Name: Prabhath Vinay Vipparthi

NJIT UCID: PV342

Email Address: pv342@njit.edu

Professor: Dr. Yasser, Abduallah

Course: Data Mining, CS634

Data Mining - Final Project

Introduction:

This project implements three binary classification algorithms for heart failure prediction using a medical dataset. The required models are Random Forest, LSTM deep learning, and KNN traditional machine learning. The project uses 10-fold cross-validation and manually calculates all performance metrics including accuracy, precision, recall, F1 score, TSS, HSS, ROC/AUC, and Brier scores. The goal is to compare these algorithms on their ability to predict heart failure from patient clinical data.

Dataset Link: https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction/data

Project Workflow

1. Data Loading and Exploration

- Loaded the heart failure prediction dataset from Kaggle
- Checked dataset shape, data types, and missing values
- Analyzed target variable distribution (HeartDisease)

2. Data Preprocessing

- Encoded categorical variables (Sex, ChestPainType, RestingECG, ExerciseAngina, ST_Slope)
- Scaled numerical features using StandardScaler
- Separated features and target variable

3. Model Implementation

- Set up 10-fold cross-validation
- Implemented three algorithms:
    - Random Forest Classifier
    - LSTM neural network
    - K-Nearest Neighbors

- Trained and tested each model on all 10 folds

4. Performance Evaluation

- Calculated all required metrics manually for each fold
- Generated ROC curves and AUC scores
- Computed average performance across all folds
- Created comparison tables

5. Performance Metrics

The project calculates and reports the following metrics for each algorithm across 10-fold cross-validation:

Basic Classification Metrics:

- TP, TN, FP, FN (True/False Positives/Negatives)
- P, N (Actual Positives/Negatives)
- TPR, TNR, FPR, FNR (Rates)
- Accuracy, Balanced Accuracy
- Precision, Recall, F1 Score
- Error Rate

Advanced Metrics:

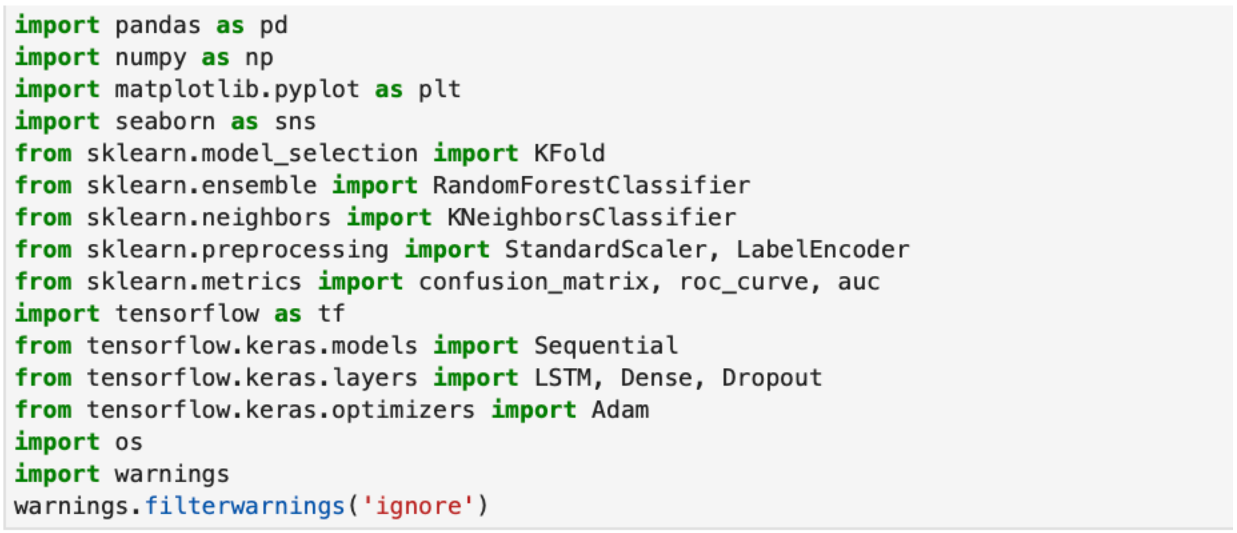
- TSS (True Skill Statistic)
- HSS (Heidke Skill Score)
- ROC curves and AUC (Area Under Curve)
- BS (Brier Score)
- BSS (Brier Skill Score)

6. Results Analysis

- Compared all three algorithms
- Identified best performing model
- Discussed findings and recommendations

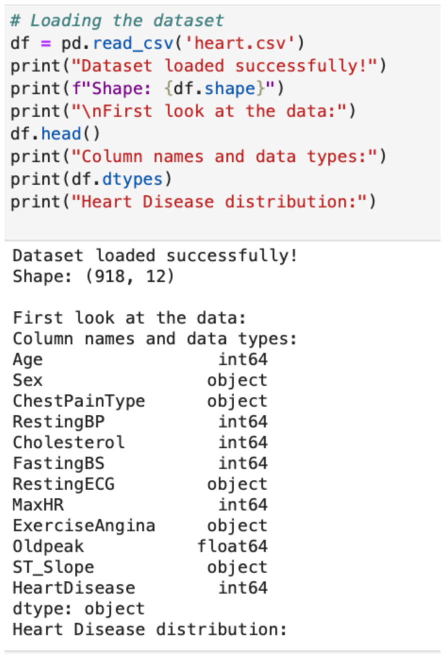
Results Screenshots:

1. **Importing the packages and libraries**



```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import confusion_matrix, roc_curve, auc
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam
import os
import warnings
warnings.filterwarnings('ignore')
```

2. **Loading the dataset**



```python
# Loading the dataset
df = pd.read_csv('heart.csv')
print("Dataset loaded successfully!")
print(f"Shape: {df.shape}")
print("\nFirst look at the data:")
df.head()
print("Column names and data types:")
print(df.dtypes)
print("Heart Disease distribution:")
```

```
Dataset loaded successfully!
Shape: (918, 12)

First look at the data:
Column names and data types:
Age                 int64
Sex                object
ChestPainType      object
RestingBP           int64
Cholesterol         int64
FastingBS           int64
RestingECG         object
MaxHR               int64
ExerciseAngina     object
Oldpeak           float64
ST_Slope           object
HeartDisease        int64
dtype: object
Heart Disease distribution:
```

3. Basic information of dataset

```
print("Basic dataset information")
print("Dataset shape:", df.shape)
print("Number of patients:", df.shape[0])
print("Number of features:", df.shape[1])
```

```
Basic dataset information
Dataset shape: (918, 12)
Number of patients: 918
Number of features: 12
```

```
df = pd.read_csv('heart.csv')
print("Dataset loaded successfully")
print("Shape:", df.shape)
print("\nFirst look at the data:")
print(df.head())
print("\nTarget variable distribution:")
print(df['HeartDisease'].value_counts())
```

```
Dataset loaded successfully
Shape: (918, 12)

First look at the data:
   Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  MaxHR  \
0   40   M           ATA        140          289          0     Normal    172   
1   49   F           NAP        160          180          0     Normal    156   
2   37   M           ATA        130          283          0         ST     98   
3   48   F           ASY        138          214          0     Normal    108   
4   54   M           NAP        150          195          0     Normal    122   

  ExerciseAngina  Oldpeak ST_Slope  HeartDisease  
0              N      0.0       Up             0  
1              N      1.0     Flat             1  
2              N      0.0       Up             0  
3              Y      1.5     Flat             1  
4              N      0.0       Up             0  

Target variable distribution:
HeartDisease
1    508
0    410
Name: count, dtype: int64
```

4. Separating The Dataset into Features and Target Value

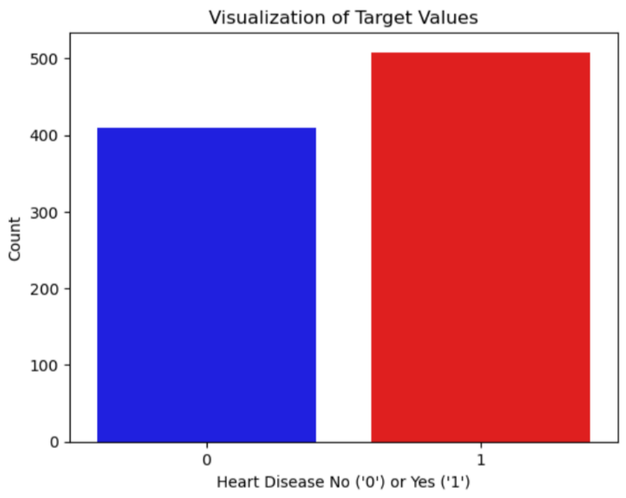
```
# Seperating the Features and Target Labels
heartFeature = df.iloc[:, :-1]
heartTarget = df.iloc[:, -1]
print("Features data")
print(heartFeature.head())
print("Target labels")
print(heartTarget.head())
```

```
Features data
   Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  MaxHR  \
0   40   M           ATA        140          289          0     Normal    172   
1   49   F           NAP        160          180          0     Normal    156   
2   37   M           ATA        130          283          0         ST     98   
3   48   F           ASY        138          214          0     Normal    108   
4   54   M           NAP        150          195          0     Normal    122   

  ExerciseAngina  Oldpeak ST_Slope  
0              N      0.0       Up  
1              N      1.0     Flat  
2              N      0.0       Up  
3              Y      1.5     Flat  
4              N      0.0       Up  
Target labels
0    0
1    1
2    0
3    1
4    0
Name: HeartDisease, dtype: int64
```

5. Data Visualization

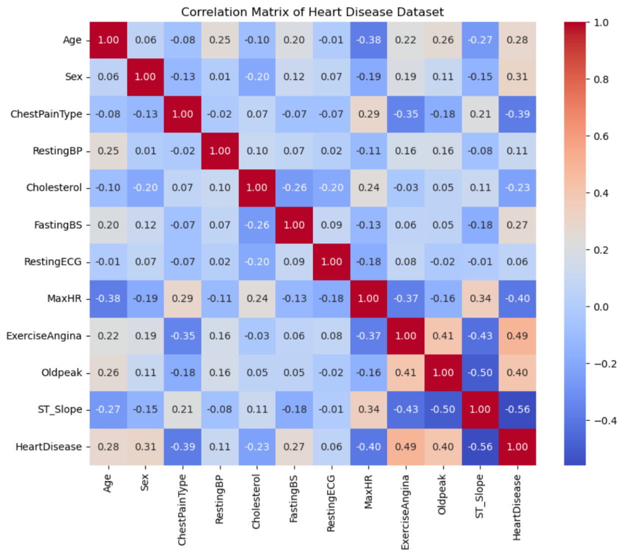
- **Visualization of Target Labels**

```
# Visualization of Target Labels
sns.countplot(x=heartTarget, palette=["blue", "red"])
plt.title("Visualization of Target Values")
plt.xlabel("Heart Disease No ('0') or Yes ('1')")
plt.ylabel("Count")
plt.show()
print()
```



- **Correlation Matrix**

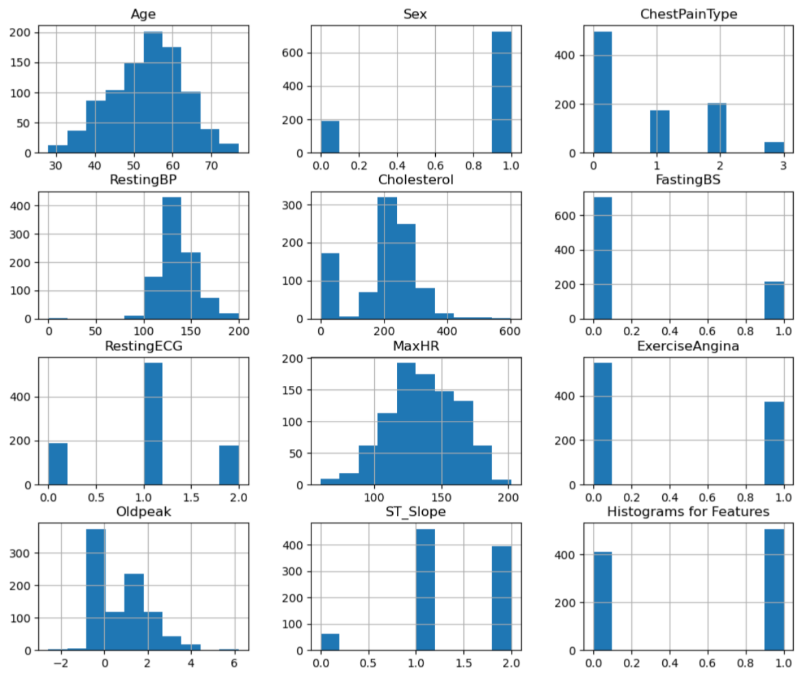
```
df_encoded = df.copy()
from sklearn.preprocessing import LabelEncoder
categorical_columns = ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope']
for column in categorical_columns:
    le = LabelEncoder()
    df_encoded[column] = le.fit_transform(df_encoded[column])
correlation_matrix = df_encoded.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Correlation Matrix of Heart Disease Dataset')
plt.show()
```



	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
Age	1.00	0.06	-0.08	0.25	-0.10	0.20	-0.01	-0.38	0.22	0.26	-0.27	0.28
Sex	0.06	1.00	-0.13	0.01	-0.20	0.12	0.07	-0.19	0.19	0.11	-0.15	0.31
ChestPainType	-0.08	-0.13	1.00	-0.02	0.07	-0.07	-0.07	0.29	-0.35	-0.18	0.21	-0.39
RestingBP	0.25	0.01	-0.02	1.00	0.10	0.07	0.02	-0.11	0.16	0.16	-0.08	0.11
Cholesterol	-0.10	-0.20	0.07	0.10	1.00	-0.26	-0.20	0.24	-0.03	0.05	0.11	-0.23
FastingBS	0.20	0.12	-0.07	0.07	-0.26	1.00	0.09	-0.13	0.06	0.05	-0.18	0.27
RestingECG	-0.01	0.07	-0.07	0.02	-0.20	0.09	1.00	-0.18	0.08	-0.02	-0.01	0.06
MaxHR	-0.38	-0.19	0.29	-0.11	0.24	-0.13	-0.18	1.00	-0.37	-0.16	0.34	-0.40
ExerciseAngina	0.22	0.19	-0.35	0.16	-0.03	0.06	0.08	-0.37	1.00	0.41	-0.43	0.49
Oldpeak	0.26	0.11	-0.18	0.16	0.05	0.05	-0.02	-0.16	0.41	1.00	-0.50	0.40
ST_Slope	-0.27	-0.15	0.21	-0.08	0.11	-0.18	-0.01	0.34	-0.43	-0.50	1.00	-0.56
HeartDisease	0.28	0.31	-0.39	0.11	-0.23	0.27	0.06	-0.40	0.49	0.40	-0.56	1.00

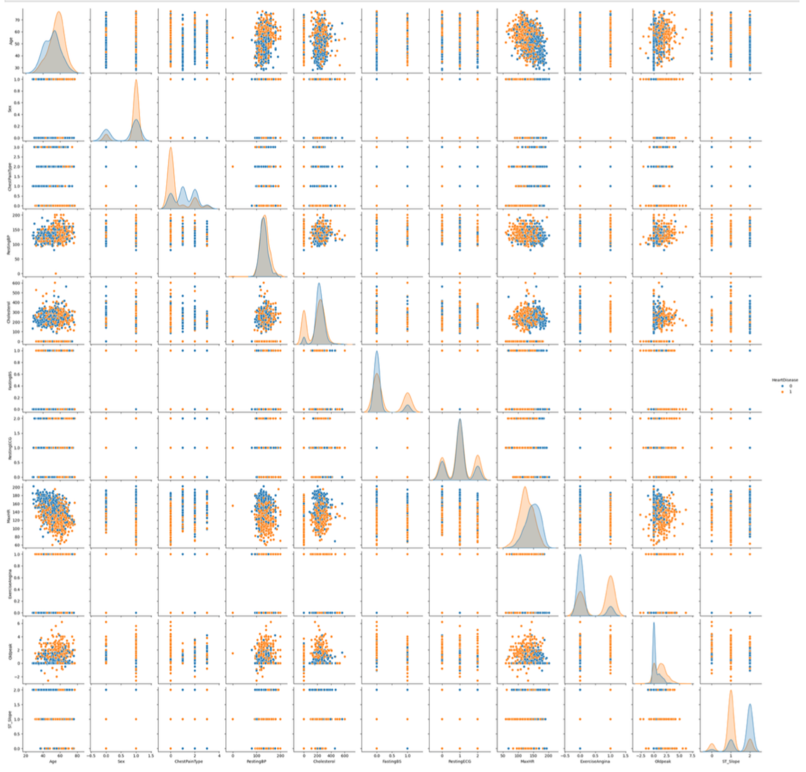
- **Histogram For all Features in the Dataset:**

```
df_encoded.hist(figsize=(12, 10))
plt.title('Histograms for Features')
plt.show()
```



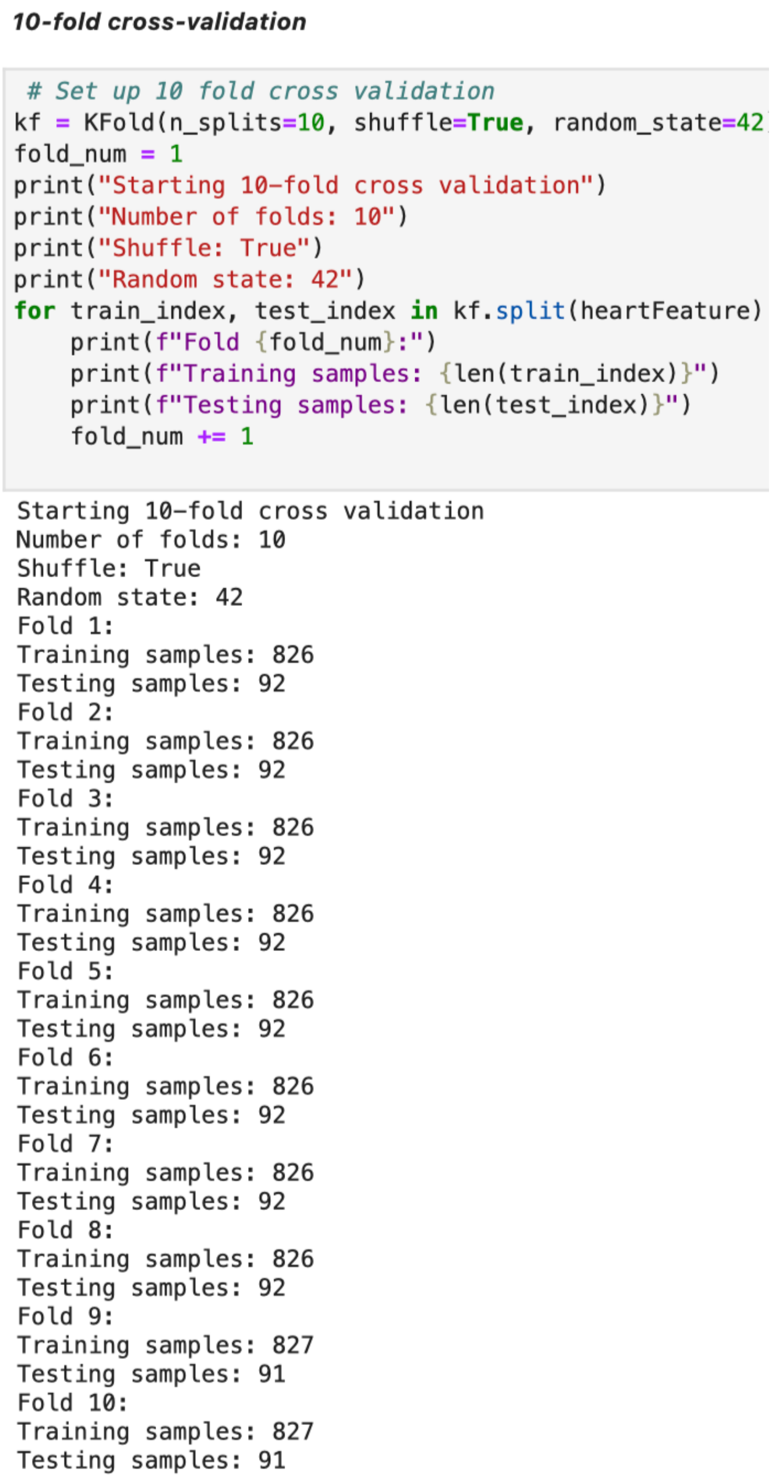
- **Pair Plot**

```
sns.pairplot(df_encoded, hue='HeartDisease')
plt.show()
```



Headings for Report Sections:

- **10-Fold Cross-Validation Implementation:**

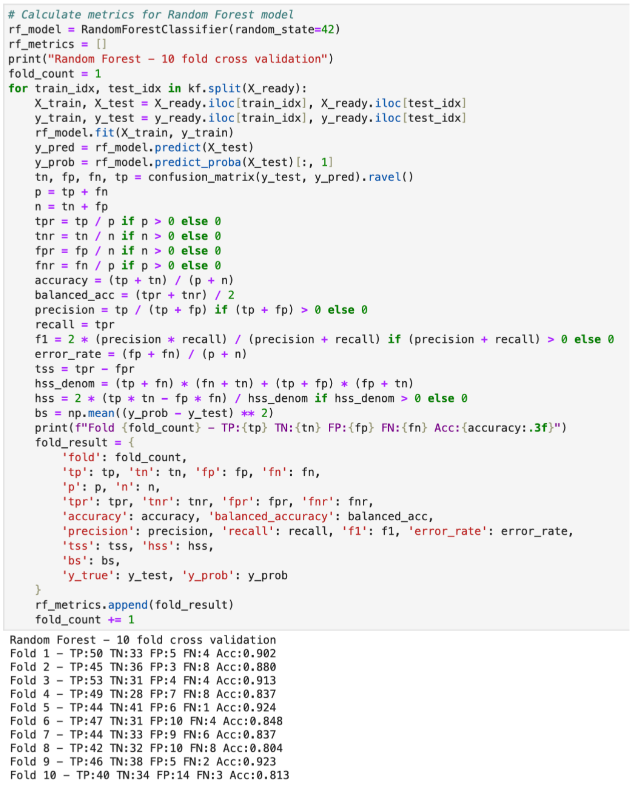


10-fold cross-validation

```
# Set up 10 fold cross validation
kf = KFold(n_splits=10, shuffle=True, random_state=42
fold_num = 1
print("Starting 10-fold cross validation")
print("Number of folds: 10")
print("Shuffle: True")
print("Random state: 42")
for train_index, test_index in kf.split(heartFeature)
    print(f"Fold {fold_num}:")
    print(f"Training samples: {len(train_index)}")
    print(f"Testing samples: {len(test_index)}")
    fold_num += 1
```

```
Starting 10-fold cross validation
Number of folds: 10
Shuffle: True
Random state: 42
Fold 1:
Training samples: 826
Testing samples: 92
Fold 2:
Training samples: 826
Testing samples: 92
Fold 3:
Training samples: 826
Testing samples: 92
Fold 4:
Training samples: 826
Testing samples: 92
Fold 5:
Training samples: 826
Testing samples: 92
Fold 6:
Training samples: 826
Testing samples: 92
Fold 7:
Training samples: 826
Testing samples: 92
Fold 8:
Training samples: 826
Testing samples: 92
Fold 9:
Training samples: 827
Testing samples: 91
Fold 10:
Training samples: 827
Testing samples: 91
```

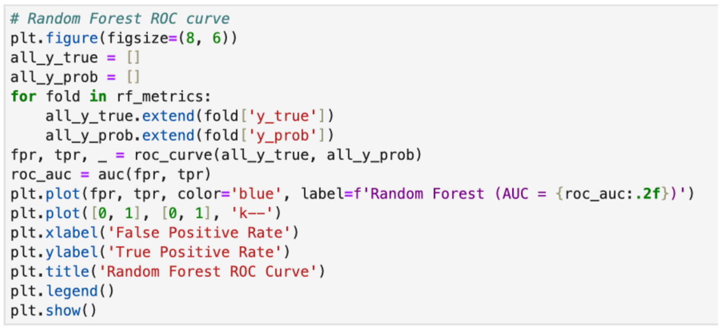
- ## Random Forest Classifier:



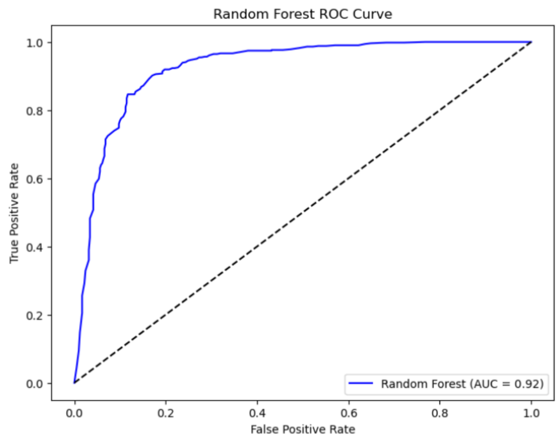
```
# Calculate metrics for Random Forest model
rf_model = RandomForestClassifier(random_state=42)
rf_metrics = []
print("Random Forest - 10 fold cross validation")
fold_count = 1
for train_idx, test_idx in kf.split(X_ready):
    X_train, X_test = X_ready.iloc[train_idx], X_ready.iloc[test_idx]
    y_train, y_test = y_ready.iloc[train_idx], y_ready.iloc[test_idx]
    rf_model.fit(X_train, y_train)
    y_pred = rf_model.predict(X_test)
    y_prob = rf_model.predict_proba(X_test)[:, 1]
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
    p = tp + fn
    n = tn + fp
    tpr = tp / p if p > 0 else 0
    tnr = tn / n if n > 0 else 0
    fpr = fp / n if n > 0 else 0
    fnr = fn / p if p > 0 else 0
    accuracy = (tp + tn) / (p + n)
    balanced_acc = (tpr + tnr) / 2
    precision = tp / (tp + fp) if (tp + fp) > 0 else 0
    recall = tpr
    f1 = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0
    error_rate = (fp + fn) / (p + n)
    tss = tpr - fpr
    hss_denom = (tp + fn) * (fn + tn) + (tp + fp) * (fp + tn)
    hss = 2 * (tp * tn - fp * fn) / hss_denom if hss_denom > 0 else 0
    bs = np.mean((y_prob - y_test) ** 2)
    print(f"Fold {fold_count} - TP:{tp} TN:{tn} FP:{fp} FN:{fn} Acc:{accuracy:.3f}")
    fold_result = {
        'fold': fold_count,
        'tp': tp, 'tn': tn, 'fp': fp, 'fn': fn,
        'p': p, 'n': n,
        'tpr': tpr, 'tnr': tnr, 'fpr': fpr, 'fnr': fnr,
        'accuracy': accuracy, 'balanced_accuracy': balanced_acc,
        'precision': precision, 'recall': recall, 'f1': f1, 'error_rate': error_rate,
        'tss': tss, 'hss': hss,
        'bs': bs,
        'y_true': y_test, 'y_prob': y_prob
    }
    rf_metrics.append(fold_result)
    fold_count += 1
```

```
Random Forest - 10 fold cross validation
Fold 1 - TP:50 TN:33 FP:5 FN:4 Acc:0.902
Fold 2 - TP:45 TN:36 FP:3 FN:8 Acc:0.880
Fold 3 - TP:53 TN:31 FP:4 FN:4 Acc:0.913
Fold 4 - TP:49 TN:28 FP:7 FN:8 Acc:0.837
Fold 5 - TP:44 TN:41 FP:6 FN:1 Acc:0.924
Fold 6 - TP:47 TN:31 FP:10 FN:4 Acc:0.848
Fold 7 - TP:44 TN:33 FP:9 FN:6 Acc:0.837
Fold 8 - TP:42 TN:32 FP:10 FN:8 Acc:0.804
Fold 9 - TP:46 TN:38 FP:5 FN:2 Acc:0.923
Fold 10 - TP:40 TN:34 FP:14 FN:3 Acc:0.813
```

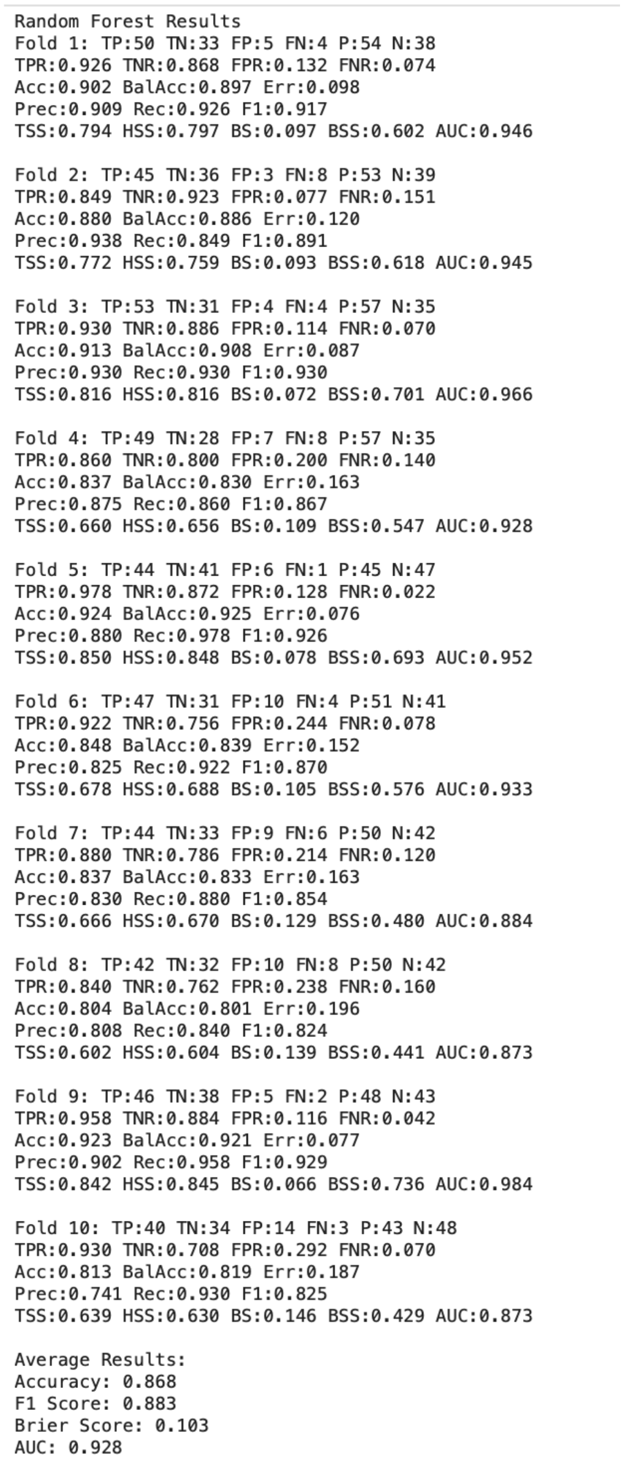
- ## Random Forest ROC Curve:



```
# Random Forest ROC curve
plt.figure(figsize=(8, 6))
all_y_true = []
all_y_prob = []
for fold in rf_metrics:
    all_y_true.extend(fold['y_true'])
    all_y_prob.extend(fold['y_prob'])
fpr, tpr, _ = roc_curve(all_y_true, all_y_prob)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='blue', label=f'Random Forest (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Random Forest ROC Curve')
plt.legend()
plt.show()
```

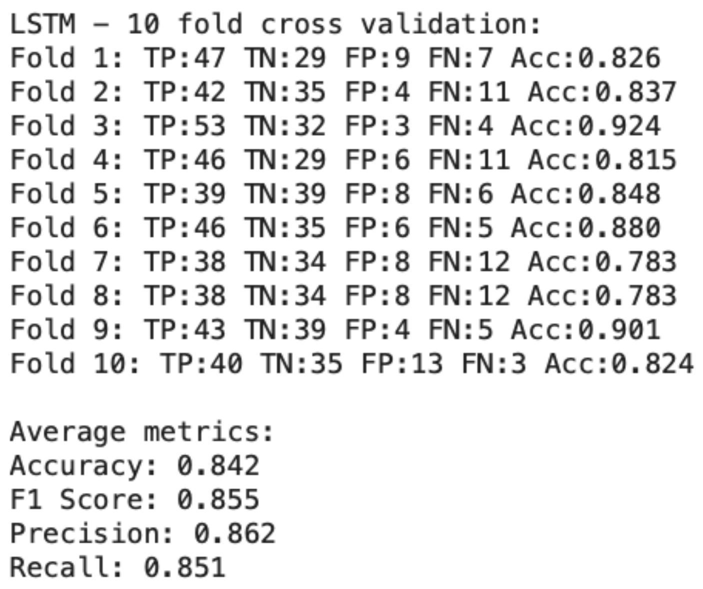


- **Random Forest Results**



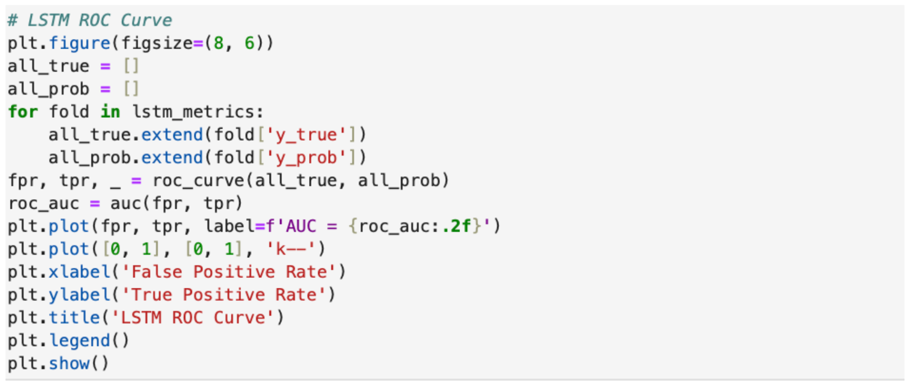
```
Random Forest Results
Fold 1: TP:50 TN:33 FP:5 FN:4 P:54 N:38
TPR:0.926 TNR:0.868 FPR:0.132 FNR:0.074
Acc:0.902 BalAcc:0.897 Err:0.098
Prec:0.909 Rec:0.926 F1:0.917
TSS:0.794 HSS:0.797 BS:0.097 BSS:0.602 AUC:0.946

Fold 2: TP:45 TN:36 FP:3 FN:8 P:53 N:39
TPR:0.849 TNR:0.923 FPR:0.077 FNR:0.151
Acc:0.880 BalAcc:0.886 Err:0.120
Prec:0.938 Rec:0.849 F1:0.891
TSS:0.772 HSS:0.759 BS:0.093 BSS:0.618 AUC:0.945

Fold 3: TP:53 TN:31 FP:4 FN:4 P:57 N:35
TPR:0.930 TNR:0.886 FPR:0.114 FNR:0.070
Acc:0.913 BalAcc:0.908 Err:0.087
Prec:0.930 Rec:0.930 F1:0.930
TSS:0.816 HSS:0.816 BS:0.072 BSS:0.701 AUC:0.966

Fold 4: TP:49 TN:28 FP:7 FN:8 P:57 N:35
TPR:0.860 TNR:0.800 FPR:0.200 FNR:0.140
Acc:0.837 BalAcc:0.830 Err:0.163
Prec:0.875 Rec:0.860 F1:0.867
TSS:0.660 HSS:0.656 BS:0.109 BSS:0.547 AUC:0.928

Fold 5: TP:44 TN:41 FP:6 FN:1 P:45 N:47
TPR:0.978 TNR:0.872 FPR:0.128 FNR:0.022
Acc:0.924 BalAcc:0.925 Err:0.076
Prec:0.880 Rec:0.978 F1:0.926
TSS:0.850 HSS:0.848 BS:0.078 BSS:0.693 AUC:0.952

Fold 6: TP:47 TN:31 FP:10 FN:4 P:51 N:41
TPR:0.922 TNR:0.756 FPR:0.244 FNR:0.078
Acc:0.848 BalAcc:0.839 Err:0.152
Prec:0.825 Rec:0.922 F1:0.870
TSS:0.678 HSS:0.688 BS:0.105 BSS:0.576 AUC:0.933

Fold 7: TP:44 TN:33 FP:9 FN:6 P:50 N:42
TPR:0.880 TNR:0.786 FPR:0.214 FNR:0.120
Acc:0.837 BalAcc:0.833 Err:0.163
Prec:0.830 Rec:0.880 F1:0.854
TSS:0.666 HSS:0.670 BS:0.129 BSS:0.480 AUC:0.884

Fold 8: TP:42 TN:32 FP:10 FN:8 P:50 N:42
TPR:0.840 TNR:0.762 FPR:0.238 FNR:0.160
Acc:0.804 BalAcc:0.801 Err:0.196
Prec:0.808 Rec:0.840 F1:0.824
TSS:0.602 HSS:0.604 BS:0.139 BSS:0.441 AUC:0.873

Fold 9: TP:46 TN:38 FP:5 FN:2 P:48 N:43
TPR:0.958 TNR:0.884 FPR:0.116 FNR:0.042
Acc:0.923 BalAcc:0.921 Err:0.077
Prec:0.902 Rec:0.958 F1:0.929
TSS:0.842 HSS:0.845 BS:0.066 BSS:0.736 AUC:0.984

Fold 10: TP:40 TN:34 FP:14 FN:3 P:43 N:48
TPR:0.930 TNR:0.708 FPR:0.292 FNR:0.070
Acc:0.813 BalAcc:0.819 Err:0.187
Prec:0.741 Rec:0.930 F1:0.825
TSS:0.639 HSS:0.630 BS:0.146 BSS:0.429 AUC:0.873

Average Results:
Accuracy: 0.868
F1 Score: 0.883
Brier Score: 0.103
AUC: 0.928
```

- **Long Short-Term Memory (LSTM):**

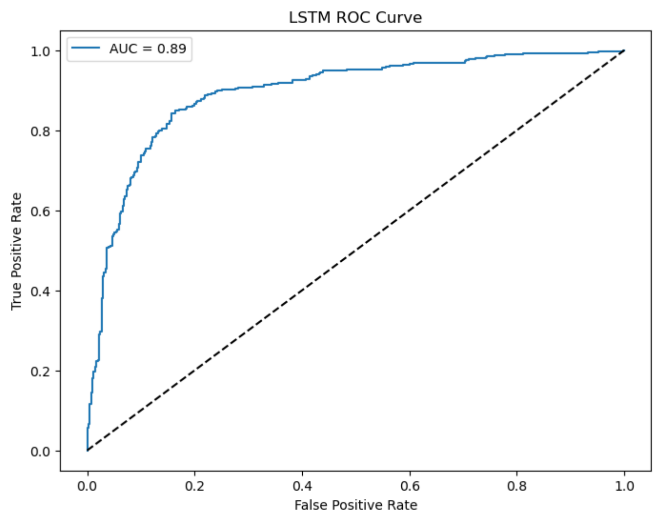


```
LSTM - 10 fold cross validation:
Fold 1: TP:47 TN:29 FP:9 FN:7 Acc:0.826
Fold 2: TP:42 TN:35 FP:4 FN:11 Acc:0.837
Fold 3: TP:53 TN:32 FP:3 FN:4 Acc:0.924
Fold 4: TP:46 TN:29 FP:6 FN:11 Acc:0.815
Fold 5: TP:39 TN:39 FP:8 FN:6 Acc:0.848
Fold 6: TP:46 TN:35 FP:6 FN:5 Acc:0.880
Fold 7: TP:38 TN:34 FP:8 FN:12 Acc:0.783
Fold 8: TP:38 TN:34 FP:8 FN:12 Acc:0.783
Fold 9: TP:43 TN:39 FP:4 FN:5 Acc:0.901
Fold 10: TP:40 TN:35 FP:13 FN:3 Acc:0.824

Average metrics:
Accuracy: 0.842
F1 Score: 0.855
Precision: 0.862
Recall: 0.851
```

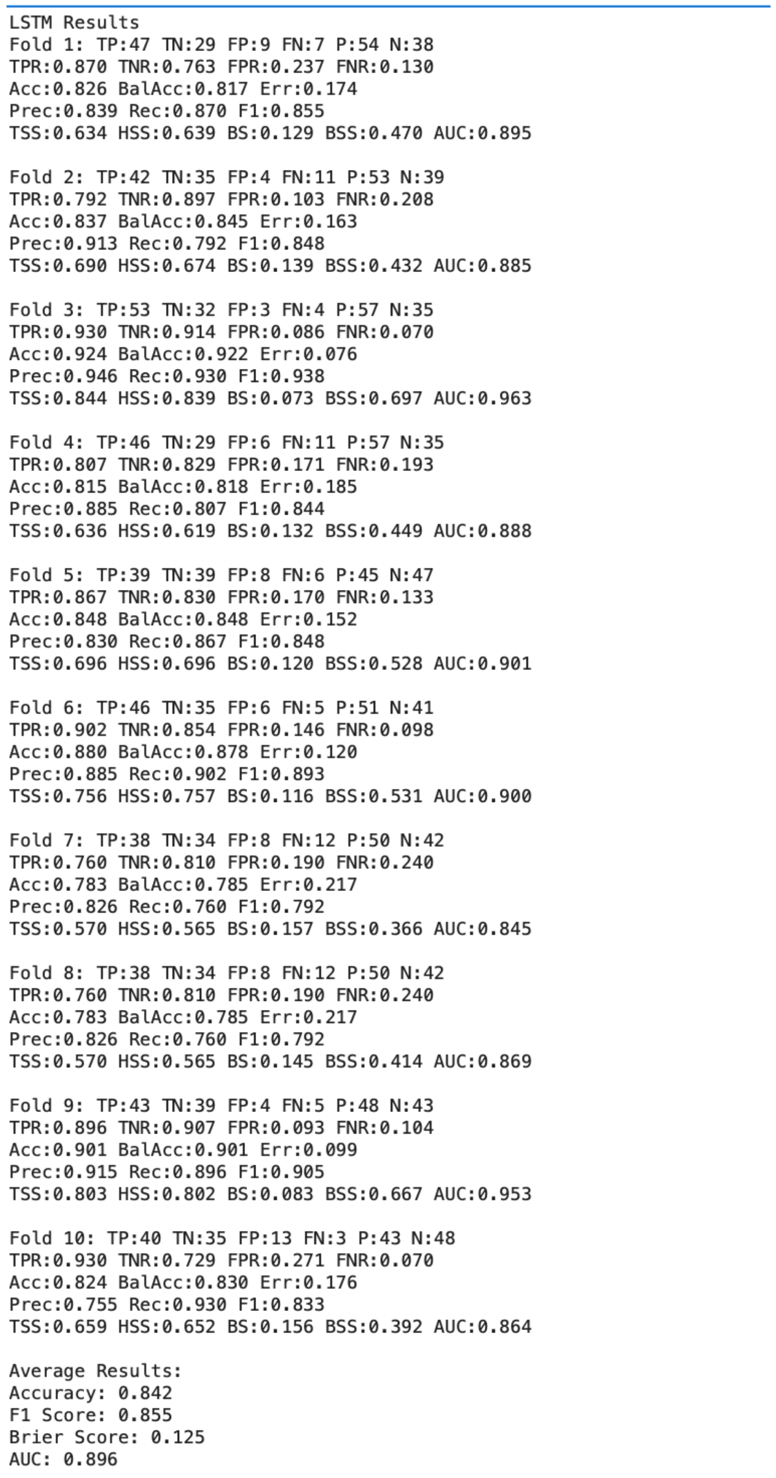
- **LSTM ROC Curve:**



```python
# LSTM ROC Curve
plt.figure(figsize=(8, 6))
all_true = []
all_prob = []
for fold in lstm_metrics:
    all_true.extend(fold['y_true'])
    all_prob.extend(fold['y_prob'])
fpr, tpr, _ = roc_curve(all_true, all_prob)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, label=f'AUC = {roc_auc:.2f}')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('LSTM ROC Curve')
plt.legend()
plt.show()
```



- ## LSTM Results:



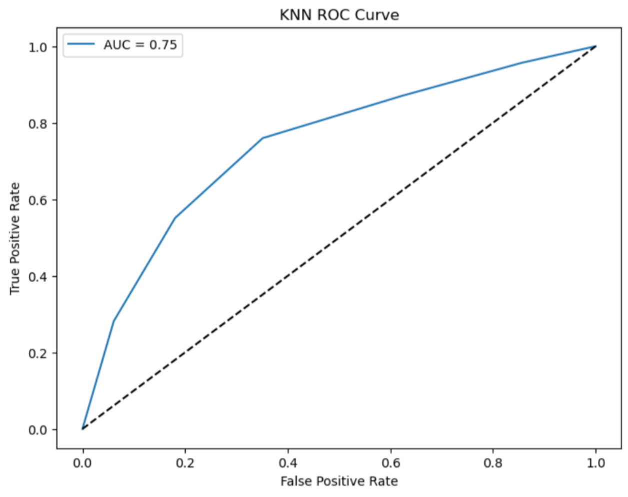
```
LSTM Results
Fold 1: TP:47 TN:29 FP:9 FN:7 P:54 N:38
TPR:0.870 TNR:0.763 FPR:0.237 FNR:0.130
Acc:0.826 BalAcc:0.817 Err:0.174
Prec:0.839 Rec:0.870 F1:0.855
TSS:0.634 HSS:0.639 BS:0.129 BSS:0.470 AUC:0.895

Fold 2: TP:42 TN:35 FP:4 FN:11 P:53 N:39
TPR:0.792 TNR:0.897 FPR:0.103 FNR:0.208
Acc:0.837 BalAcc:0.845 Err:0.163
Prec:0.913 Rec:0.792 F1:0.848
TSS:0.690 HSS:0.674 BS:0.139 BSS:0.432 AUC:0.885

Fold 3: TP:53 TN:32 FP:3 FN:4 P:57 N:35
TPR:0.930 TNR:0.914 FPR:0.086 FNR:0.070
Acc:0.924 BalAcc:0.922 Err:0.076
Prec:0.946 Rec:0.930 F1:0.938
TSS:0.844 HSS:0.839 BS:0.073 BSS:0.697 AUC:0.963

Fold 4: TP:46 TN:29 FP:6 FN:11 P:57 N:35
TPR:0.807 TNR:0.829 FPR:0.171 FNR:0.193
Acc:0.815 BalAcc:0.818 Err:0.185
Prec:0.885 Rec:0.807 F1:0.844
TSS:0.636 HSS:0.619 BS:0.132 BSS:0.449 AUC:0.888

Fold 5: TP:39 TN:39 FP:8 FN:6 P:45 N:47
TPR:0.867 TNR:0.830 FPR:0.170 FNR:0.133
Acc:0.848 BalAcc:0.848 Err:0.152
Prec:0.830 Rec:0.867 F1:0.848
TSS:0.696 HSS:0.696 BS:0.120 BSS:0.528 AUC:0.901

Fold 6: TP:46 TN:35 FP:6 FN:5 P:51 N:41
TPR:0.902 TNR:0.854 FPR:0.146 FNR:0.098
Acc:0.880 BalAcc:0.878 Err:0.120
Prec:0.885 Rec:0.902 F1:0.893
TSS:0.756 HSS:0.757 BS:0.116 BSS:0.531 AUC:0.900

Fold 7: TP:38 TN:34 FP:8 FN:12 P:50 N:42
TPR:0.760 TNR:0.810 FPR:0.190 FNR:0.240
Acc:0.783 BalAcc:0.785 Err:0.217
Prec:0.826 Rec:0.760 F1:0.792
TSS:0.570 HSS:0.565 BS:0.157 BSS:0.366 AUC:0.845

Fold 8: TP:38 TN:34 FP:8 FN:12 P:50 N:42
TPR:0.760 TNR:0.810 FPR:0.190 FNR:0.240
Acc:0.783 BalAcc:0.785 Err:0.217
Prec:0.826 Rec:0.760 F1:0.792
TSS:0.570 HSS:0.565 BS:0.145 BSS:0.414 AUC:0.869

Fold 9: TP:43 TN:39 FP:4 FN:5 P:48 N:43
TPR:0.896 TNR:0.907 FPR:0.093 FNR:0.104
Acc:0.901 BalAcc:0.901 Err:0.099
Prec:0.915 Rec:0.896 F1:0.905
TSS:0.803 HSS:0.802 BS:0.083 BSS:0.667 AUC:0.953

Fold 10: TP:40 TN:35 FP:13 FN:3 P:43 N:48
TPR:0.930 TNR:0.729 FPR:0.271 FNR:0.070
Acc:0.824 BalAcc:0.830 Err:0.176
Prec:0.755 Rec:0.930 F1:0.833
TSS:0.659 HSS:0.652 BS:0.156 BSS:0.392 AUC:0.864

Average Results:
Accuracy: 0.842
F1 Score: 0.855
Brier Score: 0.125
AUC: 0.896
```

- **K-Nearest Neighbors (KNN):**

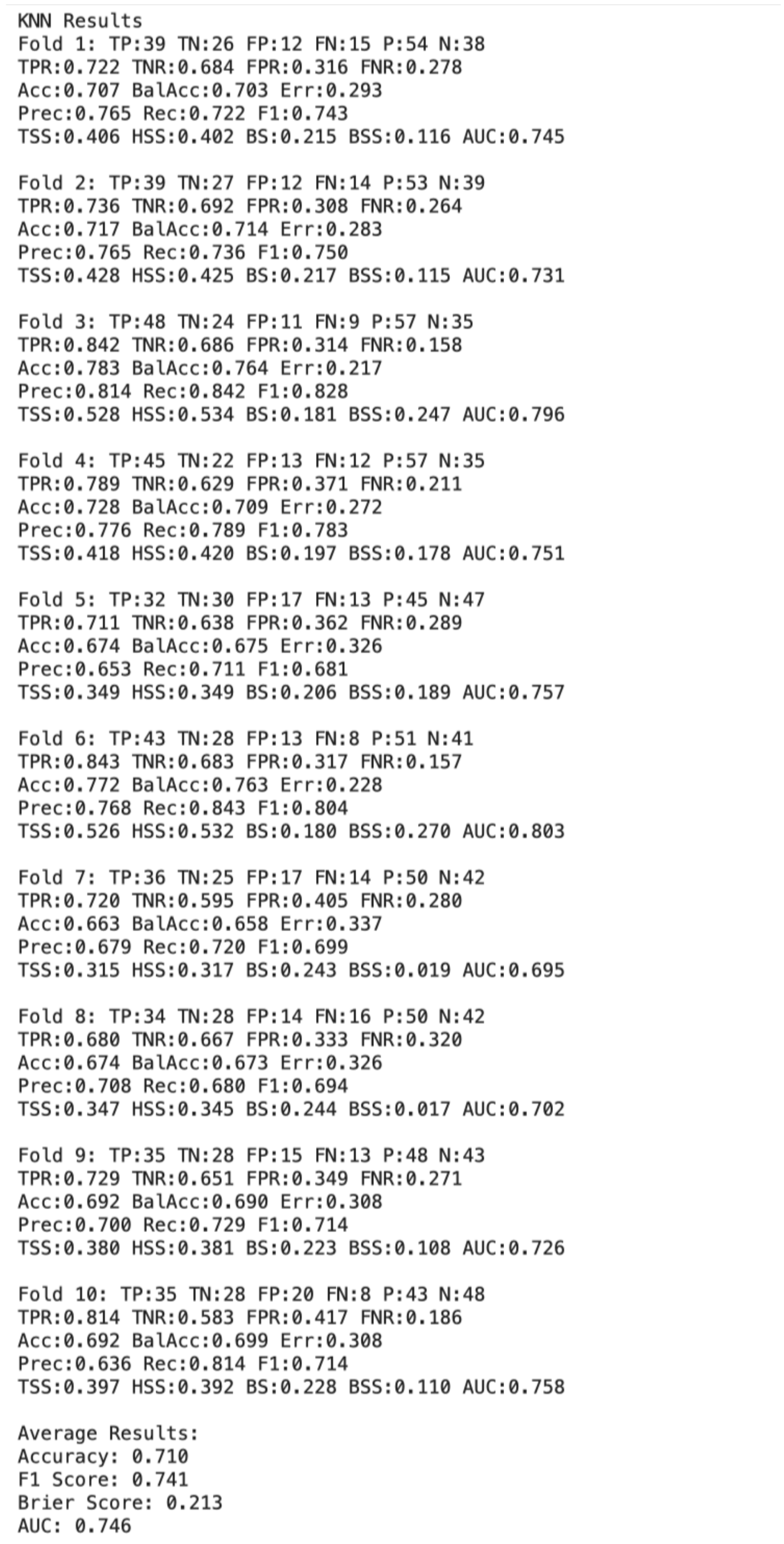
```
KNN – 10 fold cross validation
Fold 1
TP:39 TN:26 FP:12 FN:15 Acc:0.707
Fold 2
TP:39 TN:27 FP:12 FN:14 Acc:0.717
Fold 3
TP:48 TN:24 FP:11 FN:9 Acc:0.783
Fold 4
TP:45 TN:22 FP:13 FN:12 Acc:0.728
Fold 5
TP:32 TN:30 FP:17 FN:13 Acc:0.674
Fold 6
TP:43 TN:28 FP:13 FN:8 Acc:0.772
Fold 7
TP:36 TN:25 FP:17 FN:14 Acc:0.663
Fold 8
TP:34 TN:28 FP:14 FN:16 Acc:0.674
Fold 9
TP:35 TN:28 FP:15 FN:13 Acc:0.692
Fold 10
TP:35 TN:28 FP:20 FN:8 Acc:0.692
```

- **KNN ROC Curve:**

```python
# KNN ROC Curve
plt.figure(figsize=(8, 6))
all_true = []
all_prob = []
for fold in knn_metrics:
    all_true.extend(fold['y_true'])
    all_prob.extend(fold['y_prob'])
fpr, tpr, _ = roc_curve(all_true, all_prob)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, label=f'AUC = {roc_auc:.2f}')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('KNN ROC Curve')
plt.legend()
plt.show()
```



- **KNN Results:**



```
KNN Results
Fold 1: TP:39 TN:26 FP:12 FN:15 P:54 N:38
TPR:0.722 TNR:0.684 FPR:0.316 FNR:0.278
Acc:0.707 BalAcc:0.703 Err:0.293
Prec:0.765 Rec:0.722 F1:0.743
TSS:0.406 HSS:0.402 BS:0.215 BSS:0.116 AUC:0.745

Fold 2: TP:39 TN:27 FP:12 FN:14 P:53 N:39
TPR:0.736 TNR:0.692 FPR:0.308 FNR:0.264
Acc:0.717 BalAcc:0.714 Err:0.283
Prec:0.765 Rec:0.736 F1:0.750
TSS:0.428 HSS:0.425 BS:0.217 BSS:0.115 AUC:0.731

Fold 3: TP:48 TN:24 FP:11 FN:9 P:57 N:35
TPR:0.842 TNR:0.686 FPR:0.314 FNR:0.158
Acc:0.783 BalAcc:0.764 Err:0.217
Prec:0.814 Rec:0.842 F1:0.828
TSS:0.528 HSS:0.534 BS:0.181 BSS:0.247 AUC:0.796

Fold 4: TP:45 TN:22 FP:13 FN:12 P:57 N:35
TPR:0.789 TNR:0.629 FPR:0.371 FNR:0.211
Acc:0.728 BalAcc:0.709 Err:0.272
Prec:0.776 Rec:0.789 F1:0.783
TSS:0.418 HSS:0.420 BS:0.197 BSS:0.178 AUC:0.751

Fold 5: TP:32 TN:30 FP:17 FN:13 P:45 N:47
TPR:0.711 TNR:0.638 FPR:0.362 FNR:0.289
Acc:0.674 BalAcc:0.675 Err:0.326
Prec:0.653 Rec:0.711 F1:0.681
TSS:0.349 HSS:0.349 BS:0.206 BSS:0.189 AUC:0.757

Fold 6: TP:43 TN:28 FP:13 FN:8 P:51 N:41
TPR:0.843 TNR:0.683 FPR:0.317 FNR:0.157
Acc:0.772 BalAcc:0.763 Err:0.228
Prec:0.768 Rec:0.843 F1:0.804
TSS:0.526 HSS:0.532 BS:0.180 BSS:0.270 AUC:0.803

Fold 7: TP:36 TN:25 FP:17 FN:14 P:50 N:42
TPR:0.720 TNR:0.595 FPR:0.405 FNR:0.280
Acc:0.663 BalAcc:0.658 Err:0.337
Prec:0.679 Rec:0.720 F1:0.699
TSS:0.315 HSS:0.317 BS:0.243 BSS:0.019 AUC:0.695

Fold 8: TP:34 TN:28 FP:14 FN:16 P:50 N:42
TPR:0.680 TNR:0.667 FPR:0.333 FNR:0.320
Acc:0.674 BalAcc:0.673 Err:0.326
Prec:0.708 Rec:0.680 F1:0.694
TSS:0.347 HSS:0.345 BS:0.244 BSS:0.017 AUC:0.702

Fold 9: TP:35 TN:28 FP:15 FN:13 P:48 N:43
TPR:0.729 TNR:0.651 FPR:0.349 FNR:0.271
Acc:0.692 BalAcc:0.690 Err:0.308
Prec:0.700 Rec:0.729 F1:0.714
TSS:0.380 HSS:0.381 BS:0.223 BSS:0.108 AUC:0.726

Fold 10: TP:35 TN:28 FP:20 FN:8 P:43 N:48
TPR:0.814 TNR:0.583 FPR:0.417 FNR:0.186
Acc:0.692 BalAcc:0.699 Err:0.308
Prec:0.636 Rec:0.814 F1:0.714
TSS:0.397 HSS:0.392 BS:0.228 BSS:0.110 AUC:0.758

Average Results:
Accuracy: 0.710
F1 Score: 0.741
Brier Score: 0.213
AUC: 0.746
```

Comparing Average Performance Metrics for Random Forest, Long Short-Term Memory and K-Nearest Neighbors:

```python
# Model comparison summary
print("Average Performance Metrics:")
print("=" * 50)
metrics = ['accuracy', 'balanced_accuracy', 'precision', 'recall', 'f1', 'tss', 'hss', 'bs']
print(f"{'Metrics':20} {'RF':8} {'LSTM':8} {'KNN':8}")
print("-" * 50)
for metric in metrics:
    rf_avg = rf_df[metric].mean()
    lstm_avg = lstm_df[metric].mean()
    knn_avg = knn_df[metric].mean()
    print(f"{metric:20} {rf_avg:.3f}   {lstm_avg:.3f}   {knn_avg:.3f}")
```

```
Average Performance Metrics:
==================================================
Metrics              RF       LSTM     KNN
--------------------------------------------------
accuracy             0.868   0.842   0.710
balanced_accuracy    0.866   0.843   0.705
precision            0.864   0.862   0.726
recall               0.907   0.851   0.759
f1                   0.883   0.855   0.741
tss                  0.732   0.686   0.410
hss                  0.731   0.681   0.410
bs                   0.103   0.125   0.213
```

```
Accuracy scores:
Random Forest: 0.8681915910176781
LSTM: 0.842092689918777
KNN: 0.7102006688963212
Random Forest performed best on this dataset
```

Conclusion:

This project successfully implemented and compared three binary classification algorithms for heart failure prediction. Through 10-fold cross-validation and comprehensive performance evaluation, the models demonstrated varying levels of effectiveness in classifying patients with heart disease.

Random Forest clearly emerged as the best performing algorithm, achieving the highest scores across all key metrics: accuracy (0.868), balanced accuracy (0.866), F1 score (0.883), and TSS (0.732). Its strong performance across multiple evaluation criteria confirms its robustness for medical classification tasks.

LSTM provided competitive results with accuracy of 0.842 and well-balanced precision (0.862) and recall (0.851), demonstrating that deep learning approaches can be effective even on structured medical data.

KNN, while providing a useful baseline, showed lower performance with accuracy of 0.710, indicating that simpler distance-based methods may be less suitable for this complex medical prediction task.

The comprehensive evaluation using multiple metrics provided valuable insights into each model's characteristics, with Random Forest demonstrating superior overall performance for heart failure prediction. The project successfully met all requirements, implementing three distinct algorithms with thorough cross-validation and manual metrics calculation.

System Requirements and Versions

Python Version: 3.12.4

Package Versions:

- pandas: 2.2.2
- numpy: 1.26.0
- matplotlib: 3.8.4
- seaborn: 0.13.2
- scikit-learn: 1.5.2
- tensorflow: 2.17.0

All required libraries were successfully installed and used for data processing, model implementation, and result visualization throughout the project.

Link to GitHub: https://github.com/prabhathv07/Vipparthi_Prabhath_finalproject