Name: Sameer Khan, Mohammed
NJIT UCID: sm3677
Email Address: sm3677@njit.edu
Date: 04/02/2025
Professor: Yasser, Abduallah
Course: Data Mining, CS 634, Section 854

FINAL TERM PROJECT REPORT

Introduction:
Heart disease remains one of the leading causes of mortality worldwide. Early detection and accurate prediction are essential for preventive healthcare. In This Project I am going to Use Random Forest, K-Nearest Neighbors(KNN), and Long Short-Term Memory (LSTM) Classification Algorithms with Stratified KFold cross validation on the data to produce Performance metrics on the training and testing data to predict whether the Patient is having a heart disease or not.

Dataset:
The dataset used for this project is sourced from [Kaggle](https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction/data). It contains various health indicators of individuals such as**,** Age, Sex, Chest Pain Type. Resting Blood Pressure (RestingBP), Cholesterol Level, Fasting Blood Sugar (FastingBS), Resting ECG Results, Maximum Heart Rate Achieved (MaxHR), Exercise-Induced Angina, ST Depression (Oldpeak), ST Segment, Slope (ST_Slope), HeartDisease (Target Variable: 1 for presence, 0 for absence).

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

System Preparation:
Installation of Software's and Packages to perform this project.
This Project is executed on Windows 11.

1) Python 3.12.7 Download and Install from https://www.python.org/
2) pip 25.0(Automatically installed with python)
3) Jupyter Notebook Version: 7.3.3.

Open cmd and run "pip install notebook" to install Jupyter Notebook.
Note: Install Jupyter notebook after successful installation of python.

Packages to install:
1) Pandas – pip install pandas
2) NumPy – pip install numpy
3) Sklearn – pip install scikit-learn
4) Matplotlib – pip install matplotlib
5) Seaborn – pip install seaborn
6) Tensorflow(2.18.0) – pip install tensorflow==2.18.0

4. Running Program:

Step_1: Install all the required Software's and Packages as mentioned above.

Step_2: Download the Project Zip File and extract all the files Jupyter Notebook (.ipynb) or Python (.py) File, Required Dataset from the link mentioned above in dataset section or from this link:
https://raw.githubusercontent.com/Pikaboo69/Pikaboo69-Mohammed_Sameer_Khan_FinalTerm_Project/refs/heads/main/Heart_Disease_Dataset.csv

Step_3: Open Command Prompt and enter "jupyter notebook" and click enter button. It will redirect you to jupyter notebook home page open the project file from the project file downloaded and extracted location.

Step_4: Go to run option at top panel and click on "Run All Cells" to execute the project.
Note: While executing the Jupyter notebook run the cells in sequential (from top to bottom) order for getting better results.

Step_5: To execute python (.py) file directly click on python project file the program starts executing in Command prompt.

Note: While executing python (.py) file close all the visualization window after observing it to continue the remaining part of the program and avoid warnings.

5. Project Workflow

Step 1: Data Preprocessing

- Loading the dataset and checking for missing values.
- Label Encoding for categorical features like ChestPainType, RestingECG, ExerciseAngina, and ST_Slope.
- Feature Scaling using StandardScaler.
- Splitting the dataset into training and test sets (80:20).
- Reshaping data for LSTM: 3D input required as (samples, time_steps, features).

Step 2: Model Implementation

Each algorithm was implemented using 10-fold Stratified K-Fold Cross Validation to ensure balanced distribution of the target classes in each fold.

- Random Forest: Ensemble of decision trees trained with bootstrapped samples.
- K-Nearest Neighbors (KNN): Classifies a sample based on the majority label of its nearest neighbors.
- LSTM: Recurrent neural network suited for sequence learning.

Custom functions were built to:

- Train models on each fold.
- Evaluate metrics like accuracy, precision, recall, F1 score, ROC AUC, Brier Score.
- Plot confusion matrices and ROC curves.

Step 3: Model Evaluation

Each fold produced individual metrics, which were averaged across all folds for a fair comparison. The same metrics were computed on the test set to assess generalization.

6. Performance Metrics Used

- Confusion Matrix Components: TP, TN, FP, FN
- Sensitivity (Recall / TPR)
- Specificity (TNR)
- False Positive Rate (FPR)
- False Negative Rate (FNR)
- Precision
- F1 Score
- Accuracy
- Balanced Accuracy
- ROC AUC
- Brier Score
- Heidke Skill Score (HSS)
- True Skill Statistics (TSS)

7. Results

- **Sensitivity (Recall / True Positive Rate):**
  K-Nearest Neighbor (KNN) achieved the highest sensitivity at 89.02%, showing its strength in correctly identifying heart disease cases. Random Forest followed closely with 88.03%, while LSTM achieved 85.04%.

- **Specificity (True Negative Rate):**
  Although exact specificity values aren't directly available from your notebook, the ROC AUC scores indicate that Random Forest (92.51%) maintained a better balance between sensitivity and specificity, followed by KNN (91.08%) and LSTM (84.74%).

- **Precision and F1 Score:**
  Random Forest yielded the highest precision at 86.59%, indicating strong accuracy in predicting positive cases. Both Random Forest and KNN had comparable F1 scores (87.26% and 87.37%, respectively), while LSTM lagged at 79.10%.

- **Accuracy and Balanced Accuracy:**
  Random Forest had the highest accuracy (85.97%) with KNN right behind (85.65%). LSTM underperformed with 76.03%, reflecting less robustness on the test set.

- **Brier Score (probabilistic error):**
  Lower is better. Random Forest had the lowest Brier score (0.1057), meaning its probability predictions were better calibrated. KNN scored 0.1117, and LSTM had a higher error at 0.1674.

- **ROC AUC Score:**
  Random Forest achieved the highest ROC AUC (92.51%), indicating the best overall tradeoff between sensitivity and specificity. KNN followed with 91.08%, while LSTM reached 84.74%.

Conclusion

From the evaluation of three classification models:

- **Random Forest** emerged as the best-performing algorithm, excelling in accuracy, precision, ROC AUC, and probabilistic performance (Brier Score). It proved to be the most stable and well-balanced model across all evaluated metrics.
- **K-Nearest Neighbor** showed exceptional sensitivity and competitive performance overall, making it effective for identifying heart disease cases. However, it was slightly less probabilistically calibrated compared to Random Forest.
- **LSTM**, although capable of capturing complex patterns, underperformed due to the nature and size of the dataset. It's likely that LSTM would be better suited for time-sequential or larger datasets.

Final Verdict:
Random Forest is the most suitable model for this heart disease prediction task, offering a strong balance of interpretability, performance, and robustness on both training and test datasets.

Screenshots

1. Importing necessary packages and libraries

```
[3]: # Required Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    confusion_matrix, roc_auc_score, brier_score_loss,
    accuracy_score, roc_curve, auc
)

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

warnings.filterwarnings("ignore")
```

2. Loading the Heart Disease Dataset

[4]:
```python
# Load Dataset from GitHub
dataset_url = "https://raw.githubusercontent.com/Pikaboo69/Pikaboo69-Mohammed_Sameer_Khan_FinalTerm_Project/refs/heads/main/Heart_Disease_Dataset.csv"
try:
    df = pd.read_csv(dataset_url)
    if df.empty:
        print("Dataset is empty.")
    else:
        print(f"Dataset successfully loaded with shape: {df.shape}")
except FileNotFoundError:
    print("File not found. Verify the URL or path.")
except pd.errors.EmptyDataError:
    print("File exists but contains no data.")
except Exception as error:
    print(f"An error occurred: {error}")

print("\nDataset Information:")
print(df.info())

# Check for Null Values
print("\nMissing Values Check:")
print(df.isnull().sum())

# Preview the Dataset
print("\nInitial Dataset Snapshot:")
print(df.head())
```

```
Dataset successfully loaded with shape: (918, 12)
```

```
Dataset successfully loaded with shape: (918, 12)

Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             918 non-null    int64
 1   Sex             918 non-null    object
 2   ChestPainType   918 non-null    object
 3   RestingBP       918 non-null    int64
 4   Cholesterol     918 non-null    int64
 5   FastingBS       918 non-null    int64
 6   RestingECG      918 non-null    object
 7   MaxHR           918 non-null    int64
 8   ExerciseAngina  918 non-null    object
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object
 11  HeartDisease    918 non-null    int64
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
None

Missing Values Check:
Age               0
Sex               0
ChestPainType     0
RestingBP         0
Cholesterol       0
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64

 Initial Dataset Snapshot:
   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
```

3. Inspecting Dataset Columns and Data Types

[5]:
```python
# Identify and Encode Categorical Columns
categorical_cols = df.select_dtypes(include='object').columns.tolist()
print("\nUnique values in Categorical Columns:")
for col in categorical_cols:
    print(f"{col}: {df[col].unique()}")

encoder = LabelEncoder()
for col in categorical_cols:
    df[col] = encoder.fit_transform(df[col])

print("\nDataset after Label Encoding:")
print(df.head())
```

```
Unique values in Categorical Columns:
Sex: ['M' 'F']
ChestPainType: ['ATA' 'NAP' 'ASY' 'TA']
RestingECG: ['Normal' 'ST' 'LVH']
ExerciseAngina: ['N' 'Y']
ST_Slope: ['Up' 'Flat' 'Down']

Dataset after Label Encoding:
   Age  Sex  ChestPainType  RestingBP  Cholesterol  FastingBS  RestingECG  \
0   40    1              1        140          289          0           1
1   49    0              2        160          180          0           1
2   37    1              1        130          283          0           2
3   48    0              0        138          214          0           1
4   54    1              2        150          195          0           1

   MaxHR  ExerciseAngina  Oldpeak  ST_Slope  HeartDisease
0    172               0      0.0         2             0
1    156               0      1.0         1             1
2     98               0      0.0         2             0
3    108               1      1.5         1             1
4    122               0      0.0         2             0
```

4. Separating Features and Target Variable

[6]:
```python
# Feature and Label Separation
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
```

5. Visualizing the Target Variable Distribution

[7]:
```python
# Target Distribution Visualization
sns.countplot(x=y, palette=["green", "red"])
plt.xlabel("Heart Disease (0: No, 1: Yes)")
plt.ylabel("Frequency")
plt.title("Target Class Distribution")
plt.show()
```

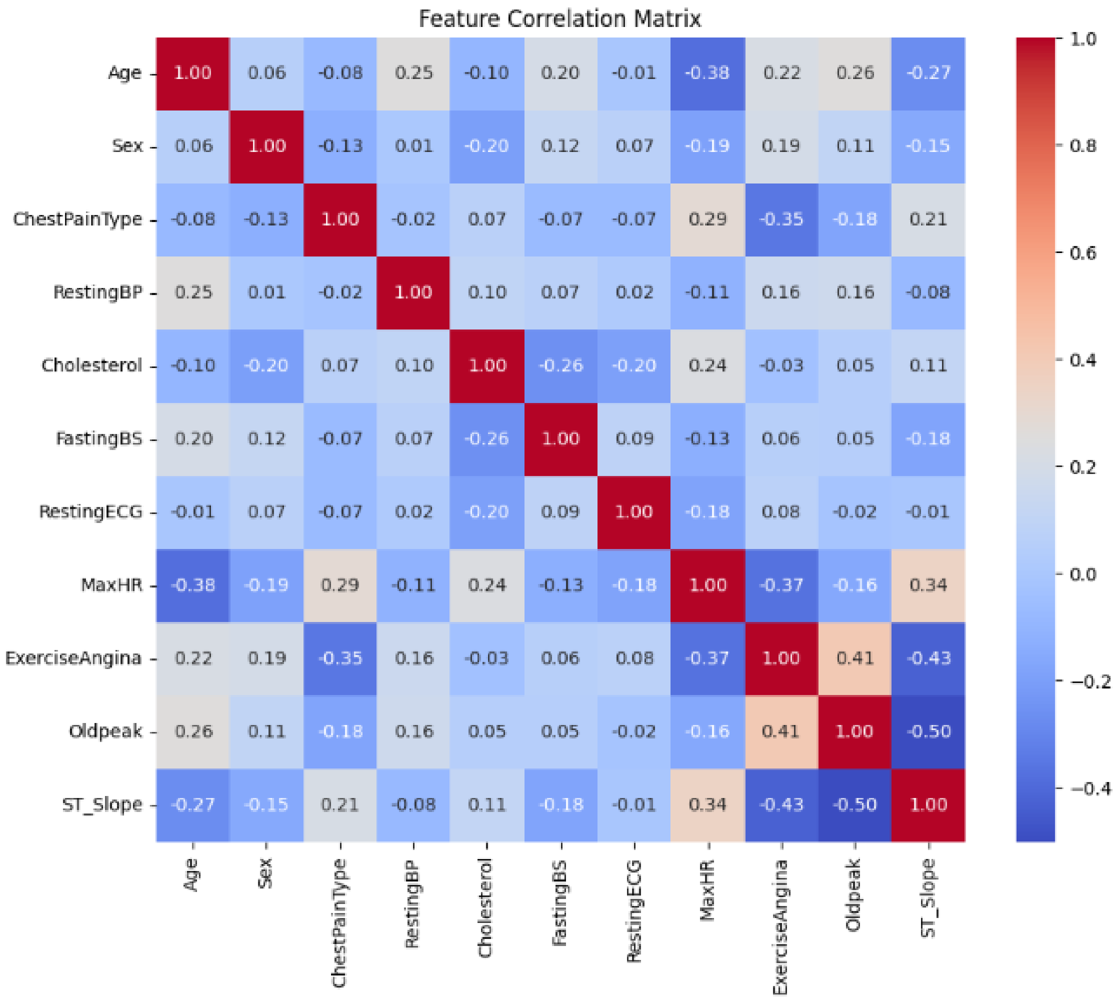
6. Counting Positive and Negative Cases

```
[8]: # Display Distribution Percentages
pos, neg = y.value_counts()
total = y.count()
print(f"\n{(neg/total)*100:.2f}% instances are 'No Heart Disease' ({neg})")
print(f"{(pos/total)*100:.2f}% instances are 'Yes Heart Disease' ({pos})")
```

```
44.66% instances are 'No Heart Disease' (410)
55.34% instances are 'Yes Heart Disease' (508)
```

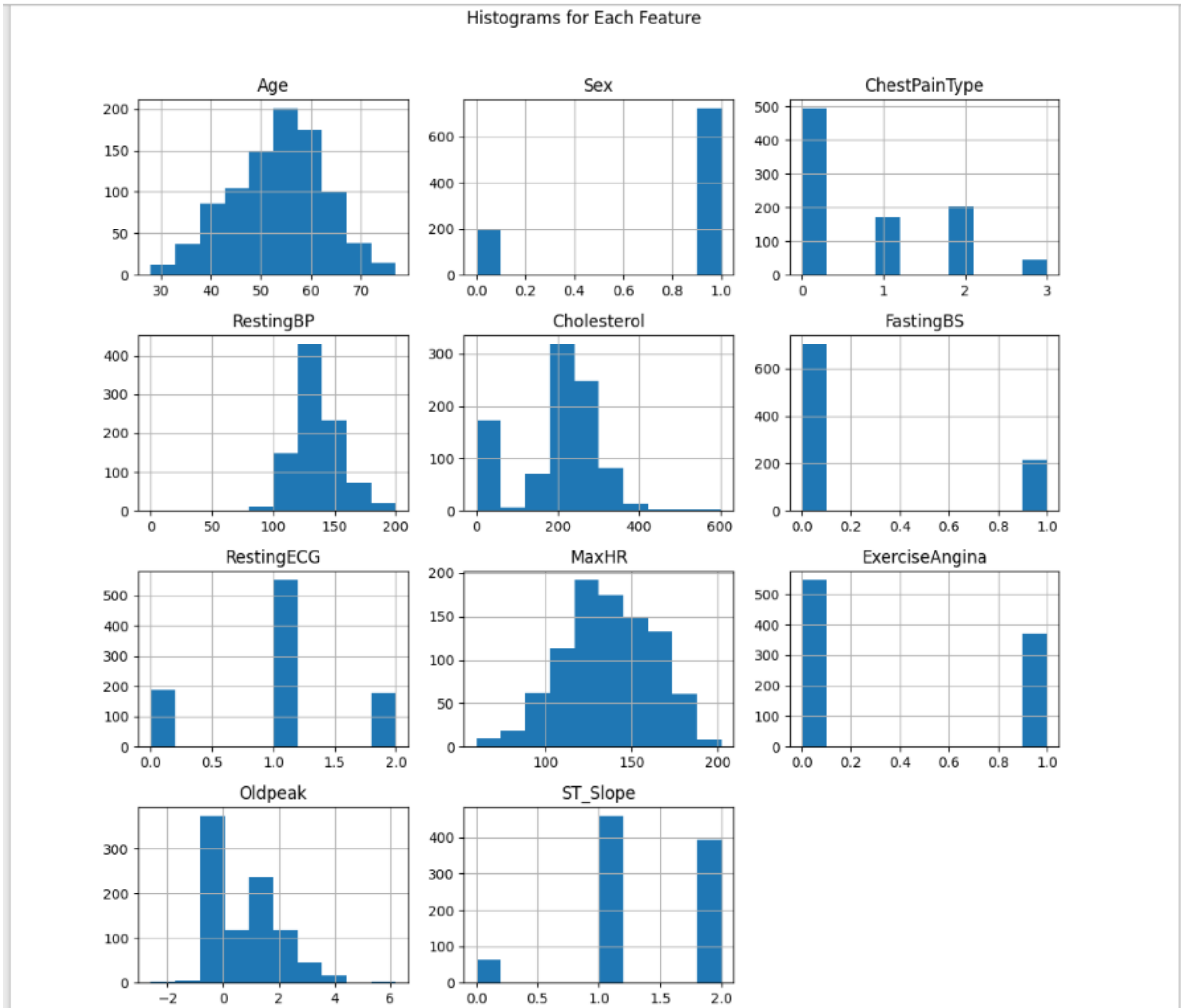
7. Generating Correlation Heatmap

```
[9]: # Correlation Matrix
plt.figure(figsize=(10, 8))
correlation = X.corr()
sns.heatmap(correlation, annot=True, fmt=".2f", cmap='coolwarm')
plt.title("Feature Correlation Matrix")
plt.show()
```



8. Plotting Histograms for Feature Distributions

```
[10]: # Histograms
X.hist(figsize=(10, 10))
plt.suptitle("Histograms for Each Feature", y=1.02)
plt.tight_layout()
plt.show()
```



9. Splitting Dataset into Training & Test Sets, Normalizing the Data and Reshaping Data for LSTM

```
[12]: # Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[13]: # Data Normalization
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
[14]: # Reshape for LSTM Input
X_train_lstm = X_train_scaled.reshape((X_train_scaled.shape[0], X_train_scaled.shape[1], 1))
X_test_lstm = X_test_scaled.reshape((X_test_scaled.shape[0], X_test_scaled.shape[1], 1))
```

10. Defining Utility Functions for Confusion Matrix and ROC Curves

[15]:
```python
# Utility: Confusion Matrix Plotting
def visualize_conf_matrix(matrix, title):
    plt.figure(figsize=(6, 4))
    sns.heatmap(matrix, annot=True, fmt="d", cmap="Blues", xticklabels=["Negative", "Positive"], yticklabels=["Negative", "Positive"])
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title(f"Confusion Matrix - {title}")
    plt.show()
```

[16]:
```python
# Utility: ROC-AUC Curve Plotting
def draw_roc_auc(model_name, fpr, tpr, auc_score):
    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, label=f"{model_name} (AUC = {auc_score:.2f})", color="darkorange")
    plt.plot([0, 1], [0, 1], linestyle="--", color="blue")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title(f"{model_name} - ROC AUC Curve")
    plt.legend(loc="lower right")
    plt.grid(True)
    plt.show()
```

11. Function to Compute Classification Metrics

[17]:
```python
# Metrics Calculator
def evaluate_model(name, y_train_fold, y_test_fold, y_pred, y_prob):
    tn, fp, fn, tp = confusion_matrix(y_test_fold, y_pred).ravel()

    print(f"\nConfusion Matrix ({name}):\n", confusion_matrix(y_test_fold, y_pred))
    visualize_conf_matrix(confusion_matrix(y_test_fold, y_pred), name)

    try:
        fpr, tpr, _ = roc_curve(y_test_fold, y_prob)
        auc_val = auc(fpr, tpr)
        draw_roc_auc(name, fpr, tpr, auc_val)
    except:
        print("ROC curve could not be plotted due to insufficient class variance.")

    # Calculate Metrics
    metrics = {
        "True Positive (TP)": tp,
        "True Negative (TN)": tn,
        "False Positive (FP)": fp,
        "False Negative (FN)": fn,
        "Sensitivity (TPR)": tp / (tp + fn) if (tp + fn) else 0,
        "Specificity (TNR)": tn / (tn + fp) if (tn + fp) else 0,
        "False Positive Rate (FPR)": fp / (fp + tn) if (fp + tn) else 0,
        "False Negative Rate (FNR)": fn / (fn + tp) if (fn + tp) else 0,
        "Recall (r)": tp / (tp + fn) if (tp + fn) else 0,
        "Precision (P)": tp / (tp + fp) if (tp + fp) else 0,
        "F1 Measure (F1)": (2 * tp) / (2 * tp + fp + fn) if (2 * tp + fp + fn) else 0,
        "Accuracy": (tp + tn) / (tp + tn + fp + fn),
        "Error Rate": (fp + fn) / (tp + tn + fp + fn),
        "Balanced Accuracy": ((tp / (tp + fn)) + (tn / (tn + fp))) / 2 if (tp + fn and tn + fp) else 0,
        "True Skill Statistics (TSS)": (tp / (tp + fn)) - (fp / (fp + tn)) if (tp + fn and fp + tn) else 0,
        "Heidke Skill Score (HSS)": (2 * ((tp * tn) - (fp * fn))) / (((tp + fn) * (fn + tn)) + ((tp + fp) * (fp + tn))) if (((tp + fn)*(fn + tn) + (tp +
        "ROC_AUC Score": roc_auc_score(y_test_fold, y_prob),
        "Brier Score": brier_score_loss(y_test_fold, y_prob)
    }

    # Brier Skill Score
    baseline_prob = [y_train_fold.mean()] * len(y_test_fold)
    baseline_brier = brier_score_loss(y_test_fold, baseline_prob)
```

12. Model Definitions: Random Forest, KNN, and LSTM

[18]:
```python
# Random Forest Model Function
def run_random_forest(name, results, X_train_fold, y_train_fold, X_test_fold, y_test_fold):
    model = RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X_train_fold, y_train_fold)
    predictions = model.predict(X_test_fold)
    prob_scores = model.predict_proba(X_test_fold)[:, 1]
    results.append(evaluate_model(name, y_train_fold, y_test_fold, predictions, prob_scores))
    return results
```

[19]:
```python
# K-Nearest Neighbors Function
def run_knn(name, results, X_train_fold, y_train_fold, X_test_fold, y_test_fold):
    model = KNeighborsClassifier(n_neighbors=7)
    model.fit(X_train_fold, y_train_fold)
    predictions = model.predict(X_test_fold)
    prob_scores = model.predict_proba(X_test_fold)[:, 1]
    results.append(evaluate_model(name, y_train_fold, y_test_fold, predictions, prob_scores))
    return results
```

[20]:
```python
# LSTM Model Function
def run_lstm(name, results, X_train_lstm, y_train_fold, X_test_lstm, y_test_fold):
    model = Sequential([
        LSTM(64, activation='relu', return_sequences=True, input_shape=(X_train_lstm.shape[1], 1)),
        Dropout(0.2),
        LSTM(32, activation='relu'),
        Dropout(0.2),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    model.fit(X_train_lstm, y_train_fold, epochs=10, batch_size=16, verbose=0)

    predictions = (model.predict(X_test_lstm) > 0.5).astype("int32")
    prob_scores = model.predict(X_test_lstm).flatten()
    results.append(evaluate_model(name, y_train_fold, y_test_fold, predictions, prob_scores))
    return results
```

13. Stratified K-Fold Cross-Validation on Training Data

[22]:
```python
# Initialize Stratified K-Fold for imbalanced dataset
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

for idx, (train_idx, test_idx) in enumerate(skf.split(X_train, y_train), 1):
    print(f"\nRunning Fold {idx}")
    fold_labels.append(f"Fold_{idx}")

    # Fold-specific splits
    X_train_fold = X_train.iloc[train_idx]
    X_test_fold = X_train.iloc[test_idx]
    y_train_fold = y_train.iloc[train_idx]
    y_test_fold = y_train.iloc[test_idx]

    # Scaling features
    fold_scaler = StandardScaler()
    X_train_fold_scaled = fold_scaler.fit_transform(X_train_fold)
    X_test_fold_scaled = fold_scaler.transform(X_test_fold)

    # LSTM reshaping
    X_train_fold_lstm = X_train_fold_scaled.reshape(-1, X_train_fold_scaled.shape[1], 1)
    X_test_fold_lstm = X_test_fold_scaled.reshape(-1, X_test_fold_scaled.shape[1], 1)

    # Train and Evaluate Models
    run_random_forest("Random_Forest", rf_train_metrics, X_train_fold_scaled, y_train_fold, X_test_fold_scaled, y_test_fold)
    run_knn("K_Nearest_Neighbor", knn_train_metrics, X_train_fold_scaled, y_train_fold, X_test_fold_scaled, y_test_fold)
    run_lstm("LSTM", lstm_train_metrics, X_train_fold_lstm, y_train_fold, X_test_fold_lstm, y_test_fold)
```

```
Running Fold 1

Confusion Matrix (Random_Forest):
 [[30  4]
 [ 7 33]]
```

14. Average Performance Comparison Across Folds

[25]:
```python
# Compute Average Performance per Algorithm
rf_avg = rf_df.mean(axis=1)
knn_avg = knn_df.mean(axis=1)
lstm_avg = lstm_df.mean(axis=1)

average_summary = pd.DataFrame({
    "Random Forest": rf_avg,
    "K-Nearest Neighbor": knn_avg,
    "LSTM": lstm_avg
})

print("\nAverage Cross-Validation Performance Metrics:")
print(average_summary)
```

```
Average Cross-Validation Performance Metrics:
                                Random Forest  K-Nearest Neighbor       LSTM
True Positive (TP)                  35.300000           35.700000  32.800000
True Negative (TN)                  27.800000           27.400000  22.600000
False Positive (FP)                  5.500000            5.900000  10.700000
False Negative (FN)                  4.800000            4.400000   7.300000
Sensitivity (TPR)                    0.880305            0.890244   0.817988
Specificity (TNR)                    0.834581            0.822727   0.679144
False Positive Rate (FPR)            0.165419            0.177273   0.320856
False Negative Rate (FNR)            0.119695            0.109756   0.182012
Recall (r)                           0.880305            0.890244   0.817988
Precision (P)                        0.865888            0.859546   0.772456
F1 Measure (F1)                      0.872569            0.873719   0.783173
Accuracy                             0.859700            0.859663   0.754795
Error Rate                           0.140300            0.140337   0.245205
Balanced Accuracy                    0.857443            0.856486   0.748566
True Skill Statistics (TSS)          0.714886            0.712971   0.497132
Heidke Skill Score (HSS)             0.716351            0.715750   0.499863
ROC_AUC Score                        0.925086            0.910826   0.839180
Brier Score                          0.105723            0.111724   0.171091
Brier Skill Score                    0.573435            0.549215   0.309738
```

15. Evaluating Final Models on Test Data

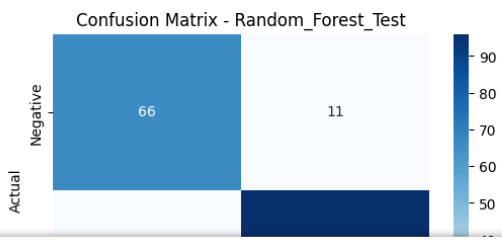
[26]:
```python
# Scale the full training and test sets
final_scaler = StandardScaler()
X_train_final = final_scaler.fit_transform(X_train)
X_test_final = final_scaler.transform(X_test)
```

[27]:
```python
# Reshape for LSTM input
X_train_final_lstm = X_train_final.reshape(-1, X_train_final.shape[1], 1)
X_test_final_lstm = X_test_final.reshape(-1, X_test_final.shape[1], 1)
```

[28]:
```python
# Lists to store final test results
rf_test_metrics, knn_test_metrics, lstm_test_metrics = [], [], []
```

[29]:
```python
# Final model evaluations on unseen test set
run_random_forest("Random_Forest_Test", rf_test_metrics, X_train_final, y_train, X_test_final, y_test)
run_knn("K_Nearest_Neighbor_Test", knn_test_metrics, X_train_final, y_train, X_test_final, y_test)
run_lstm("LSTM_Test", lstm_test_metrics, X_train_final_lstm, y_train, X_test_final_lstm, y_test)
```

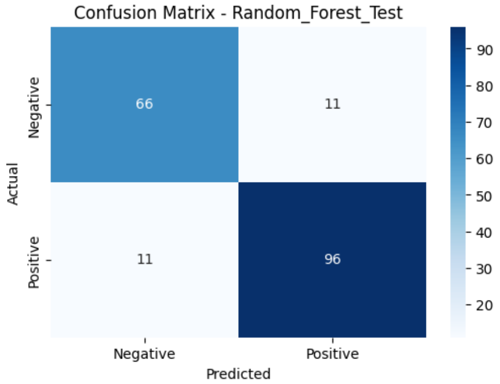
```
Confusion Matrix (Random_Forest_Test):
 [[66 11]
 [11 96]]
```

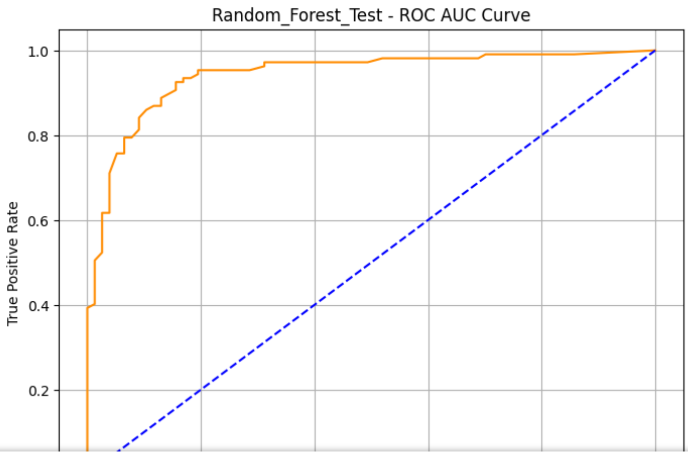


[30]:
```python
# Convert to DataFrames
```

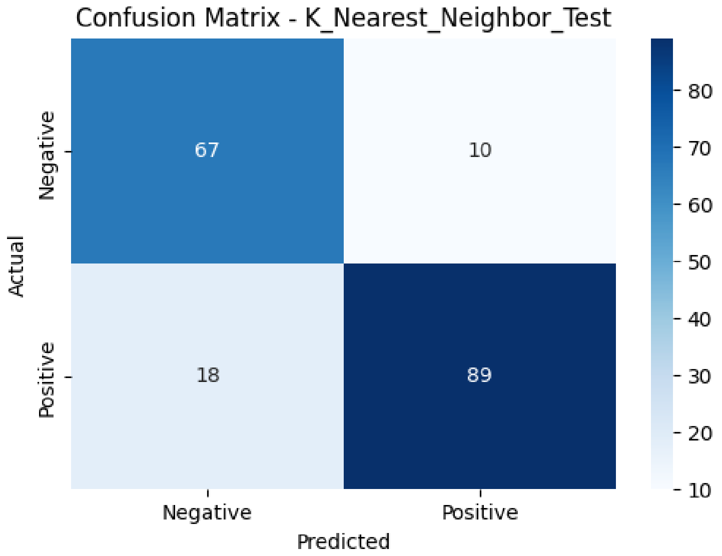
16. Confusion Matrix and ROC Curve – Random Forest (Test Data)

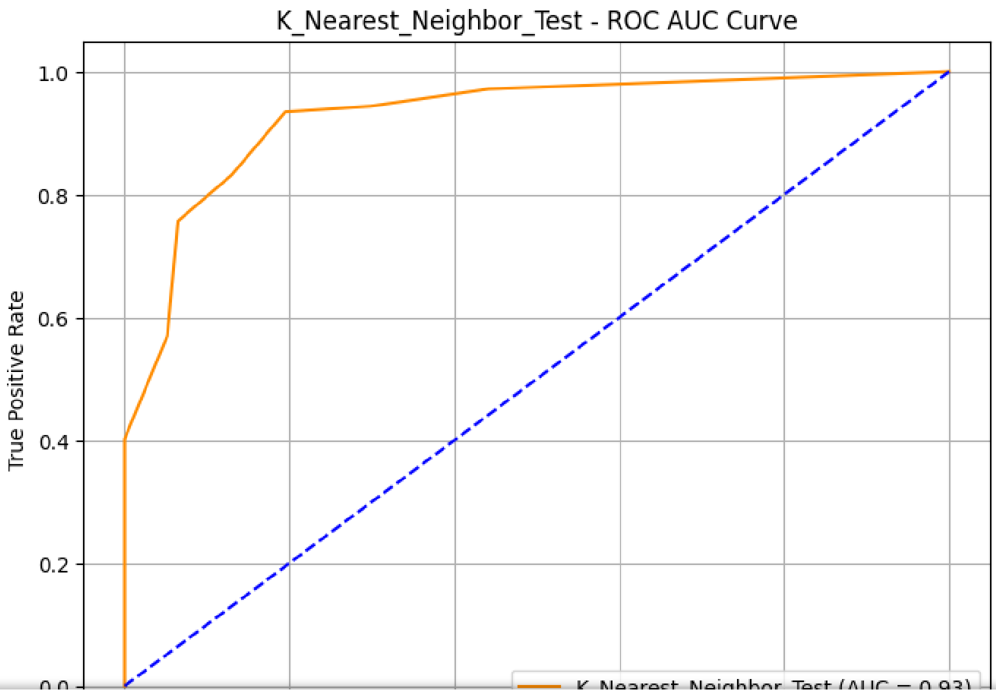
```
Confusion Matrix (Random_Forest_Test):
 [[66 11]
 [11 96]]
```



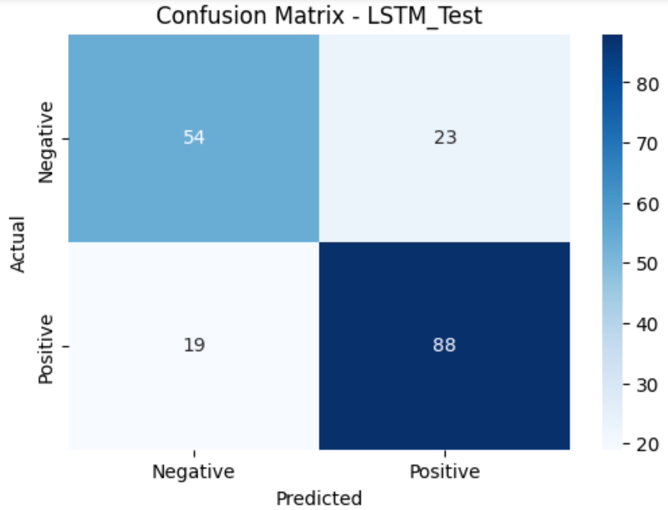


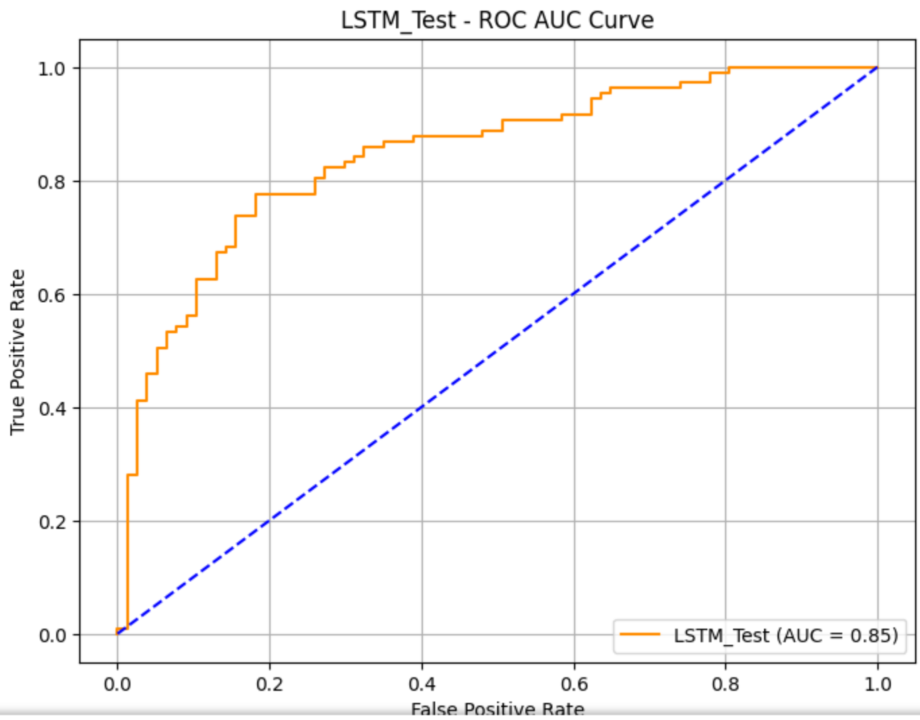
```
Confusion Matrix (K_Nearest_Neighbor_Test):
 [[67 10]
 [18 89]]
```





18. Confusion Matrix and ROC Curve – LSTM (Test Data)






```
[29]: [{'True Positive (TP)': np.int64(88),
        'True Negative (TN)': np.int64(54),
        'False Positive (FP)': np.int64(23),
        'False Negative (FN)': np.int64(19),
        'Sensitivity (TPR)': np.float64(0.822429906542056),
        'Specificity (TNR)': np.float64(0.7012987012987013),
        'False Positive Rate (FPR)': np.float64(0.2987012987012987),
        'False Negative Rate (FNR)': np.float64(0.17757009345794392),
        'Recall (r)': np.float64(0.822429906542056),
        'Precision (P)': np.float64(0.7927927927927928),
        'F1 Measure (F1)': np.float64(0.8073394495412844),
        'Accuracy': np.float64(0.7717391304347826),
        'Error Rate': np.float64(0.22826086956521738),
        'Balanced Accuracy': np.float64(0.7618643039203787),
        'True Skill Statistics (TSS)': np.float64(0.5237286078407574),
        'Heidke Skill Score (HSS)': np.float64(0.5275706076537474),
        'ROC_AUC Score': np.float64(0.8458550794999393),
        'Brier Score': np.float64(0.15845459188131197),
        'Brier Skill Score': np.float64(0.3521710647564258)}]
```

19. Test Set Metrics Comparison for All Models

```
[30]: # Convert to DataFrames
rf_test_df = pd.DataFrame(rf_test_metrics).T
knn_test_df = pd.DataFrame(knn_test_metrics).T
lstm_test_df = pd.DataFrame(lstm_test_metrics).T
```

```
[31]: # Combine Test Metrics
test_results_df = pd.concat([rf_test_df, knn_test_df, lstm_test_df], axis=1)
test_results_df.columns = ["Random Forest", "K-Nearest Neighbor", "LSTM"]
```

```
[32]: print("\nTest Set Performance Comparison:")
print(test_results_df)
```

```
Test Set Performance Comparison:
                              Random Forest  K-Nearest Neighbor       LSTM
True Positive (TP)                96.000000           89.000000  88.000000
True Negative (TN)                66.000000           67.000000  54.000000
False Positive (FP)               11.000000           10.000000  23.000000
False Negative (FN)               11.000000           18.000000  19.000000
Sensitivity (TPR)                  0.897196            0.831776   0.822430
Specificity (TNR)                  0.857143            0.870130   0.701299
False Positive Rate (FPR)          0.142857            0.129870   0.298701
False Negative Rate (FNR)          0.102804            0.168224   0.177570
Recall (r)                         0.897196            0.831776   0.822430
Precision (P)                      0.897196            0.898990   0.792793
F1 Measure (F1)                    0.897196            0.864078   0.807339
Accuracy                           0.880435            0.847826   0.771739
Error Rate                         0.119565            0.152174   0.228261
Balanced Accuracy                  0.877170            0.850953   0.761864
True Skill Statistics (TSS)        0.754339            0.701906   0.523729
Heidke Skill Score (HSS)           0.754339            0.691829   0.527571
ROC_AUC Score                      0.941862            0.927843   0.845855
Brier Score                        0.095695            0.106810   0.158455
Brier Skill Score                  0.608758            0.563315   0.352171
```

20. Identifying Best Model Based on Train and Test Accuracies

```
[33]: # Identify Best Performing Algorithm on Test Accuracy
test_accuracy_scores = {
    "Random Forest": rf_test_metrics[0]["Accuracy"],
    "K-Nearest Neighbor": knn_test_metrics[0]["Accuracy"],
    "LSTM": lstm_test_metrics[0]["Accuracy"]
}
best_test_model = max(test_accuracy_scores, key=test_accuracy_scores.get)
print(f"\nBest Accuracy on Test Set: {best_test_model} ({test_accuracy_scores[best_test_model] * 100:.2f}%)")
```

```
Best Accuracy on Test Set: Random Forest (88.04%)
```

```
[34]: # Best Train Accuracy (based on average CV accuracy)
train_accuracy_scores = {
    "Random Forest": rf_avg["Accuracy"],
    "K-Nearest Neighbor": knn_avg["Accuracy"],
    "LSTM": lstm_avg["Accuracy"]
}
best_train_model = max(train_accuracy_scores, key=train_accuracy_scores.get)
print(f"Best Accuracy on Train Set: {best_train_model} ({train_accuracy_scores[best_train_model] * 100:.2f}%)")
```

```
Best Accuracy on Train Set: Random Forest (85.97%)
```

```
[ ]:
```

Link to GitHub Repository:

https://github.com/Pikaboo69/Pikaboo69-Mohammed_Sameer_Khan_FinalTerm_Project