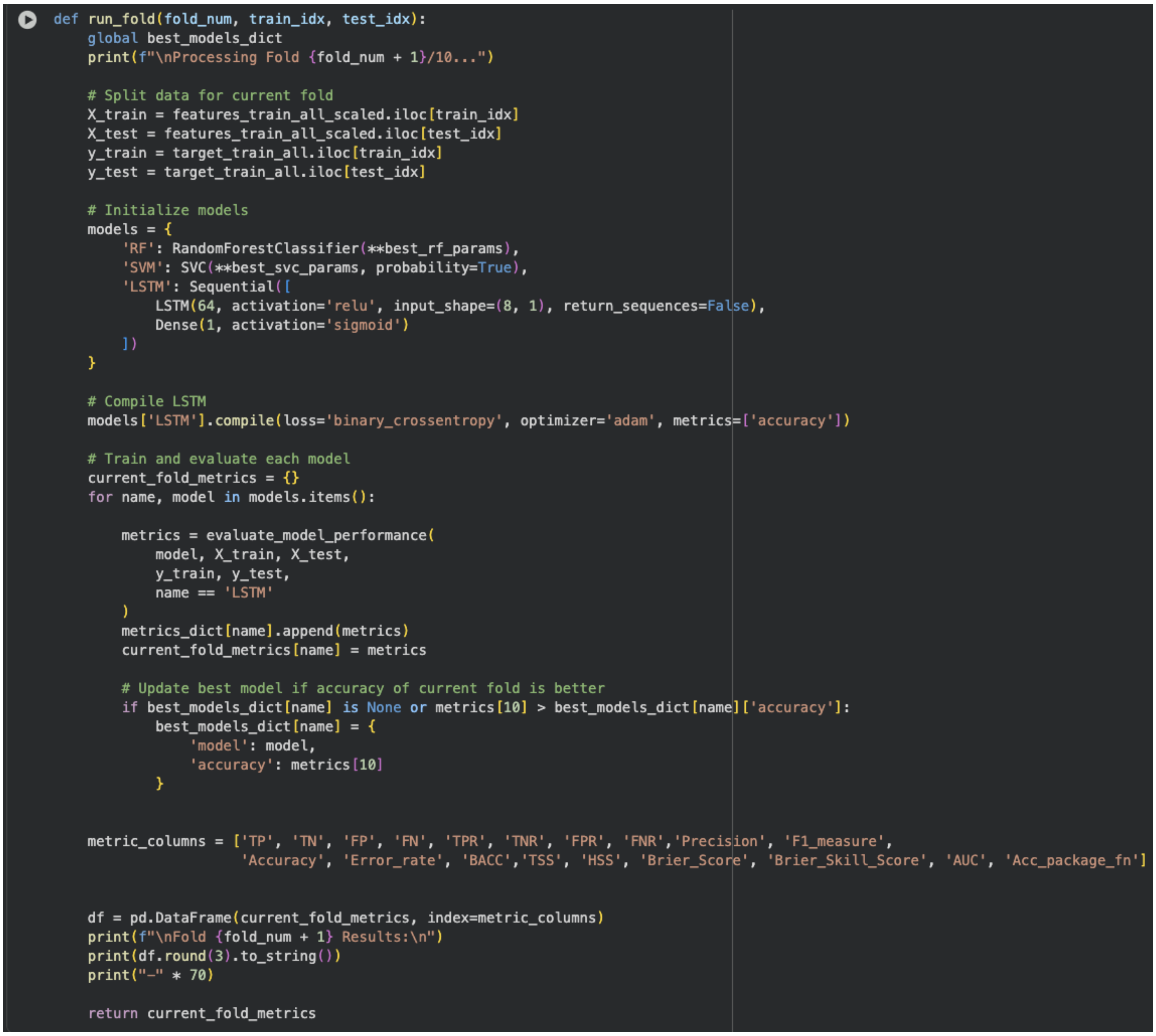
```python
def evaluate_model_performance(model, X_train, X_test, y_train, y_test, lstm_flag):

    if lstm_flag:
        # Reshape for LSTM
        X_train_array = X_train.to_numpy()
        X_test_array = X_test.to_numpy()
        X_train_reshaped = X_train_array.reshape(len(X_train_array), X_train_array.shape[1], 1)
        X_test_reshaped = X_test_array.reshape(len(X_test_array), X_test_array.shape[1], 1)

        # Train and evaluate LSTM
        model.fit(X_train_reshaped, y_train, epochs=50,
                  validation_data=(X_test_reshaped, y_test), verbose=0)
        predict_prob = model.predict(X_test_reshaped)
        predicted_labels = (predict_prob > 0.5).astype(int)
        config_matrix = confusion_matrix(y_test, predicted_labels, labels=[1, 0])

        # Calculate metrics for LSTM
        brier_score = brier_score_loss(y_test, predict_prob)
        p = y_test.mean()
        bs_ref = np.mean((p - y_test)**2)
        #Brier Skill Score
        brier_skill_score = 1 - (brier_score / bs_ref)
        roc_auc = roc_auc_score(y_test, predict_prob)
        accuracy = model.evaluate(X_test_reshaped, y_test, verbose=0)[1]

    else:
        # Train and evaluate Random Forest & SVM models
        model.fit(X_train, y_train)
        predicted_labels = model.predict(X_test)
        config_matrix = confusion_matrix(y_test, predicted_labels, labels=[1, 0])

        # Calculate metrics for random forest & SVM model
        brier_score = brier_score_loss(y_test, model.predict_proba(X_test)[:, 1])
        p = y_test.mean()
        bs_ref = np.mean((p - y_test)**2)
        #Brier Skill Score
        brier_skill_score = 1 - (brier_score / bs_ref)

        roc_auc = roc_auc_score(y_test, model.predict_proba(X_test)[:, 1])
        accuracy = model.score(X_test, y_test)

    # Combine all metrics
    metrics = calculate_performance_metrics(config_matrix)
    metrics.extend([brier_score, brier_skill_score, roc_auc, accuracy])
    return metrics
```

11.Cross Validation Function:

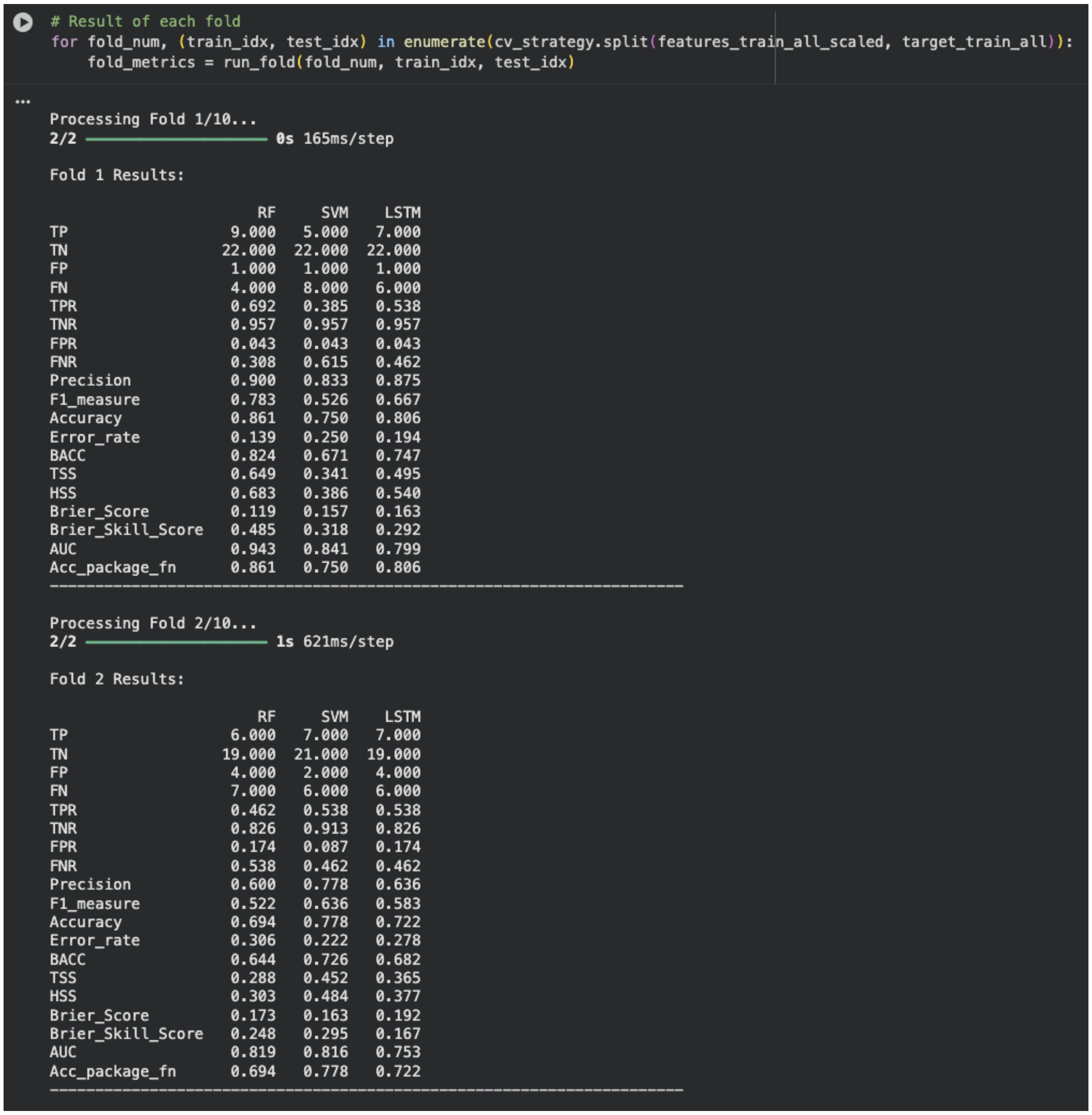
- Performs stratified k fold cross validation for multiple models at once.
- Supports both traditional ML models and LSTM networks, taking care of all required preprocessing and metric calculations.
- Includes progress tracking, error management, and detailed performance metrics to facilitate model comparison and evaluation.



```python
cv_strategy = StratifiedKFold(n_splits=10, shuffle=True, random_state=21)
metrics_dict = {
    'RF': [],
    'SVM': [],
    'LSTM': []
}

# Initialize best_models_dict to track the best performing model for each algorithm
best_models_dict = {
    'RF': None,
    'SVM': None,
    'LSTM': None
}
```

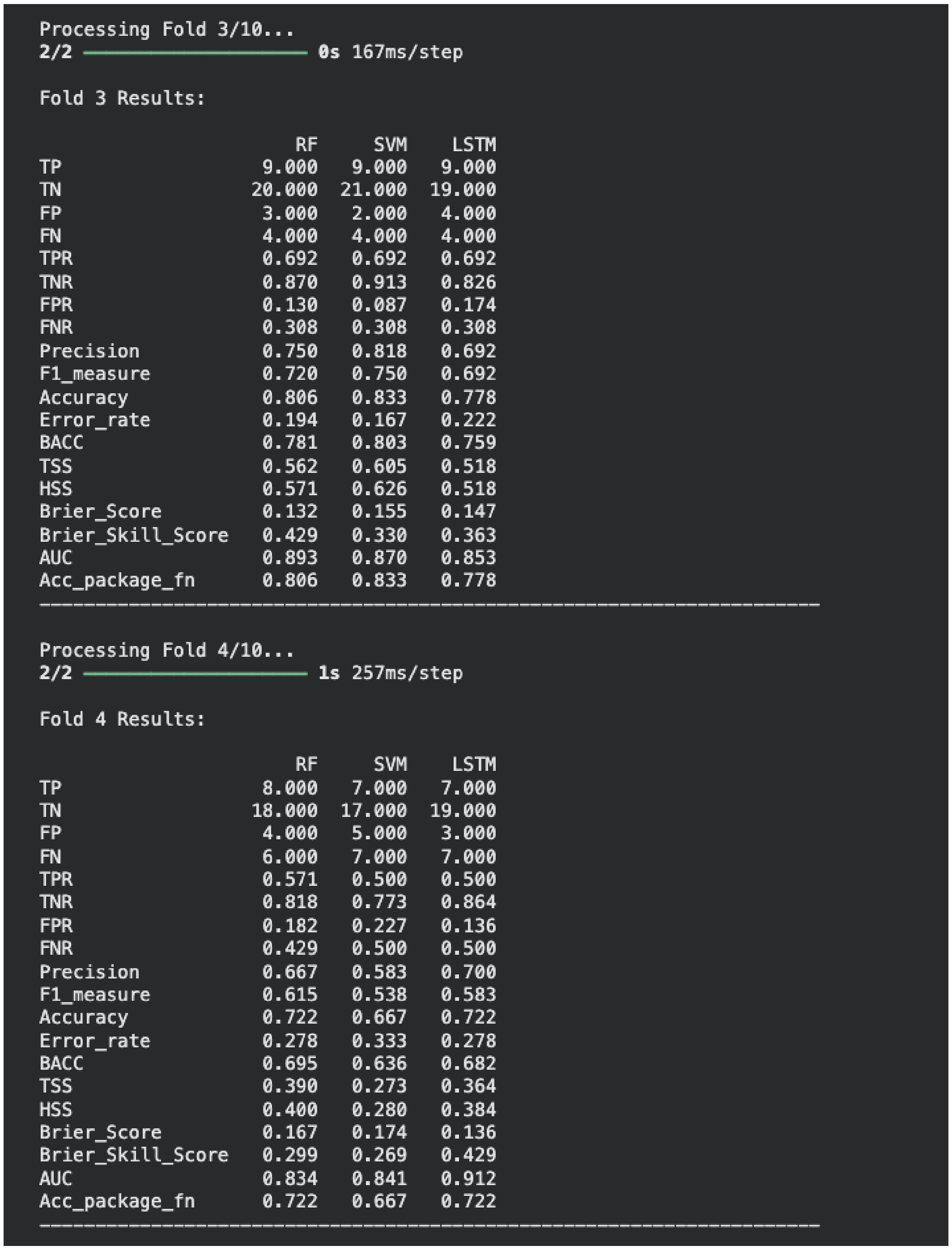


```python
def run_fold(fold_num, train_idx, test_idx):
    global best_models_dict
    print(f"\nProcessing Fold {fold_num + 1}/10...")

    # Split data for current fold
    X_train = features_train_all_scaled.iloc[train_idx]
    X_test = features_train_all_scaled.iloc[test_idx]
    y_train = target_train_all.iloc[train_idx]
    y_test = target_train_all.iloc[test_idx]

    # Initialize models
    models = {
        'RF': RandomForestClassifier(**best_rf_params),
        'SVM': SVC(**best_svc_params, probability=True),
        'LSTM': Sequential([
            LSTM(64, activation='relu', input_shape=(8, 1), return_sequences=False),
            Dense(1, activation='sigmoid')
        ])
    }

    # Compile LSTM
    models['LSTM'].compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

    # Train and evaluate each model
    current_fold_metrics = {}
    for name, model in models.items():

        metrics = evaluate_model_performance(
            model, X_train, X_test,
            y_train, y_test,
            name == 'LSTM'
        )
        metrics_dict[name].append(metrics)
        current_fold_metrics[name] = metrics

        # Update best model if accuracy of current fold is better
        if best_models_dict[name] is None or metrics[10] > best_models_dict[name]['accuracy']:
            best_models_dict[name] = {
                'model': model,
                'accuracy': metrics[10]
            }


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


    df = pd.DataFrame(current_fold_metrics, index=metric_columns)
    print(f"\nFold {fold_num + 1} Results:\n")
    print(df.round(3).to_string())
    print("-" * 70)

    return current_fold_metrics
```

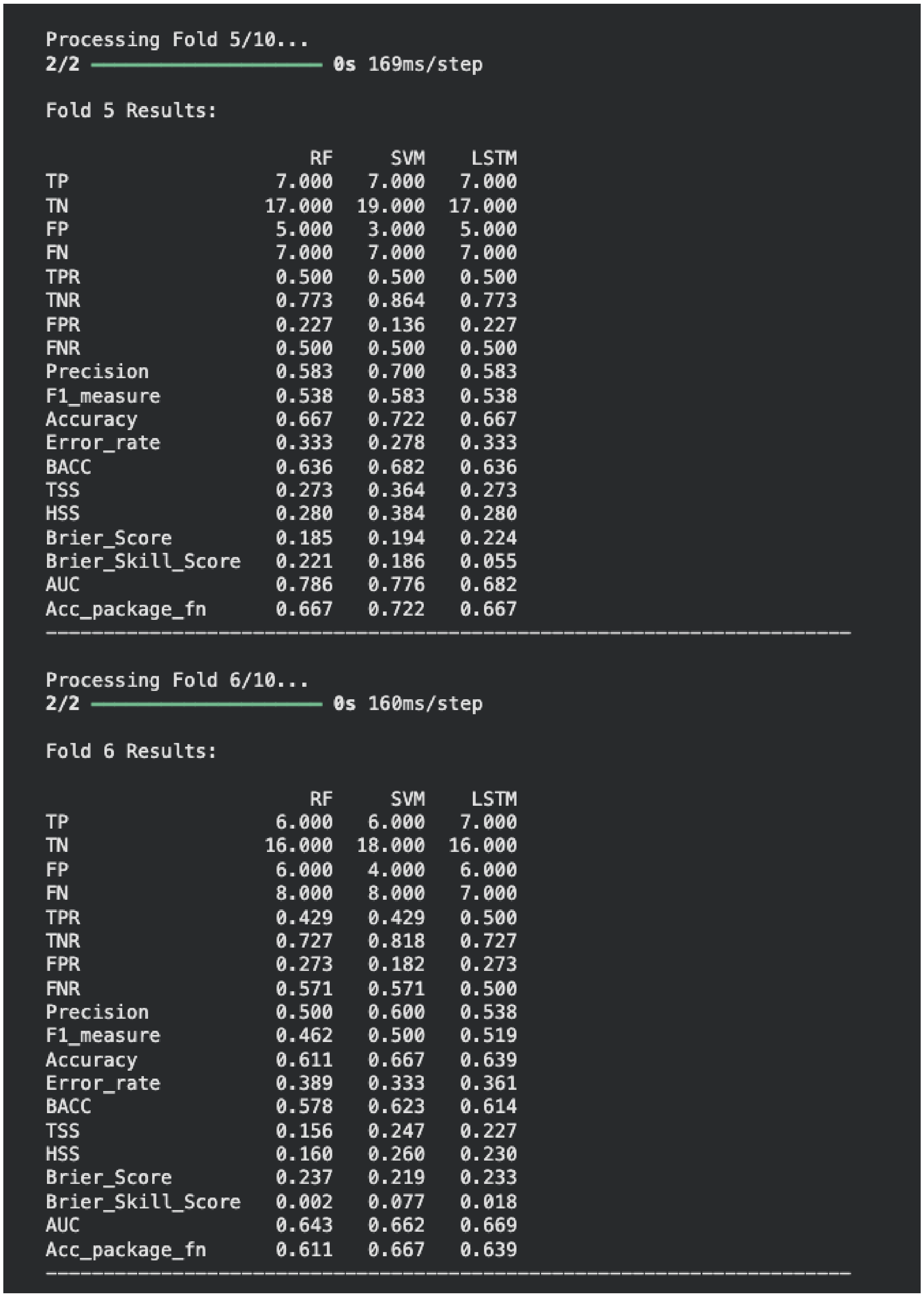


```
# Result of each fold
for fold_num, (train_idx, test_idx) in enumerate(cv_strategy.split(features_train_all_scaled, target_train_all)):
    fold_metrics = run_fold(fold_num, train_idx, test_idx)
```

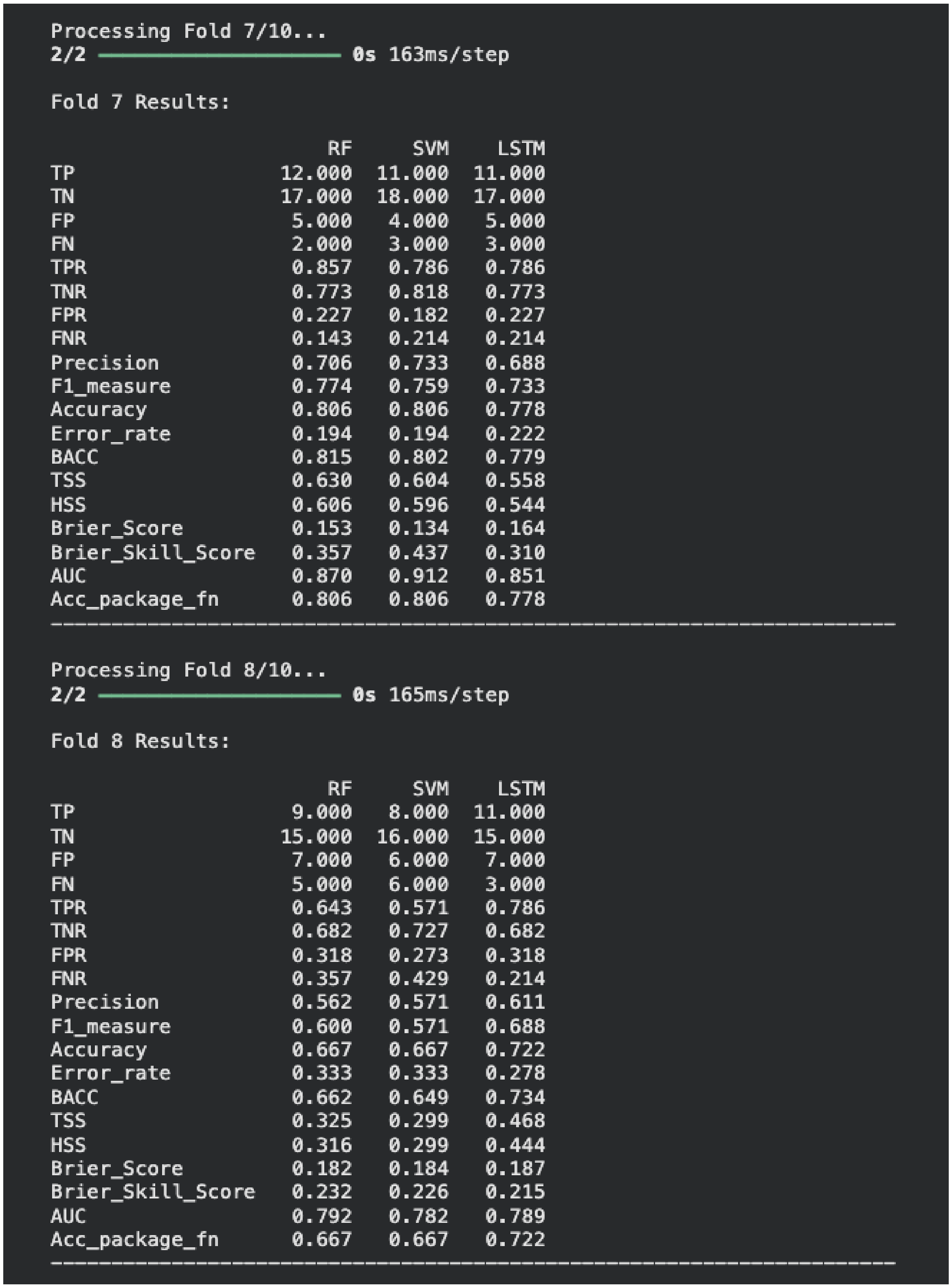
```
...
Processing Fold 1/10...
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step

Fold 1 Results:

                      RF     SVM    LSTM
TP                 9.000   5.000   7.000
TN                22.000  22.000  22.000
FP                 1.000   1.000   1.000
FN                 4.000   8.000   6.000
TPR                0.692   0.385   0.538
TNR                0.957   0.957   0.957
FPR                0.043   0.043   0.043
FNR                0.308   0.615   0.462
Precision          0.900   0.833   0.875
F1_measure         0.783   0.526   0.667
Accuracy           0.861   0.750   0.806
Error_rate         0.139   0.250   0.194
BACC               0.824   0.671   0.747
TSS                0.649   0.341   0.495
HSS                0.683   0.386   0.540
Brier_Score        0.119   0.157   0.163
Brier_Skill_Score  0.485   0.318   0.292
AUC                0.943   0.841   0.799
Acc_package_fn     0.861   0.750   0.806
----------------------------------------------------------------------

Processing Fold 2/10...
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 621ms/step

Fold 2 Results:

                      RF     SVM    LSTM
TP                 6.000   7.000   7.000
TN                19.000  21.000  19.000
FP                 4.000   2.000   4.000
FN                 7.000   6.000   6.000
TPR                0.462   0.538   0.538
TNR                0.826   0.913   0.826
FPR                0.174   0.087   0.174
FNR                0.538   0.462   0.462
Precision          0.600   0.778   0.636
F1_measure         0.522   0.636   0.583
Accuracy           0.694   0.778   0.722
Error_rate         0.306   0.222   0.278
BACC               0.644   0.726   0.682
TSS                0.288   0.452   0.365
HSS                0.303   0.484   0.377
Brier_Score        0.173   0.163   0.192
Brier_Skill_Score  0.248   0.295   0.167
AUC                0.819   0.816   0.753
Acc_package_fn     0.694   0.778   0.722
----------------------------------------------------------------------
```



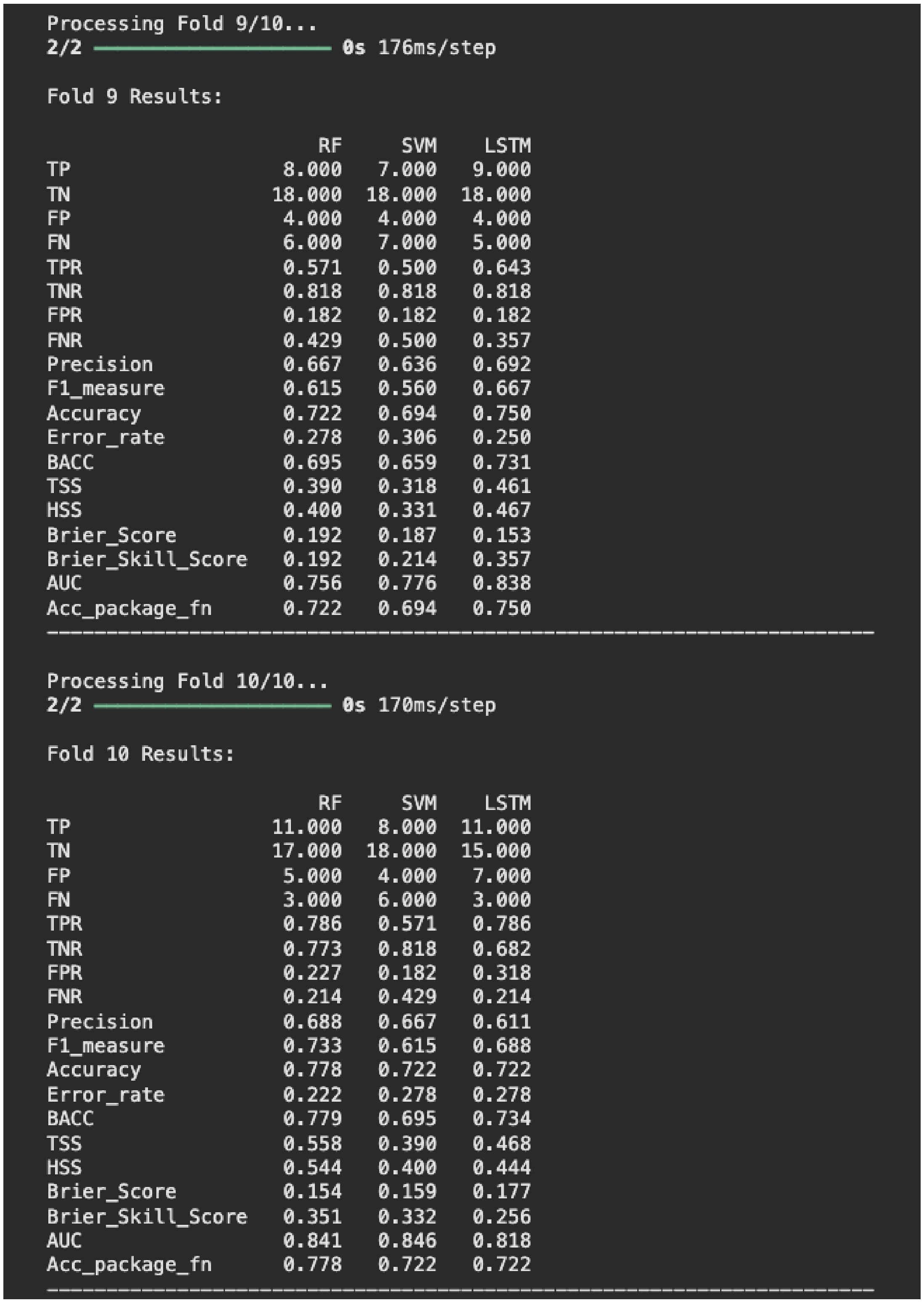
```
Processing Fold 3/10...
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 167ms/step

Fold 3 Results:

                        RF      SVM    LSTM
TP                   9.000    9.000   9.000
TN                  20.000   21.000  19.000
FP                   3.000    2.000   4.000
FN                   4.000    4.000   4.000
TPR                  0.692    0.692   0.692
TNR                  0.870    0.913   0.826
FPR                  0.130    0.087   0.174
FNR                  0.308    0.308   0.308
Precision            0.750    0.818   0.692
F1_measure           0.720    0.750   0.692
Accuracy             0.806    0.833   0.778
Error_rate           0.194    0.167   0.222
BACC                 0.781    0.803   0.759
TSS                  0.562    0.605   0.518
HSS                  0.571    0.626   0.518
Brier_Score          0.132    0.155   0.147
Brier_Skill_Score    0.429    0.330   0.363
AUC                  0.893    0.870   0.853
Acc_package_fn       0.806    0.833   0.778
----------------------------------------------------------------------

Processing Fold 4/10...
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 257ms/step

Fold 4 Results:

                        RF      SVM    LSTM
TP                   8.000    7.000   7.000
TN                  18.000   17.000  19.000
FP                   4.000    5.000   3.000
FN                   6.000    7.000   7.000
TPR                  0.571    0.500   0.500
TNR                  0.818    0.773   0.864
FPR                  0.182    0.227   0.136
FNR                  0.429    0.500   0.500
Precision            0.667    0.583   0.700
F1_measure           0.615    0.538   0.583
Accuracy             0.722    0.667   0.722
Error_rate           0.278    0.333   0.278
BACC                 0.695    0.636   0.682
TSS                  0.390    0.273   0.364
HSS                  0.400    0.280   0.384
Brier_Score          0.167    0.174   0.136
Brier_Skill_Score    0.299    0.269   0.429
AUC                  0.834    0.841   0.912
Acc_package_fn       0.722    0.667   0.722
----------------------------------------------------------------------
```



```
Processing Fold 5/10...
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 169ms/step

Fold 5 Results:

                        RF     SVM    LSTM
TP                   7.000   7.000   7.000
TN                  17.000  19.000  17.000
FP                   5.000   3.000   5.000
FN                   7.000   7.000   7.000
TPR                  0.500   0.500   0.500
TNR                  0.773   0.864   0.773
FPR                  0.227   0.136   0.227
FNR                  0.500   0.500   0.500
Precision            0.583   0.700   0.583
F1_measure           0.538   0.583   0.538
Accuracy             0.667   0.722   0.667
Error_rate           0.333   0.278   0.333
BACC                 0.636   0.682   0.636
TSS                  0.273   0.364   0.273
HSS                  0.280   0.384   0.280
Brier_Score          0.185   0.194   0.224
Brier_Skill_Score    0.221   0.186   0.055
AUC                  0.786   0.776   0.682
Acc_package_fn       0.667   0.722   0.667
----------------------------------------------------------------------

Processing Fold 6/10...
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 160ms/step

Fold 6 Results:

                        RF     SVM    LSTM
TP                   6.000   6.000   7.000
TN                  16.000  18.000  16.000
FP                   6.000   4.000   6.000
FN                   8.000   8.000   7.000
TPR                  0.429   0.429   0.500
TNR                  0.727   0.818   0.727
FPR                  0.273   0.182   0.273
FNR                  0.571   0.571   0.500
Precision            0.500   0.600   0.538
F1_measure           0.462   0.500   0.519
Accuracy             0.611   0.667   0.639
Error_rate           0.389   0.333   0.361
BACC                 0.578   0.623   0.614
TSS                  0.156   0.247   0.227
HSS                  0.160   0.260   0.230
Brier_Score          0.237   0.219   0.233
Brier_Skill_Score    0.002   0.077   0.018
AUC                  0.643   0.662   0.669
Acc_package_fn       0.611   0.667   0.639
----------------------------------------------------------------------
```

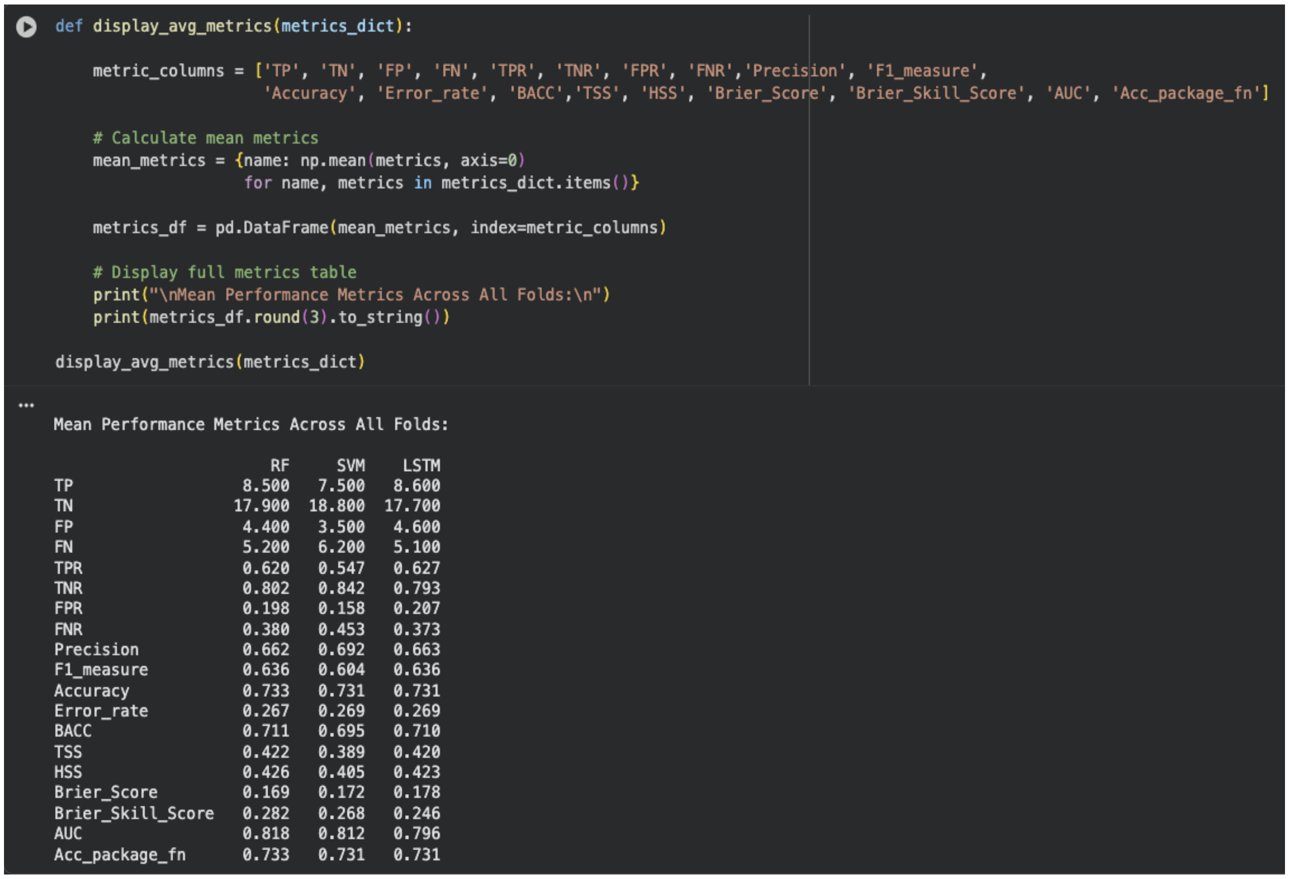


```
Processing Fold 7/10...
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 163ms/step

Fold 7 Results:

                       RF      SVM    LSTM
TP                 12.000  11.000  11.000
TN                 17.000  18.000  17.000
FP                  5.000   4.000   5.000
FN                  2.000   3.000   3.000
TPR                 0.857   0.786   0.786
TNR                 0.773   0.818   0.773
FPR                 0.227   0.182   0.227
FNR                 0.143   0.214   0.214
Precision           0.706   0.733   0.688
F1_measure          0.774   0.759   0.733
Accuracy            0.806   0.806   0.778
Error_rate          0.194   0.194   0.222
BACC                0.815   0.802   0.779
TSS                 0.630   0.604   0.558
HSS                 0.606   0.596   0.544
Brier_Score         0.153   0.134   0.164
Brier_Skill_Score   0.357   0.437   0.310
AUC                 0.870   0.912   0.851
Acc_package_fn      0.806   0.806   0.778
--------------------------------------------------------------

Processing Fold 8/10...
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step

Fold 8 Results:

                       RF      SVM    LSTM
TP                  9.000   8.000  11.000
TN                 15.000  16.000  15.000
FP                  7.000   6.000   7.000
FN                  5.000   6.000   3.000
TPR                 0.643   0.571   0.786
TNR                 0.682   0.727   0.682
FPR                 0.318   0.273   0.318
FNR                 0.357   0.429   0.214
Precision           0.562   0.571   0.611
F1_measure          0.600   0.571   0.688
Accuracy            0.667   0.667   0.722
Error_rate          0.333   0.333   0.278
BACC                0.662   0.649   0.734
TSS                 0.325   0.299   0.468
HSS                 0.316   0.299   0.444
Brier_Score         0.182   0.184   0.187
Brier_Skill_Score   0.232   0.226   0.215
AUC                 0.792   0.782   0.789
Acc_package_fn      0.667   0.667   0.722
--------------------------------------------------------------
```



```
Processing Fold 9/10...
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 176ms/step

Fold 9 Results:

                       RF      SVM    LSTM
TP                  8.000    7.000   9.000
TN                 18.000   18.000  18.000
FP                  4.000    4.000   4.000
FN                  6.000    7.000   5.000
TPR                 0.571    0.500   0.643
TNR                 0.818    0.818   0.818
FPR                 0.182    0.182   0.182
FNR                 0.429    0.500   0.357
Precision           0.667    0.636   0.692
F1_measure          0.615    0.560   0.667
Accuracy            0.722    0.694   0.750
Error_rate          0.278    0.306   0.250
BACC                0.695    0.659   0.731
TSS                 0.390    0.318   0.461
HSS                 0.400    0.331   0.467
Brier_Score         0.192    0.187   0.153
Brier_Skill_Score   0.192    0.214   0.357
AUC                 0.756    0.776   0.838
Acc_package_fn      0.722    0.694   0.750
----------------------------------------------------------------------

Processing Fold 10/10...
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 170ms/step

Fold 10 Results:

                       RF      SVM    LSTM
TP                 11.000    8.000  11.000
TN                 17.000   18.000  15.000
FP                  5.000    4.000   7.000
FN                  3.000    6.000   3.000
TPR                 0.786    0.571   0.786
TNR                 0.773    0.818   0.682
FPR                 0.227    0.182   0.318
FNR                 0.214    0.429   0.214
Precision           0.688    0.667   0.611
F1_measure          0.733    0.615   0.688
Accuracy            0.778    0.722   0.722
Error_rate          0.222    0.278   0.278
BACC                0.779    0.695   0.734
TSS                 0.558    0.390   0.468
HSS                 0.544    0.400   0.444
Brier_Score         0.154    0.159   0.177
Brier_Skill_Score   0.351    0.332   0.256
AUC                 0.841    0.846   0.818
Acc_package_fn      0.778    0.722   0.722
----------------------------------------------------------------------
```

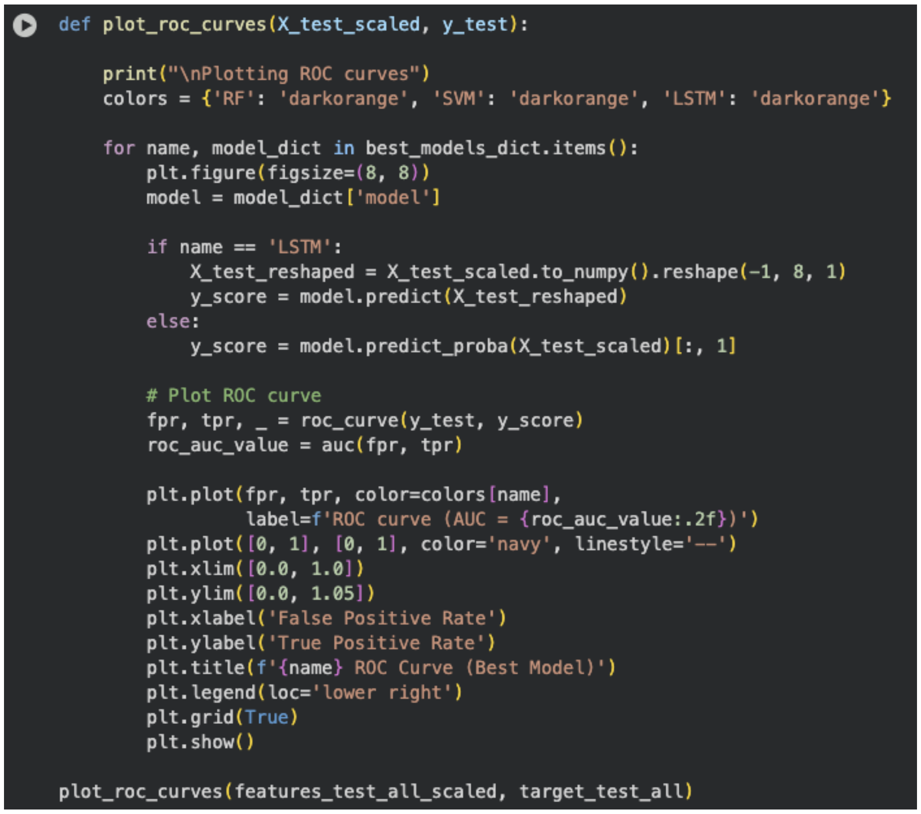
12. Average of Metrics: The below code calculates the average of all folds and gives proper metrics to compare across all models and folds.



```
def display_avg_metrics(metrics_dict):

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

    # Calculate mean metrics
    mean_metrics = {name: np.mean(metrics, axis=0)
                    for name, metrics in metrics_dict.items()}

    metrics_df = pd.DataFrame(mean_metrics, index=metric_columns)

    # Display full metrics table
    print("\nMean Performance Metrics Across All Folds:\n")
    print(metrics_df.round(3).to_string())

display_avg_metrics(metrics_dict)
```

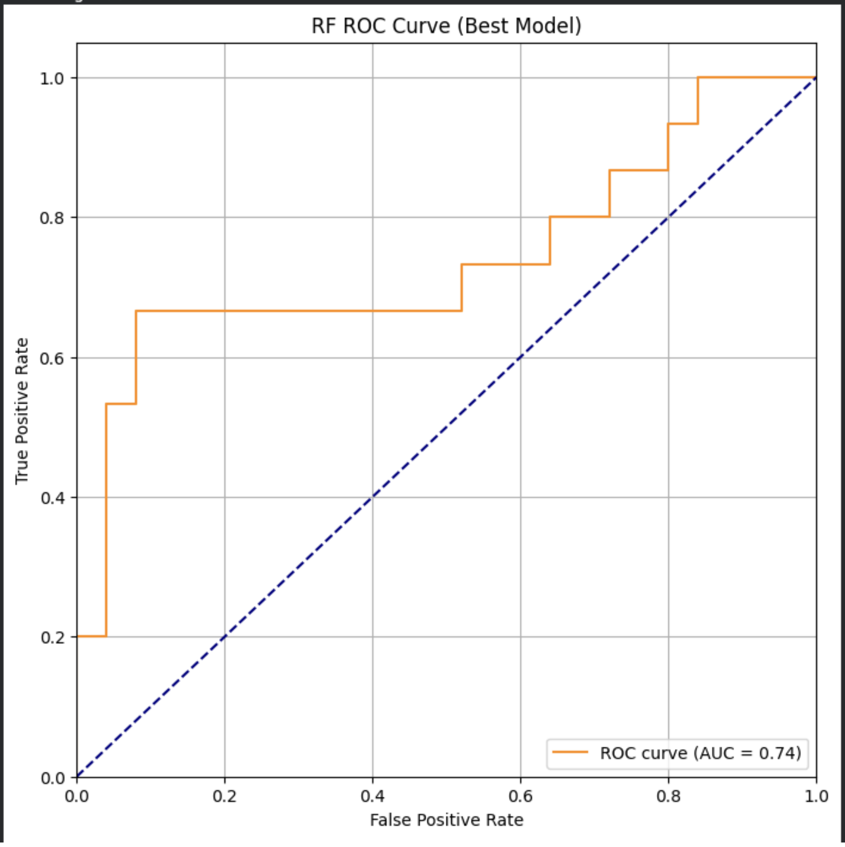
```
...
Mean Performance Metrics Across All Folds:

                       RF     SVM    LSTM
TP                  8.500   7.500   8.600
TN                 17.900  18.800  17.700
FP                  4.400   3.500   4.600
FN                  5.200   6.200   5.100
TPR                 0.620   0.547   0.627
TNR                 0.802   0.842   0.793
FPR                 0.198   0.158   0.207
FNR                 0.380   0.453   0.373
Precision           0.662   0.692   0.663
F1_measure          0.636   0.604   0.636
Accuracy            0.733   0.731   0.731
Error_rate          0.267   0.269   0.269
BACC                0.711   0.695   0.710
TSS                 0.422   0.389   0.420
HSS                 0.426   0.405   0.423
Brier_Score         0.169   0.172   0.178
Brier_Skill_Score   0.282   0.268   0.246
AUC                 0.818   0.812   0.796
Acc_package_fn      0.733   0.731   0.731
```

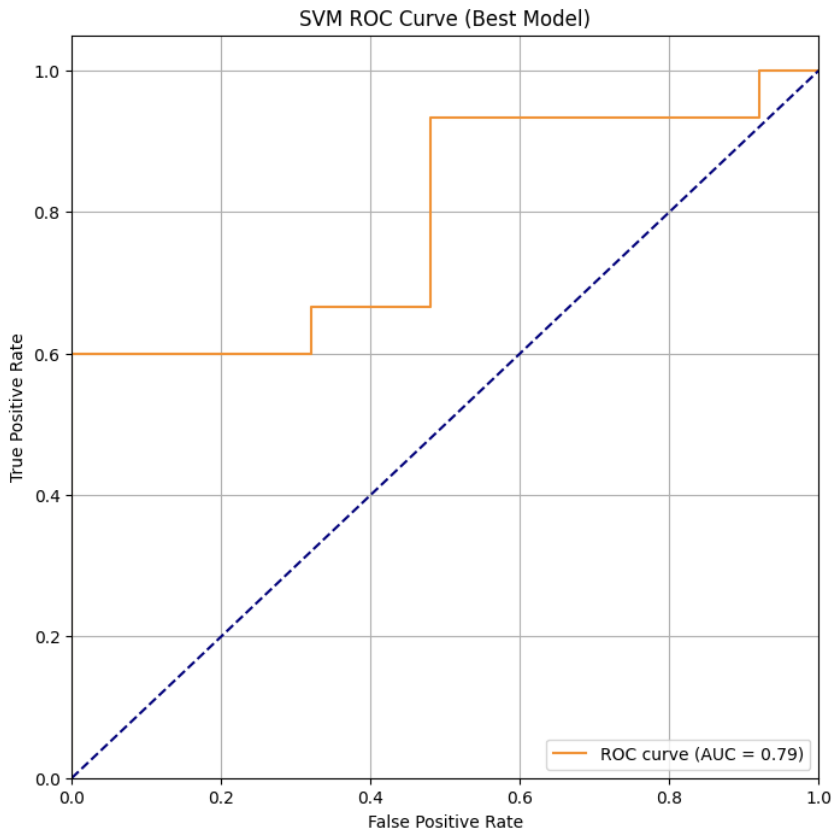
13. Evaluating Algorithm Performance: Compare the ROC curves and AUC scores of different algorithms to assess their performance on the test dataset.



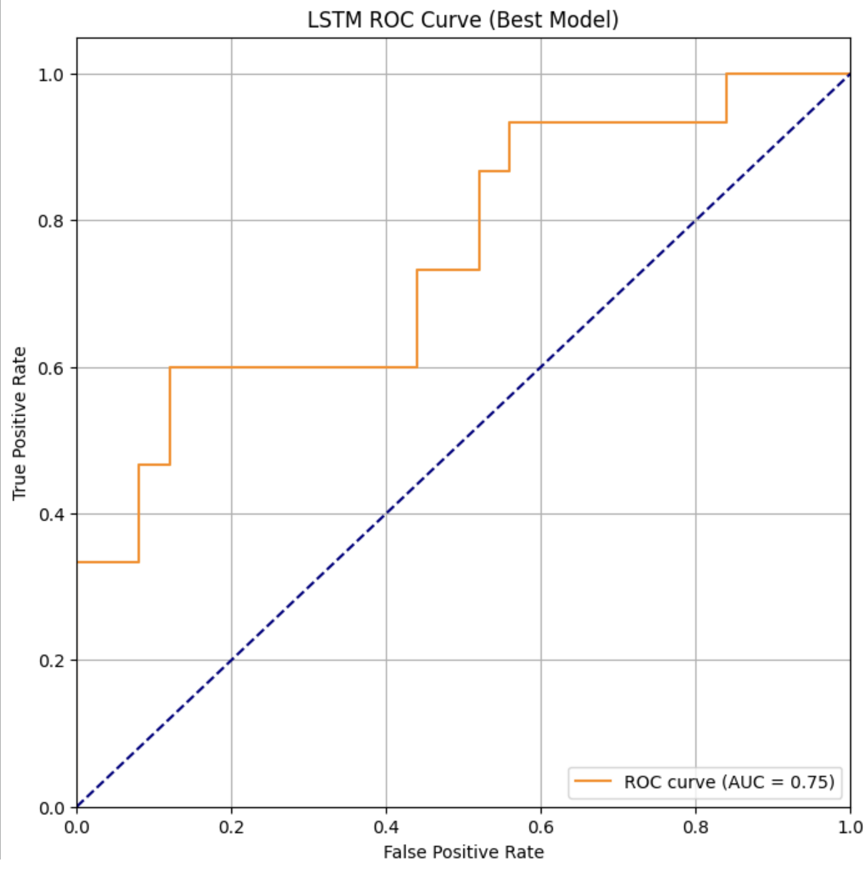
```python
def plot_roc_curves(X_test_scaled, y_test):

    print("\nPlotting ROC curves")
    colors = {'RF': 'darkorange', 'SVM': 'darkorange', 'LSTM': 'darkorange'}

    for name, model_dict in best_models_dict.items():
        plt.figure(figsize=(8, 8))
        model = model_dict['model']

        if name == 'LSTM':
            X_test_reshaped = X_test_scaled.to_numpy().reshape(-1, 8, 1)
            y_score = model.predict(X_test_reshaped)
        else:
            y_score = model.predict_proba(X_test_scaled)[:, 1]

        # Plot ROC curve
        fpr, tpr, _ = roc_curve(y_test, y_score)
        roc_auc_value = auc(fpr, tpr)

        plt.plot(fpr, tpr, color=colors[name],
                 label=f'ROC curve (AUC = {roc_auc_value:.2f})')
        plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title(f'{name} ROC Curve (Best Model)')
        plt.legend(loc='lower right')
        plt.grid(True)
        plt.show()

plot_roc_curves(features_test_all_scaled, target_test_all)
```

```
Plotting ROC curves
```

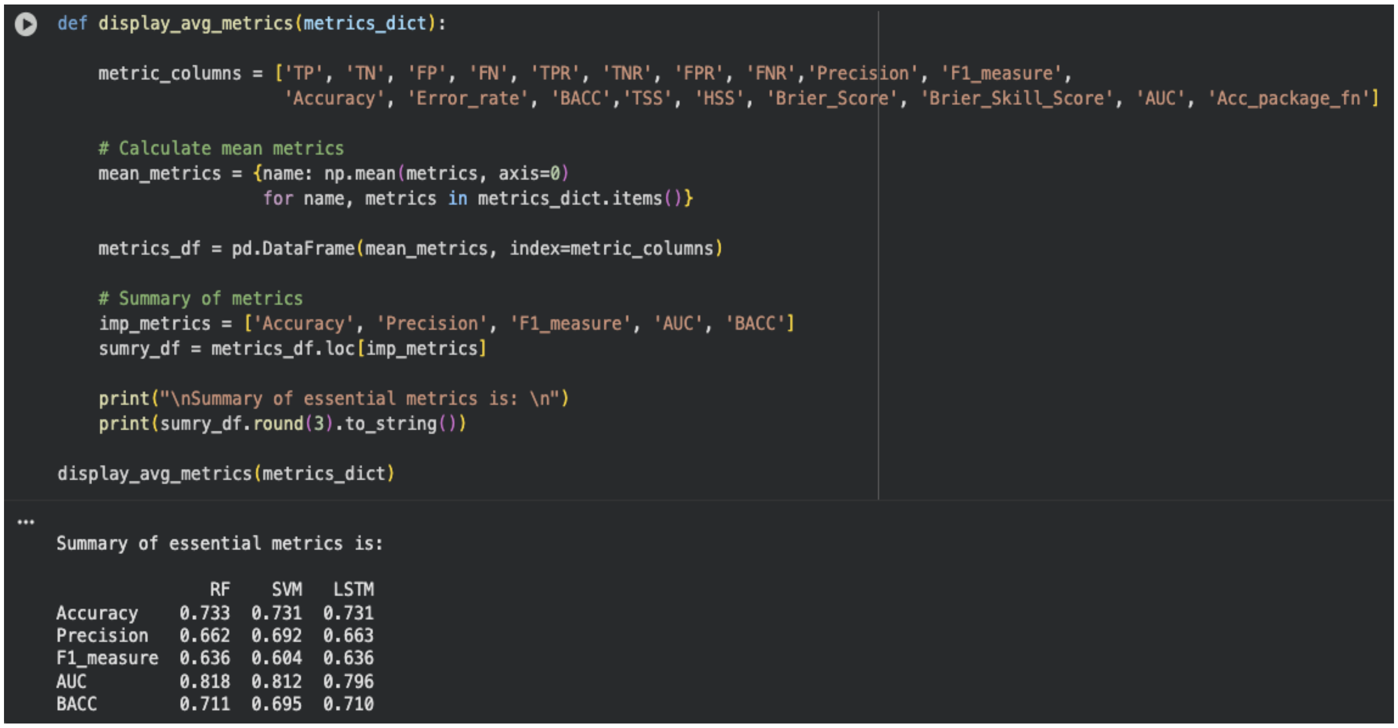




```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
```



14.Summary of Key Metrics



```
def display_avg_metrics(metrics_dict):

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

    # Calculate mean metrics
    mean_metrics = {name: np.mean(metrics, axis=0)
                    for name, metrics in metrics_dict.items()}

    metrics_df = pd.DataFrame(mean_metrics, index=metric_columns)

    # Summary of metrics
    imp_metrics = ['Accuracy', 'Precision', 'F1_measure', 'AUC', 'BACC']
    sumry_df = metrics_df.loc[imp_metrics]

    print("\nSummary of essential metrics is: \n")
    print(sumry_df.round(3).to_string())

display_avg_metrics(metrics_dict)
```

```
Summary of essential metrics is:

              RF    SVM   LSTM
Accuracy    0.733  0.731  0.731
Precision   0.662  0.692  0.663
F1_measure  0.636  0.604  0.636
AUC         0.818  0.812  0.796
BACC        0.711  0.695  0.710
```

- **Comparison Based on Metrics**
  1. The Random Forest model emerged as the top performer, consistently achieving higher accuracy than both SVM and LSTM. This indicates superior overall predictive capability.
     - Demonstrated the most balanced metrics across folds.
     - Recorded the highest Balanced Accuracy (BACC), reflecting strong performance on both classes.
     - Achieved a solid F1score, showing a good trade-off between precision and recall.

  2. ROC Curve Analysis:
     - The Random Forest's ROC curve shows a strong early increase in true positive rate while maintaining low false positive rates.
     - The curve appears smoother than that of the LSTM, suggesting more stable and consistent predictions.

    - It matches the SVM’s AUC but performs slightly better at certain threshold levels.
3. While SVM delivers nearly comparable results and LSTM shows competitive performance, the Random Forest model is the most practical and effective choice for this diabetes prediction task due to its slightly higher accuracy and easier implementation and maintenance.
4. Training with a larger dataset may alter the results, potentially enhancing all models as clearer patterns emerge with more data.

- **Conclusion**

  The comparative study shows that although all three models perform reasonably well for diabetes prediction, the Random Forest classifier provides the best balance between accuracy and computational efficiency. The LSTM model, while promising, demands significantly higher computational resources without yielding major performance gains for this dataset.

- **Referral Link**

  https://github.com/Aniket-NJIT/khalate_aniket_finaltermproject