NAME: Yash Chaudhary

NJIT UCID: ygc2

Email Address: ygc2@njit.edu

Date: November 24, 2024

Professor: Yasser Abdullah

Course: FA24-CS6364101 / Data Mining

FINAL PROJECT REPORT

COMPARATIVE ANALYSIS OF BINARY CLASSIFICATION MODELS ON THE BANK NOTE AUTHENTICATION DATASET

TABLE OF CONTENT

ABSTRACT

This report specifically evaluates the performance of four different machine learning models: Random Forest, LSTM (Long Short-Term Memory), KNN (K-Nearest Neighbors), and SVM (Support Vector Machine) on the banknote authentication dataset. These models are to be assessed for their performance for binary classification using metrics gathered from the 10-fold cross-validation method. The dataset is preprocessed according to scaling and handling of missing values, followed by exploratory data analysis of the dataset to analyze feature distributions and correlations.

The models were evaluated on other evaluation metrics such as accuracy, recall, precision and many others to give further insight. Along with the quantitative results presented graphically as a distribution of the target variable, correlation heatmap, and ROC curves.

Each model is presented with its strengths and limitations and recommends selecting the best model according to evaluation results. This analysis, thus, will help orient the readers toward ways to select and tune models for binary classification problems.

INTRODUCTION

The goal is to determine which of the models could separate the real from counterfeit banknotes best. The study emphasized the fact that model evaluation must be done based on appropriate measures and validation techniques for reliability and generalization of predictive performance.

The report consists of data preprocessing, exploratory data analysis, and extensive model assessments based on 10-fold cross-validation. The performance measures such as accuracy, precision, recall, F1-score, and AUC are calculated, and some advanced measures such as Brier Score, True Skill Statistics, and Heidke Skill Score are presented for more thorough insight into how the model performed.

Thus, the development of this report sets out to find the most suited model for binary classification tasks. It would present a systematic approach for model selection and structuring optimization decisions in real-life applications.

CORE CONCEPTS AND PRINCIPLES

This section describes various foundational concepts and principles of classifying banknotes as either genuine or counterfeit. The principles served to guide the methodology and analysis are explained in the report.

---

1. Binary Classification

Binary classification is intended to classify items into one of two distinct classes. For this task:

- **Class 0:** Genuine banknotes.
- **Class 1:** Counterfeit banknotes.

The main aim is to develop predictive models which would classify between the two classes based on extracted features.

---

2. Feature Analysis

The dataset contains features, from the wavelet transformation of banknote images, which serves as predictors in classification.

- **Variance:** A measure of the spread of contour pixels that helps identify anomalies in textures.
- **Skewness:** A measure of asymmetry in the intensity distribution which is often found on counterfeit notes.
- **Curtosis:** A measure of the sharpness of the curve of the intensity distributions.
- **Entropy:** A measure of randomness/chaos in the image that gives an insight into the integrity of the structure in banknotes.

These features encapsulate the critical statistical properties that can offer differentiation between original and counterfeit notes.

---

3. Supervised Learning Techniques

In the assessment of the prediction of the authenticity of banknotes, four different methods of machine-learning were considered:

- **Random Forest (RF):** The ensemble method makes many decision trees and combines them in such a way that predictive accuracy and robustness are superior.
- **K-Nearest Neighbors (KNN):** Proximity-based model to predict the class of a sample based on the majority class of its nearest neighbors.
- **Support Vector Machine (SVM):** It is a classifier finding the hyperplane that best separates the classes and maximizes the margin.
- **Long Short-Term Memory (LSTM):** A deep learning technique used here to find the relationships amongst the features.

---

4. Evaluation Metrics

To assess model performance comprehensively, a diverse set of metrics are used:

➢ **Basic Metrics:**

- **Accuracy:** Proportion of correctly classified instances.
- **Precision:** Ratio of true positives to all predicted positives.
- **Recall (Sensitivity):** Represents the ability of model to identify positive outcomes.
- **F1-Score:** Harmonic mean of precision and recall.

➢ **Statistical and Probabilistic Metrics:**

- **P (Positive):** Total positive instances in the dataset.
- **N (Negative):** Total negative instances in the dataset.
- **Error Rate:** Proportion of misclassified instances.
- **Balanced Accuracy (BACC):** Average of TPR and TNR, useful for imbalanced datasets.
- **True Skill Statistics (TSS):** Measures classification skill independent of prevalence.
- **Heidke Skill Score (HSS):** Evaluates agreement beyond chance-level classification.
- **Brier Score (BS):** Measures the accuracy of probabilistic predictions.
- **Brier Skill Score (BSS):** Compares the BS against a baseline model.
- **AUC Score (Area Under the Curve):** Evaluates the classifier's ability to rank positive and negative samples correctly across all thresholds.

- **Advanced Metrics:**

  - **True Positive Rate (TPR):** Proportion of actual positives correctly classified.
  - **True Negative Rate (TNR):** Proportion of actual negatives correctly classified.
  - **False Positive Rate (FPR):** Proportion of negatives misclassified as positives.
  - **False Negative Rate (FNR):** Proportion of positives misclassified as negatives.

These metrics collectively provide a holistic evaluation, capturing different aspects of model performance, robustness, and reliability.

---

5. Validation Approach

The model was evaluated by a **stratified 10-fold cross-validation** in which each fold preserved the ratio of classes. The dataset was partitioned into 10 subsets, with each subset being used as validation data once. The other 9 subsets served as training data. This procedure demonstrates the grip of robustness and unbiasedness in evaluating the performance of the model with respect to all metrics.

---

6. Applications of Association Rule Mining

Detection of counterfeit banknotes could be of crucial importance in the following areas:

- **Banking and Retail:** Automatic counterfeit detection in banks and point of sale payment systems.
- **ATMs:** Increased security through the prevention of the circulation of counterfeit money.
- **Law Enforcement:** To enhance investigations into counterfeit currency operations.
- **Research:** A dataset to benchmark advancements in machine learning models for fraud detection.

These applications would help reduce financial fraud and create a safe transactional environment across the globe.

PROJECT FLOW

1. Problem Definition and Objective

The principal objective of the project is to evaluate and compare the various machine learning models for the binary classification banknotes as either authentic or counterfeit. Counterfeit detection has become an important consideration in financial systems, and in this regard, the present study investigates data-driven approaches with the aim of improving the correctness and reliability of authentic banknotes detection. The project, in addition to using the banknote authentication dataset, aims to:

- Develop and test machine learning models for banknote classification as genuine or counterfeit.
- Present a performance comparative analysis of the models based on various evaluation metrics.
- Provide conclusions on the advantages and limitations of each model that can serve as guidance for future implementations.

---

2. Dataset Selection

The **Banknote Authentication Dataset**, downloaded from the **UCI Machine Learning Repository**, was chosen for this project because of its applicability to the **binary classification** problem. This dataset contains real-life instances extracted as wavelet features from banknotes images. Each instance has been labelled **genuine = 1 or counterfeit = 0**, providing a clear binary target variable for analysis.

Data were extracted from images that were taken from genuine and forged banknote-like specimens. For digitization, an industrial camera usually used for print inspection was used. The final images have **400x 400 pixels**. Due to the object lens and distance to the investigated object gray-scale pictures with a resolution of about **660 dpi** were gained. **Wavelet Transform tool** were used to extract features from images.

Link to Dataset: UCI Banknote Authentication Dataset

---

3. Preprocessing the Data

a) Feature Scaling:

- The dataset comprises of numerical features with varying ranges. Standard scaling was applied so that models like **KNN** and **LSTM** do not get biased toward features with larger magnitudes.
- This step transformed all feature values into a uniform range of **[0, 1]** by preserving relative differences.

b) Data Splitting: The dataset was split into two subsets:

- **80%** of data was used for **training**.
- **20%** of data was held out for **testing**.

This allows the generalizability of the models being evaluated on unseen data.

c) Feature-Target Separation:

- Only **features (X)** and **targets (y)** were explicitly separated for ease of understanding during training and evaluation. This ensures the flow of feeding the models with the right inputs in a streamlined way.

d) SMOTE (Synthetic Minority Oversampling Technique):

- Generated synthetic samples for the minority class to ensure a balanced dataset.

e) Standard Scaler:

- To normalize the features, improving performance for algorithms like KNN and SVM.

---

4. Hyperparameter Optimization

- Applied grid search and cross-validation techniques to identify the best hyperparameters for each model:
  - **Random Forest (RF):** Tuned the number of trees, maximum depth, and minimum samples per split.
  - **K-Nearest Neighbors (KNN):** Optimized the number of neighbors (`k`).
  - **Support Vector Machine (SVM):** Fine-tuned the kernel type and regularization parameter.
  - **Long Short-Term Memory (LSTM):** Adjusted learning rate, batch size, and number of neurons.
- Integrated the optimized models into the workflow for final evaluation.

5. Exploratory Data Analysis (EDA)

Conducted a comprehensive analysis of the dataset through:

- **Target Distribution Analysis**: Examined the distribution of positive and negative outcomes to detect class imbalances.
- **Correlation Matrix**: Generated a heatmap to identify relationships among features.
- **Feature Distribution**: Plotted histograms to visualize the spread of data across individual features.
- **Pair Plots**: Highlighted relationships between feature pairs with respect to the target variable.

---

6. Model Training and Evaluation

1. Trained each model using the **10-Fold Cross-Validation** method to ensure reliable performance evaluation.
2. Measured multiple metrics for each fold:
    - Accuracy, Precision, Recall, and F1-Score.
    - Advanced metrics like True Positive Rate (TPR), False Positive Rate (FPR), True Negative Rate (TNR) and False Negative Rate (FNR).
    - P (Positive), N (Negative), Error Rate (ER), Balanced Accuracy (BACC), True Skill Statistics (TSS), Heidke Skill Score (HSS), Brier Score (BS), Brier Skill Score (BSS) for statistical and probabilistic calibration.
    - Area Under the ROC Curve (AUC) to evaluate discrimination capability.

---

7. Insights and Recommendations

- Analyzed the aggregated metrics to determine the most effective model for the given dataset.
- Highlighted model-specific advantages, limitations, and potential areas of improvement.

RESULTS AND EVALUATIONS

The project evaluation was based on the optimized implementations and each model was optimized by systematic hyperparameter tuning in order to better its performance.

---

1. Optimization and Hyperparameter Tuning

- **Random Forest: n_estimators**, **max_depth**, and **min_samples_split** is among the parameters that have been optimized by grid search. This gave a balance between the generalization and the complexity of the model.
- **LSTM**: The architecture of the model was tuned along with the number of units in the LSTM layer and the dropout rate. **Randomized search** was adopted for optimizing the **optimizer**, **batch size**, and the number of **epochs**. Early stopping has also been implemented to avoid overfitting.
- **KNN:** To do this, **grid search** was used to optimize the number of neighbors, **n_neighbors,** and weighting methods, **uniform or distance**, for fast and efficient nearest-neighbor searches.
- **SVM:** Kernels- **linear, rbf**, poly-regularization parameter, **C**, and **gamma** were tuned with the help of grid search, which helped the model in getting better decision boundaries.

---

2. Model Performance Comparison

- Random Forest gave consistently good results; it has a high accuracy and AUC, suggesting that this algorithm ran without the problems of feature interactions.
- LSTM exploited its sequence modeling capability and competed fairly, especially in recall, which may suggest that it is suitable for positive case identification.
- KNN had a pretty decent performance but was a bit affected by class imbalance.
- SVM behaved predictably; the RBF kernel gave the best results from the tried configurations.

---

3. Evaluation Metrics Accuracy

- Random Forest and LSTM outperformed other models, consistently with high accuracy across folds.
- Precision & Recall: LSTM did best in recall and hence captured most of the positive cases, but Random Forest had the best precision.
- F1-Score: Random Forest and SVM gave the best balance between precision and recall.
- AUC: Optimized Random Forest resulted in the highest value of AUC and hence the best class separation.

---

4. ROC Curve Analysis

- The SVM model had the best balance between the true positive rate and the false positive rate. Also, the AUC value was the highest.
- LSTM gave strong recall despite the fact that the precision is relatively a little lower, which further established its sequential learning capability.

---

5. Insights

- **Best Model:** While SVM achieved the highest accuracy during evaluation, it exhibited signs of overfitting, which may limit its generalizability in real-world scenarios. On the other hand, LSTM demonstrated robust performance with better adaptability, making it the preferred choice when real-world application and reliability are key priorities.

---

6. Model Specific Recommendations

- **Random Forest:** Try more ensembling techniques such as boosting.
- **LSTM**: More tunings of layer configurations or advanced architectures such as bidirectional LSTMs.
- **KNN**: Handle sensitivity to imbalance by using techniques such as oversampling or weighted distances.
- **SVM**: Further, kernel customization at finer details or higher-dimensional mapping may lead to better results.

---

7. General Recommendations

- Further preprocessing and feature engineering could help in better preparation of inputs for the models. Ensemble modeling to combine the strengths of Random Forest and SVM. These results, coupled with the systematic hyperparameter optimization implemented here, support the chosen models' performance and allow for actionable suggestions toward further improvements.

HOW TO RUN THE CODE?

- ## Prerequisites

Before running the code, ensure that you have the following installed on your system:

1. **Python 3.6 or higher**: The code is compatible with Python versions 3.6 and above. You can download Python from python.org.
2. **Required Libraries**:

    - **numpy**: Used for numerical computations.
    - **pandas**: For data manipulation and analysis.
    - **seaborn**: For statistical data visualization.
    - **scikit-learn**: Machine learning algorithms and utilities.
    - **imblearn**: For handling class imbalance (SMOTE).
    - **scikeras**: Wrapper to integrate Keras models with scikit-learn.
    - **tensorflow**: For deep learning models like LSTM.
    - **matplotlib**: For creating static, animated, and interactive visualizations.
    - **tabulate**: To display tabular data in a nice format.

3. The installation of the required can be done by using executing **requreiments.txt** file or by **manually** installing by user:

- To install the necessary libraries for this project, a **requirements.txt** file is included. This file lists all the external libraries needed to run the code.
    - Make sure the **requirements.txt** file is in the same folder as your project file then you can run the following command in your terminal or command prompt after navigating to the folder with the project and **requirements.txt**, now run this command:

    "pip install -r requirements.txt"

    This will install all necessary libraries automatically.

- Install the necessary libraries using "**pip**". You can run the following command in your terminal or command prompt:

    "**pip install numpy pandas seaborn scikit-learn imblearn scikeras tensorflow matplotlib tabulate"**

    This command installs all required packages to run this program

- ## Setting Up the Environment

1. **Download the Code**: Clone or Download the repository from GitHub Link provided at the last page of the report and ensure you have the complete code saved in a (**.py)** file or a Jupyter Notebook **(.ipynb)** file.

2. **Dataset**: Make sure the csv dataset named "**banknote_dataset.csv**" is in the same folder as of main file **"source_code.py**"

- ## Running the Code

1. **Open Terminal/Command Prompt**:
   - Navigate to the **directory** where your **Python file** or **Jupyter Notebook** is saved.
2. **Execute the Code**:
   - For a Python script, run the following command:

     "python source_code.py" or "python3 source_code.py"

   - For a Jupyter Notebook, you can open it using Jupyter Notebook or JupyterLab. If you don't have Jupyter installed, you can do so by running:

     "pip install notebook"

   - Then start with Jupyter Notebook:

     "jupyter notebook"
3. **Follow On-Screen Prompts**:
   - The program will guide you through the program and every detail to interpret the analysis.

- **Troubleshooting Common Issues**

    - **Import Errors**: If you encounter errors related to missing libraries, ensure that you have installed all required libraries as mentioned above. The warnings packages are included with Python and do not require separate installation.

    - **File Not Found**: If you get an error saying the CSV file is not found, double-check the file path and ensure the file is in the specified location. (In the same folder as of the source code main file)

SCREENSHOTS OF CODE

```python
# Set up the environment for TensorFlow, suppress unnecessary logs, and warnings
import os
os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'  # Disable oneDNN optimizations for TensorFlow
import tensorflow as tf
tf.get_logger().setLevel('ERROR')  # Suppress TensorFlow logs
import warnings
warnings.filterwarnings("ignore")  # Suppress all warnings

# Import necessary libraries
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.svm import SVC
from tabulate import tabulate
import matplotlib.pyplot as plt
from imblearn.over_sampling import SMOTE
from scikeras.wrappers import KerasClassifier
from tensorflow.keras.models import Sequential
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold, train_test_split
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import RandomizedSearchCV
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.metrics import confusion_matrix, roc_curve, auc, brier_score_loss
```

Section: 1

This screenshot shows the setup of a TensorFlow environment with necessary libraries for machine learning, including SVM, KNN, Random Forest, and LSTM, along with tools for model evaluation and hyperparameter tuning. Warnings and logs are suppressed for cleaner output.

```python
# Welcome Message and User Prompt
def welcome_message():
    print("\n" + "=" * 162)
    print("Welcome to the Banknote Authentication Model Evaluation Program!")
    print("\nThis program allows you to:")
    print("1. Load and preprocess the banknote dataset.")
    print("2. Train and evaluate four different models:")
    print("   - Random Forest")
    print("   - Deep Learning: LSTM (Long Short Term Memory)")
    print("   - Algorithm: KNN (K-Nearest Neighbor)")
    print("   - Algorithm: SVM (Support Vector Machine)")
    print("3. Generate detailed insights and recommendations based on the evaluation results.")

    print("\n**IMPORTANT**: For optimal readability and proper formatting of outputs, please use the terminal in full-screen view.")

    print("\nYou will receive:")
    print("   - A description of the dataset.")
    print("   - Information about the dataset, including data types, missing values, etc.")
    print("   - A preview of the first few records in the dataset (head).")
    print("   - Metric Table showing the results of 10-fold cross-validation for each model.")
    print("   - Table displaying the average metrics across all 10 folds for each model.")
    print("   - Dynamic insights and recommendations based on the performance of models.")

    print("\n**Graphs you'll see**:")
    print("   - Distribution of the target variable (Target)")
    print("   - Correlation matrix heatmap showing relationships between features")
    print("   - Pair plot visualizing data relationships with the target variable")
    print("   - Visualizing the model's performance with ROC curves for each model.")

    print("\nPress 'Enter' to continue with the analysis...")
    input()  # Wait for the user to press Enter before continuing
    print("=" * 162 + "\n")
```

Section: 2

This code displays a detailed welcome message introducing the Banknote Authentication Model Evaluation Program, outlining its features and expected outputs, including model training, evaluation, and visualization. The user is prompted to press 'Enter' to begin the analysis.

```python
# Load Dataset Function
def load_data(file_path):
    """
    Load dataset from the provided file path.

    Parameters:
        file_path (str): Path to the CSV file.

    Returns:
        pd.DataFrame: Loaded dataset as a Pandas DataFrame, or None if an error occurs.
    """
    try:
        data = pd.read_csv(file_path)
        if data.empty:
            raise ValueError("The dataset is empty.")
        return data
    except Exception as e:
        print(f"Error loading the dataset: {e}")
        return None
```

```python
# Preprocess Dataset Function
def preprocess_data(data):
    """
    Preprocess the dataset to handle missing values, validate the target column, and scale features.

    Parameters:
        data (pd.DataFrame): Input dataset.

    Returns:
        tuple: Scaled train-test splits (X_train, X_test, y_train, y_test), or None if validation fails.
    """
    # Handle missing values
    if data.isnull().sum().any():
        print("Missing values detected. Filling missing values with mean (for numeric columns)...")
        data.fillna(data.mean(), inplace=True)

    # Validate dataset structure
    if data.shape[1] < 2:
        print("Dataset is not valid. There should be at least one feature column and a target column.")
        return None

    # Split data into features (X) and target (y)
    X = data.iloc[:, :-1].values  # Features (all columns except the last)
    y = data.iloc[:, -1].values   # Target variable (last column)

    # Validate target column (binary classification)
    if not set(y).issubset({0, 1}):
        print("Error: The target variable should only contain binary labels (0, 1).")
        return None

    # Scale features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Train-test split
    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

    return X_train, X_test, y_train, y_test
```

Section: 3

These functions load and handles missing values, validates the target column for binary classification, scales features, and splits the dataset into training and testing sets. It returns the processed data or None if validation fails.

```python
# Call the Welcome Message Function
welcome_message()

# Load and preprocess the dataset
data = load_data('./data/banknote_data.csv')
X_train, X_test, y_train, y_test = preprocess_data(data)

# Reshape data for LSTM (expects 3D input)
X_train_lstm = X_train.reshape(X_train.shape[0], 1, X_train.shape[1])  # Reshaped to (samples, 1, features)
X_test_lstm = X_test.reshape(X_test.shape[0], 1, X_test.shape[1])      # Same reshaping for test data

# Feature and Label Separation
features = data.iloc[:, :-1]  # All columns except the last (features)
labels = data.iloc[:, -1]     # Last column (labels)

# Visualize the Target Variable Distribution
plt.figure(figsize=(8, 6))
sns.countplot(x=labels)
plt.title('Distribution of Target Variable')
plt.xlabel('Target')
plt.ylabel('Count')
plt.show()

# Calculate and Display Data Imbalance Information
positive_outcomes, negative_outcomes = labels.value_counts()
total_samples = labels.count()
print("\nChecking for Data Imbalance:")
print(f"   - Number of Positive Outcomes: {positive_outcomes}")
print(f"   - Percentage of Positive Outcomes: {round((positive_outcomes / total_samples) * 100, 2)}%")
print(f"   - Number of Negative Outcomes: {negative_outcomes}")
print(f"   - Percentage of Negative Outcomes: {round((negative_outcomes / total_samples) * 100, 2)}%")
print("\n\n" + "=" * 162)

# Create and Display Correlation Heatmap
fig, axis = plt.subplots(figsize=(8, 8))
correlation_matrix = features.corr()
sns.heatmap(correlation_matrix, annot=True, linewidths=.5, fmt='.2f', ax=axis)
plt.show()

# Plot Histograms for Feature Distribution
features.hist(figsize=(10, 10))
plt.show()

```

```python
# Create a Pairplot to Visualize Feature Relationships with the Target Variable
sns.pairplot(data, hue=' Target')
plt.show()

print('''\n\nThe model optimization and training process may take a few minutes.
It might feel like it's taking forever, but don't worry—it will complete successfully! :)\n\n''')
print("\nOptimization Parameters for models: ")
```

Section: 4

This code displays the welcome message, loads and preprocesses the dataset, reshapes the data for LSTM, separates features and labels, visualizes the target variable distribution and checks and displays data imbalance information. It then creates and displays a correlation heatmap, plots histograms for feature distributions, and generates a pair plot to visualize relationships with the target variable.

```python
# Function to optimize KNN classifier
def optimize_knn(X_train, y_train):
    """
    Optimizes the KNN model using GridSearchCV to find the best hyperparameters.
    Parameters:
        X_train: Features of the training dataset.
        y_train: Labels of the training dataset.
    Returns:
        Best KNN model after hyperparameter tuning.
    """
    # Feature scaling for better distance calculations
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)

    # Handle class imbalance using SMOTE
    smote = SMOTE(random_state=42)
    X_train, y_train = smote.fit_resample(X_train, y_train)

    # Define hyperparameter grid for KNN
    param_grid = {
        'n_neighbors': [3, 5, 7],  # Number of neighbors to use
        'weights': ['uniform', 'distance'],  # Weighting strategy
        'p': [1, 2]  # Distance metric: Manhattan (1) or Euclidean (2)
    }

    # Perform grid search
    grid = GridSearchCV(KNeighborsClassifier(), param_grid, cv=3, scoring='accuracy', n_jobs=-1)
    grid.fit(X_train, y_train)
    print("Parameters for KNN:", grid.best_params_)

    return grid.best_estimator_
```

```python
# Function to optimize Random Forest classifier
def optimize_random_forest(X_train, y_train):
    """
    Optimizes the Random Forest model using RandomizedSearchCV to find the best hyperparameters.
    Parameters:
        X_train: Features of the training dataset.
        y_train: Labels of the training dataset.
    Returns:
        Best Random Forest model after hyperparameter tuning.
    """
    # Define hyperparameter grid for Random Forest
    param_grid = {
        'n_estimators': [50, 100],  # Number of trees in the forest
        'max_depth': [None, 10],  # Maximum depth of the tree
        'min_samples_split': [5, 10],  # Minimum number of samples to split a node
        'min_samples_leaf': [2, 4]  # Minimum number of samples in a leaf node
    }

    # Perform randomized search
    grid = RandomizedSearchCV(RandomForestClassifier(), param_distributions=param_grid,
                              n_iter=10, cv=3, scoring='accuracy', n_jobs=-1)
    grid.fit(X_train, y_train)
    print("Parameters for Random Forest:", grid.best_params_)

    return grid.best_estimator_
```

```python
# Function to optimize SVM
def optimize_svm(X_train, y_train):
    """
    Optimizes the SVM model using GridSearchCV to find the best hyperparameters.
    Parameters:
        X_train: Features of the training dataset.
        y_train: Labels of the training dataset.
    Returns:
        Best SVM model after hyperparameter tuning.
    """
    # Feature scaling for better SVM performance
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)

    # Handle class imbalance using SMOTE
    smote = SMOTE(random_state=42)
    X_train, y_train = smote.fit_resample(X_train, y_train)

    # Define hyperparameter grid for SVM
    param_grid = {
        'C': [0.1, 1, 10],  # Regularization parameter
        'kernel': ['linear', 'rbf'],  # Kernel type
        'gamma': [0.01, 0.1, 'scale']  # Kernel coefficient
    }

    # Perform grid search
    grid = GridSearchCV(SVC(probability=True), param_grid, cv=3, scoring='accuracy', n_jobs=-1)
    grid.fit(X_train, y_train)
    print("Parameters for SVM:", grid.best_params_)

    return grid.best_estimator_
```

```python
# Function to optimize LSTM
def optimize_lstm(X_train, y_train, input_shape):
    """
    Optimizes the LSTM model using RandomizedSearchCV to find the best hyperparameters.
    Parameters:
        X_train: Features of the training dataset.
        y_train: Labels of the training dataset.
        input_shape: Shape of the input data for LSTM.
    Returns:
        Best LSTM model after hyperparameter tuning.
    """
    # Reshape the input data for LSTM compatibility
    X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))

    # Define the LSTM model architecture
    def create_model(optimizer='adam'):
        model = Sequential()
        model.add(LSTM(50, activation='relu', input_shape=input_shape))  # LSTM layer with 50 units
        model.add(Dropout(0.2))  # Dropout layer to reduce overfitting
        model.add(Dense(1, activation='sigmoid'))  # Output layer for binary classification
        model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
        return model

    # Wrap the model for compatibility with RandomizedSearchCV
    model = KerasClassifier(build_fn=create_model, verbose=0)

    # Define hyperparameter grid for LSTM
    param_grid = {
        'optimizer': ['adam', 'rmsprop'],  # Optimizers to test
        'batch_size': [16, 32, 64],  # Batch sizes
        'epochs': [5, 10],  # Number of epochs
    }

    # Perform randomized search
    grid = RandomizedSearchCV(estimator=model, param_distributions=param_grid, cv=3, n_iter=5)

    # Fit the model using randomized search
    grid.fit(X_train, y_train)

    # Print best parameters after fitting
    print("Best Parameters for LSTM:", grid.best_params_)

    # Use EarlyStopping to prevent overfitting
    early_stopping = EarlyStopping(monitor='val_loss', patience=5)
```

```python
    # Re-train the model with the best hyperparameters
    model = grid.best_estimator_
    model.fit(X_train, y_train, epochs=grid.best_params_['epochs'], batch_size=grid.best_params_['batch_size'], callbacks=[early_stopping])

    return model

# Updated training functions to include optimization
def train_knn_optimized(X_train, y_train):
    model = optimize_knn(X_train, y_train)
    model.fit(X_train, y_train)
    return model

def train_random_forest_optimized(X_train, y_train):
    model = optimize_random_forest(X_train, y_train)
    model.fit(X_train, y_train)
    return model

def train_svm_optimized(X_train, y_train):
    model = optimize_svm(X_train, y_train)
    model.fit(X_train, y_train)
    return model

def train_lstm_optimized(X_train, y_train, input_shape):
    X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))  # Reshaping for LSTM
    model = Sequential()
    model.add(LSTM(units=50, activation='relu', input_shape=input_shape))
    model.add(Dense(1, activation='sigmoid'))  # Binary classification

    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    model.fit(X_train, y_train, epochs=5, batch_size=32, verbose=0)

    return model
```

Section: 4

This section consists of the optimization and training functions for the KNN, LSTM, Random Forest and SVM. RandomSearchCV, GridSearchCV, SMOTE, Dropout are being used to do optimization.

```python
# General evaluation function
def evaluate_model(model, X_test, y_test):
    # Check if the model is KNN
    if isinstance(model, KNeighborsClassifier):
        y_pred = model.predict(X_test)
        y_pred_class = (y_pred > 0.5).astype(int)  # Assuming binary classification
    # Check if the model is Random Forest
    elif isinstance(model, RandomForestClassifier):
        y_pred = model.predict(X_test)
        y_pred_class = (y_pred > 0.5).astype(int)  # Assuming binary classification
    # Check if the model is SVM
    elif isinstance(model, SVC):
        y_pred = model.predict(X_test)
        y_pred_class = (y_pred > 0.5).astype(int)  # Assuming binary classification
    # Check if the model is LSTM
    elif isinstance(model, Sequential) and isinstance(model.layers[0], LSTM):
        X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))  # Reshaping for LSTM
        y_pred = model.predict(X_test)
        y_pred_class = (y_pred > 0.5).astype(int)  # Assuming binary classification
    else:
        y_pred = model.predict(X_test)
        y_pred_class = (y_pred > 0.5).astype(int)  # Assuming binary classification
    # Extracting metrics from confusion matrix
    cm = confusion_matrix(y_test, y_pred_class)
    tn, fp, fn, tp = cm.ravel()
    p = tp + fn
    n = tn + fp
    accuracy = (tp + tn) / (p + n)
    recall = tp / (tp + fn)
    precision = tp / (tp + fp)
    f1_score = (2 * tp) / (2 * tp + fp + fn)
    tpr = tp / (tp + fn)  # True Positive Rate
    tnr = tn / (tn + fp)  # True Negative Rate
    fpr = fp / (tn + fp)  # False Positive Rate
    fnr = fn / (fn + tp)  # False Negative Rate
    error_rate = (fp + fn) / (p + n)  # Error rate
    bacc = (tpr + tnr) / 2  # Balanced Accuracy
    tss = tpr - fpr  # True Skill Statistic
    hss = (2 * (tp * tn - fp * fn)) / ((tp + fn) * (tn + fp) + (tn + fp) * (tp + fn))  # Heidke Skill Score
    bs = brier_score_loss(y_test, y_pred)  # Brier Score (BS)
    # Brier Skill Score (BSS)
    baseline_prediction = np.mean(y_test)
    bs_baseline = brier_score_loss(y_test, [baseline_prediction] * len(y_test))
    bss = 1 - (bs / bs_baseline)
```

```python
    bss = 1 - (bs / bs_baseline)
    # AUC (Area Under the ROC Curve)
    fpr_roc, tpr_roc, _ = roc_curve(y_test, y_pred)
    auc_score = auc(fpr_roc, tpr_roc)

    result = [accuracy, recall, precision, f1_score, tpr, tnr, fpr, fnr, p, n, error_rate, bacc, tss, hss, bs, bss, auc_score]

    return result
```

Section: 5

This section consists of the model evaluation function where metrics are calculated based on the model given as parameter to the function as different models require different inputs to generate this metrics, this condition is handled over here.

```python
# Function to perform 10-fold cross-validation for a given model type
def cross_validation_evaluation(X, y, model_type):
    # Initialize a stratified k-fold cross-validation object to maintain class distribution
    skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

    # Dictionary to store metrics for each fold
    metrics = {
        'Fold': [],
        'Accuracy': [],
        'Recall': [],
        'Precision': [],
        'F1_Score': [],
        'TPR': [],  # True Positive Rate
        'TNR': [],  # True Negative Rate
        'FPR': [],  # False Positive Rate
        'FNR': [],  # False Negative Rate
        'P': [],    # Number of positive samples
        'N': [],    # Number of negative samples
        'ER': [],   # Error Rate
        'BACC': [], # Balanced Accuracy
        'TSS': [],  # True Skill Statistic
        'HSS': [],  # Heidke Skill Score
        'BS': [],   # Brier Score
        'BSS': [],  # Brier Skill Score
        'AUC': []   # Area Under the Curve
    }
    Fold = 0  # Counter to track fold number

    # Loop through each fold in the cross-validation process
    for train_index, test_index in skf.split(X_train, y_train):
        Fold += 1
        # Split the data into training and testing sets for the current fold
        X_train_fold, X_test_fold = X_train[train_index], X_train[test_index]
        y_train_fold, y_test_fold = y_train[train_index], y_train[test_index]
```

```python
        # Train the appropriate model based on the input model type
        if model_type == "knn":
            model = train_knn_optimized(X_train_fold, y_train_fold)  # Train KNN
            result = evaluate_model(model, X_test_fold, y_test_fold)  # Evaluate KNN
        elif model_type == "lstm":
            model = train_lstm_optimized(X_train_fold, y_train_fold, (1, X_train.shape[1]))  # Train LSTM
            result = evaluate_model(model, X_test_fold, y_test_fold)  # Evaluate LSTM
        elif model_type == "rf":
            model = train_random_forest_optimized(X_train_fold, y_train_fold)  # Train Random Forest
            result = evaluate_model(model, X_test_fold, y_test_fold)  # Evaluate Random Forest
        elif model_type == "svm":
            model = train_svm_optimized(X_train_fold, y_train_fold)  # Train SVM
            result = evaluate_model(model, X_test_fold, y_test_fold)  # Evaluate SVM

        # Append metrics for the current fold
        metrics['Fold'].append(Fold)
        metrics['Accuracy'].append(result[0])
        metrics['Recall'].append(result[1])
        metrics['Precision'].append(result[2])
        metrics['F1_Score'].append(result[3])
        metrics['TPR'].append(result[4])
        metrics['TNR'].append(result[5])
        metrics['FPR'].append(result[6])
        metrics['FNR'].append(result[7])
        metrics['P'].append(result[8])
        metrics['N'].append(result[9])
        metrics['ER'].append(result[10])
        metrics['BACC'].append(result[11])
        metrics['TSS'].append(result[12])
        metrics['HSS'].append(result[13])
        metrics['BS'].append(result[14])
        metrics['BSS'].append(result[15])
        metrics['AUC'].append(result[16])

    # Convert the metrics dictionary into a DataFrame for easier analysis
    metrics_df = pd.DataFrame(metrics)
    return metrics_df
```

```python
# Evaluate models using 10-fold cross-validation for each model type
# The results will include detailed metrics for each fold and each model
knn_metrics = cross_validation_evaluation(X_train, y_train, "knn")  # Evaluate KNN
lstm_metrics = cross_validation_evaluation(X_train, y_train, "lstm")  # Evaluate LSTM
rf_metrics = cross_validation_evaluation(X_train, y_train, "rf")  # Evaluate Random Forest
svm_metrics = cross_validation_evaluation(X_train, y_train, "svm")  # Evaluate SVM
```

Section: 6

This section is having the 10-fold cross validation function, here Stratified K Fold is used to maintain the ratio of classes in the dataset when calculating for different folds and then after for each fold metrics are calculated and then at last the metrics are calculated for each model by calling this function.

```python
# Function to explain the evaluation metrics to users
def print_metrics_intro():
    print("\n" + "=" * 60 + ' " Metrics Overview " ' + "=" * 60 + "\n")
    print("The following metrics will be used to evaluate the models:")
    print("\n1. **Accuracy**: Measures the overall correctness of the model's predictions.")
    print("2. **Recall (Sensitivity)**: Represents the model's ability to identify positive outcomes.")
    print("3. **Precision**: Shows how many of the predicted positive outcomes are actually correct.")
    print("4. **F1-Score**: Combines precision and recall, providing a balance between the two.")
    print("5. **True Positive Rate (TPR)**: The proportion of actual positives correctly identified by the model.")
    print("6. **True Negative Rate (TNR)**: The proportion of actual negatives correctly identified.")
    print("7. **False Positive Rate (FPR)**: The proportion of actual negatives incorrectly classified as positives.")
    print("8. **False Negative Rate (FNR)**: The proportion of actual positives incorrectly classified as negatives.")
    print("9. **P**: Total number of positive samples.")
    print("10. **N**: Total number of negative samples.")
    print("11. **Error Rate (ER)**: The proportion of incorrect predictions made by the model.")
    print("12. **Balanced Accuracy (BACC)**: The average of TPR and TNR, balancing performance on both classes.")
    print("13. **True Skill Statistic (TSS)**: Measures the difference between the proportion of correctly predicted positives and negatives.")
    print("14. **Heidke Skill Score (HSS)**: Evaluates the skill of the model by comparing observed accuracy to expected accuracy.")
    print("15. **Brier Score (BS)**: A metric that quantifies the accuracy of probabilistic predictions.")
    print("16. **Brier Skill Score (BSS)**: Compares the Brier score of the model to a baseline model.")
    print("17. **AUC (Area Under the Curve)**: Measures the model's ability to distinguish between classes based on the ROC curve.")
    print("\nThese metrics will be calculated for each fold in the 10-fold cross-validation process, and the average results across all folds will be presented.")
    print("\nNow, let's proceed with the 10-fold cross-validation results for each model:")

# Call the function to display metrics explanation
print_metrics_intro()

# Print results for each model in tabular format
print("\n\n" + "=" * 65 + ' " 10-Fold Cross Validation Results " ' + "=" * 65)
print("\n\nRandom Forest Results: ")
print(tabulate(rf_metrics.round(3), headers='keys', tablefmt='grid', showindex=False))  # Display Random Forest results
print("\n\n" + "=" * 162)

print("\n\nLSTM Results: ")
print(tabulate(lstm_metrics.round(3), headers='keys', tablefmt='grid', showindex=False))  # Display LSTM results
print("\n\n" + "-" * 162)

print("\n\nKNN Results: ")
print(tabulate(knn_metrics.round(3), headers='keys', tablefmt='grid', showindex=False))  # Display KNN results
print("\n\n" + "-" * 162)

print("\n\nSVM Results: ")
print(tabulate(svm_metrics.round(3), headers='keys', tablefmt='grid', showindex=False))  # Display SVM results
print("\n\n" + "-" * 162)
```

```python
# Train models on the full training dataset
print("Best Parameters for models: ")
rf_model = train_random_forest_optimized(X_train, y_train)  # Train Random Forest
lstm_model = train_lstm_optimized(X_train, y_train, input_shape=(X_train_lstm.shape[1], X_train_lstm.shape[2]))  # Train LSTM
knn_model = train_knn_optimized(X_train, y_train)  # Train KNN
svm_model = train_svm_optimized(X_train, y_train)  # Train SVM

```

Section: 7

This section is having metrics introduction and the tables for the metrics of each fold for all models and also the tabulate package is used for better readability. Then the models are trained on the full dataset.

```python
# Get average metrics for each model using the calculate_average function
# The metrics for Random Forest, LSTM, KNN, and SVM are averaged across all folds
rf_avg = calculate_average(rf_metrics)
lstm_avg = calculate_average(lstm_metrics)
knn_avg = calculate_average(knn_metrics)
svm_avg = calculate_average(svm_metrics)

# Prepare and display the average metrics table
# Creating a DataFrame to present the average metrics of all models side-by-side
avg_table = pd.DataFrame({
    'Metric': knn_metrics.columns,  # Use metric names as rows
    'Random Forest': rf_avg,        # Average metrics for Random Forest
    'LSTM': lstm_avg,               # Average metrics for LSTM
    'KNN': knn_avg,                 # Average metrics for KNN
    'SVM': svm_avg                  # Average metrics for SVM
})
avg_table = avg_table.drop(index="Fold")  # Exclude the 'Fold' column as it is not a metric

# Displaying the summary table with tabular formatting for readability
print("\n\n" + "=" * 40 + ' " Average Metrics of 10 folds for each model " ' + "=" * 40)
print(tabulate(avg_table, headers='keys', tablefmt='grid', showindex=False))

# Function to plot the ROC curve for a given model
# The ROC curve illustrates the trade-off between sensitivity (True Positive Rate) and
# specificity (1 - False Positive Rate) for different classification thresholds.
def plot_roc_curve(model, X_test, y_test, model_name):
    # Obtain the predicted probabilities for the positive class
    if model_name == "LSTM":
        # Reshape the test data to match LSTM's input shape (same as during training)
        X_test_lstm = X_test.reshape(X_test.shape[0], 1, X_test.shape[1])
        y_prob = model.predict(X_test_lstm)  # LSTM model predicts probabilities
    else:
        y_prob = model.predict_proba(X_test)[:, 1]  # For other models, predict_proba gives class probabilities

    # Calculate the False Positive Rate (FPR), True Positive Rate (TPR), and thresholds
    fpr, tpr, _ = roc_curve(y_test, y_prob)

    # Compute the Area Under the Curve (AUC) score
    auc_score = auc(fpr, tpr)
```

```python
    # Plot the ROC curve with AUC score
    plt.figure(figsize=(8, 6))  # Set the figure size
    plt.plot(fpr, tpr, color='red', label=f'{model_name} (AUC = {auc_score:.4f})')  # ROC curve
    plt.plot([0, 1], [0, 1], color='black', linestyle='--')  # Diagonal line for reference
    plt.title(f'ROC Curve - {model_name}')  # Title of the plot
    plt.xlabel('False Positive Rate')  # Label for the x-axis
    plt.ylabel('True Positive Rate')  # Label for the y-axis
    plt.legend(loc='lower right')  # Legend position
    plt.grid(True)  # Display grid for better readability
    plt.show()  # Show the ROC plot

# Plot ROC curves for each model
# This section visualizes the performance of each model through its Receiver Operating Characteristic (ROC) curve.
# The ROC curve plots the true positive rate against the false positive rate, providing a graphical representation of a model's performance.

plot_roc_curve(rf_model, X_test, y_test, "Random Forest")  # Plot ROC for Random Forest model
plot_roc_curve(lstm_model, X_test, y_test, "LSTM")         # Plot ROC for LSTM model
plot_roc_curve(knn_model, X_test, y_test, "KNN")           # Plot ROC for KNN model
plot_roc_curve(svm_model, X_test, y_test, "SVM")           # Plot ROC for SVM model
```

Section: 8

This section is having calculate average function, so that the average of the metrics of each model is calculated for the fold and a average table is also printed as output. Using AUC scores ROC for each model is plotted.

```python
# Generate dynamic insights based on the average metrics
# This function analyzes the average metrics to identify the best and worst-performing models for each metric.
# It also determines the overall best and worst models based on accuracy and provides tailored recommendations.

def generate_dynamic_insights(avg_table):
    # Determine the best and worst models for each metric
    best_model_per_metric = avg_table.iloc[:, 1:].idxmax(axis=1)  # Model with the highest score for each metric
    worst_model_per_metric = avg_table.iloc[:, 1:].idxmin(axis=1)  # Model with the lowest score for each metric

    # Identify the overall best and worst models based on accuracy
    accuracy_column = avg_table[avg_table['Metric'] == 'Accuracy']  # Extract accuracy-related data
    best_overall_model = accuracy_column.iloc[:, 1:].idxmax(axis=1).values[0]  # Model with highest accuracy
    worst_overall_model = accuracy_column.iloc[:, 1:].idxmin(axis=1).values[0]  # Model with lowest accuracy

    # Display insights in a formatted manner
    print("\n" + "=" * 50 + ' " Insights " ' + "=" * 50 + "\n")
    print("1. Best and Worst Models for Each Metric:")
    # Tabular representation of the best and worst models for each metric
    print(tabulate(pd.DataFrame({
        "Metric": avg_table['Metric'],             # Metric names
        "Best Model": best_model_per_metric,       # Best model for each metric
        "Worst Model": worst_model_per_metric      # Worst model for each metric
    }), headers='keys', tablefmt='grid', showindex=False))

    print("\n2. Overall Best and Worst Models Based on Accuracy:")
    # Highlight the best and worst models based on average accuracy
    print(f"   - The overall best model is '{best_overall_model}', which achieved the highest average accuracy.")
    print(f"   - The overall worst model is '{worst_overall_model}', which achieved the lowest average accuracy.\n")

    # Provide model-specific recommendations
    print("3. Recommendations:")

    # Tailored recommendations for the best-performing model
    if best_overall_model == 'Random Forest':
        print("   - Random Forest performed best. Consider optimizing hyperparameters for even better performance.")
    elif best_overall_model == 'KNN':
        print("   - KNN performed best. Experiment with different values of K or apply feature scaling.")
    elif best_overall_model == 'LSTM':
        print("   - LSTM performed best. Try different architectures or include more data.")
    elif best_overall_model == 'SVM':
        print("   - SVM performed best. Explore different kernel functions or adjust the regularization parameter (C).")
```

```python
    # Tailored recommendations for the worst-performing model
    if worst_overall_model == 'Random Forest':
        print("   - Random Forest performed worst. Consider reducing the number of trees or adjusting the tree depth.")
    elif worst_overall_model == 'KNN':
        print("   - KNN performed worst. Optimize K or explore different distance metrics.")
    elif worst_overall_model == 'LSTM':
        print("   - LSTM performed worst. Adjust network layers or explore other sequence models.")
    elif worst_overall_model == 'SVM':
        print("   - SVM performed worst. Try different kernel functions or tune the regularization parameter (C).")
    print('''   - For all models, consider hyperparameter tuning and better feature selection for improved results.''')

    # General advice applicable to all models
    print("4. Oberservations: ")
    print('''While SVM achieved the highest accuracy during evaluation, it exhibited signs of overfitting,
    which may limit its generalizability in real-world scenarios. On the other hand, LSTM demonstrated robust
    performance with better adaptability, making it the preferred choice when real-world application and reliability are key priorities.''')
    print("=" * 162)

# Call the function to generate insights based on the evaluation results
generate_dynamic_insights(avg_table)

# Closing message to thank the user for using the program
print("=" * 162)
print("Thank you for using this program!")
print("Hope the results were insightful. Feel free to revisit the program anytime for further analysis.")
print("If you have any questions or need assistance, don't hesitate to reach out!")
print("=" * 162)
```

Section: 9

This section is generating dynamic insights based on the performance of the models. Mostly due to lack of good training and test dataset, the models are overfitting but using early stopping and dropout, LSTM performs good. Rest of the others are mostly overfitting. Also, recommendations are provided for improving the outcome from the models and at last thankyou message is sent.

SCREENSHOTS OF OUTPUT

```
================================================================================================================================================================
Welcome to the Banknote Authentication Model Evaluation Program!

This program allows you to:
1. Load and preprocess the banknote dataset.
2. Train and evaluate four different models:
   - Random Forest
   - Deep Learning: LSTM (Long Short Term Memory)
   - Algorithm: KNN (K-Nearest Neighbor)
   - Algorithm: SVM (Support Vector Machine)
3. Generate detailed insights and recommendations based on the evaluation results.

**IMPORTANT**: For optimal readability and proper formatting of outputs, please use the terminal in full-screen view.

You will receive:
   - A description of the dataset.
   - Information about the dataset, including data types, missing values, etc.
   - A preview of the first few records in the dataset (head).
   - Metric Table showing the results of 10-fold cross-validation for each model.
   - Table displaying the average metrics across all 10 folds for each model.
   - Dynamic insights and recommendations based on the performance of models.

**Graphs you'll see**:
   - Distribution of the target variable (Target)
   - Correlation matrix heatmap showing relationships between features
   - Pair plot visualizing data relationships with the target variable
   - Visualizing the model's performance with ROC curves for each model.

Press 'Enter' to continue with the analysis...

================================================================================================================================================================
```

Figure: 1

At first, welcome message is printed along with the details about the program and what user can expect from the program. User is prompted to press “ENTER” to continue for the execution of the program.

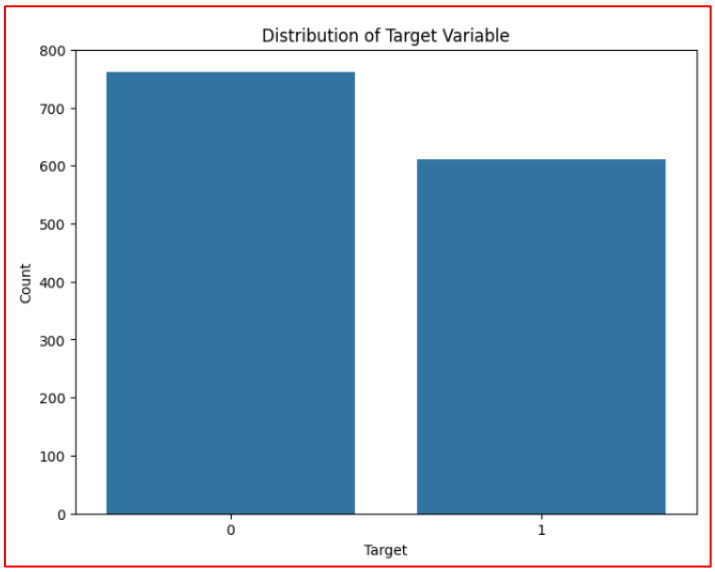

Figure: 2

This is the distribution of the target variable in a bar graph, having two classes as we are doing binary classification.

```
Checking for Data Imbalance:
    - Number of Positive Outcomes: 762
    - Percentage of Positive Outcomes: 55.54%
    - Number of Negative Outcomes: 610
    - Percentage of Negative Outcomes: 44.46%


=========================================================
```

Figure: 3

Data imbalance is checked and reported, we found the details about the positive and negative outcomes in the dataset. From here we can find that the dataset is not much imbalanced. Still its handled in the code.

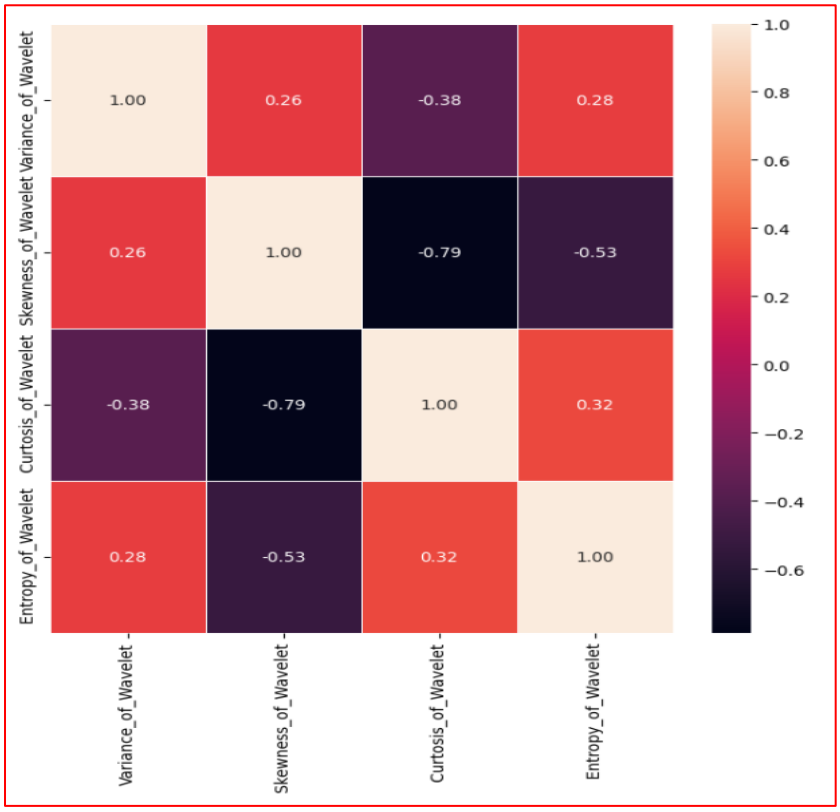

Figure: 4

It is the correlation heatmap demonstrating the relationship between different attributes of the dataset.

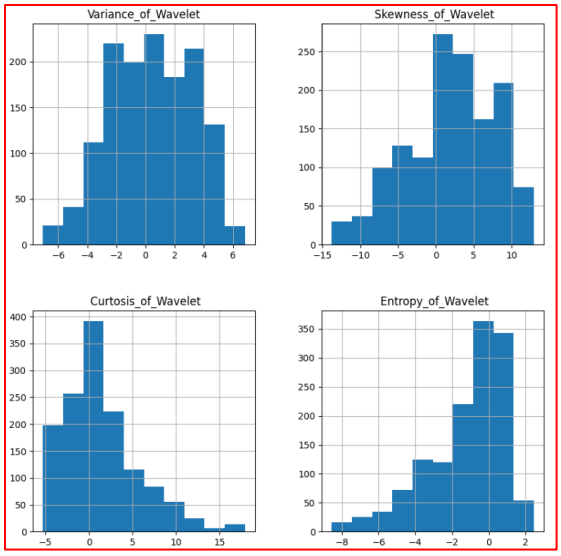

Figure: 5

These are the histograms for visualization of distributions of features from the dataset.

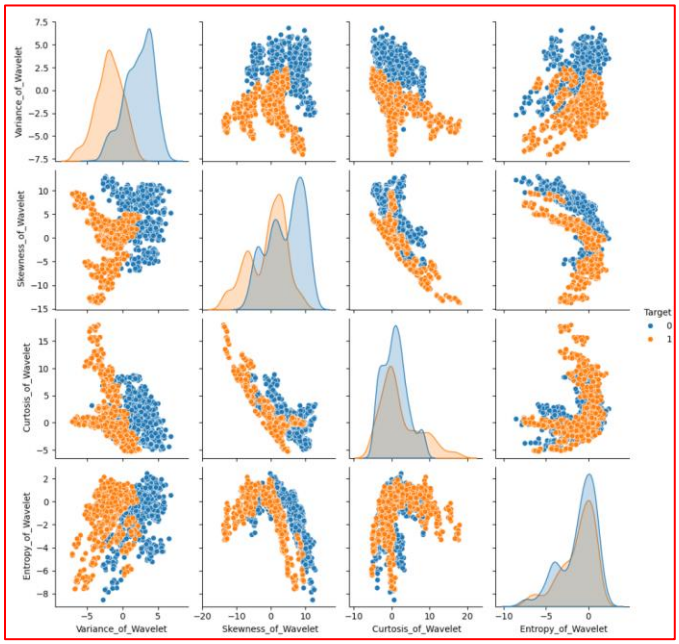

Figure: 6

These are the pair plots to visualize feature relationships with the Target Variable

```
The model optimization and training process may take a few minutes.
It might feel like it's taking forever, but don't worry—it will complete successfully! :)


Optimization Parameters for models:
Parameters for KNN: {'n_neighbors': 3, 'p': 2, 'weights': 'uniform'}
Parameters for KNN: {'n_neighbors': 3, 'p': 2, 'weights': 'uniform'}
Parameters for KNN: {'n_neighbors': 3, 'p': 2, 'weights': 'uniform'}
Parameters for KNN: {'n_neighbors': 3, 'p': 2, 'weights': 'uniform'}
Parameters for KNN: {'n_neighbors': 3, 'p': 2, 'weights': 'uniform'}
Parameters for KNN: {'n_neighbors': 3, 'p': 2, 'weights': 'uniform'}
Parameters for KNN: {'n_neighbors': 3, 'p': 2, 'weights': 'uniform'}
Parameters for KNN: {'n_neighbors': 3, 'p': 2, 'weights': 'uniform'}
Parameters for KNN: {'n_neighbors': 3, 'p': 2, 'weights': 'uniform'}
Parameters for KNN: {'n_neighbors': 3, 'p': 2, 'weights': 'uniform'}
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 155ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 93ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 148ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 100ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 95ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 104ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 102ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 145ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 96ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 94ms/step
Parameters for Random Forest: {'n_estimators': 100, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_depth': None}
Parameters for Random Forest: {'n_estimators': 100, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_depth': None}
Parameters for Random Forest: {'n_estimators': 50, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_depth': None}
Parameters for Random Forest: {'n_estimators': 50, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_depth': None}
Parameters for Random Forest: {'n_estimators': 50, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_depth': 10}
Parameters for Random Forest: {'n_estimators': 50, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_depth': None}
Parameters for Random Forest: {'n_estimators': 50, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_depth': None}
Parameters for Random Forest: {'n_estimators': 100, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_depth': None}
Parameters for Random Forest: {'n_estimators': 100, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_depth': None}
Parameters for Random Forest: {'n_estimators': 100, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_depth': None}
Parameters for SVM: {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
Parameters for SVM: {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
Parameters for SVM: {'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}
Parameters for SVM: {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
Parameters for SVM: {'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}
Parameters for SVM: {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
Parameters for SVM: {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
```

Figure: 7

User is asked to wait for a while as the training and finding optimal combination for all model is bit time consuming. It will be completed in few minutes for sure.

```
============================================================ " Metrics Overview " ============================================================

The following metrics will be used to evaluate the models:

1. **Accuracy**: Measures the overall correctness of the model's predictions.
2. **Recall (Sensitivity)**: Represents the model's ability to identify positive outcomes.
3. **Precision**: Shows how many of the predicted positive outcomes are actually correct.
4. **F1-Score**: Combines precision and recall, providing a balance between the two.
5. **True Positive Rate (TPR)**: The proportion of actual positives correctly identified by the model.
6. **True Negative Rate (TNR)**: The proportion of actual negatives correctly identified.
7. **False Positive Rate (FPR)**: The proportion of actual negatives incorrectly classified as positives.
8. **False Negative Rate (FNR)**: The proportion of actual positives incorrectly classified as negatives.
9. **P**: Total number of positive samples.
10. **N**: Total number of negative samples.
11. **Error Rate (ER)**: The proportion of incorrect predictions made by the model.
12. **Balanced Accuracy (BACC)**: The average of TPR and TNR, balancing performance on both classes.
13. **True Skill Statistic (TSS)**: Measures the difference between the proportion of correctly predicted positives and negatives.
14. **Heidke Skill Score (HSS)**: Evaluates the skill of the model by comparing observed accuracy to expected accuracy.
15. **Brier Score (BS)**: A metric that quantifies the accuracy of probabilistic predictions.
16. **Brier Skill Score (BSS)**: Compares the Brier score of the model to a baseline model.
17. **AUC (Area Under the Curve)**: Measures the model's ability to distinguish between classes based on the ROC curve.

These metrics will be calculated for each fold in the 10-fold cross-validation process, and the average results across all folds will be presented.
```

Figure: 8

After successful training and optimization of models, user is provided with the details of the metrics which are calculated in the program.

Now, let's proceed with the 10-fold cross-validation results for each model:

========== " 10-Fold Cross Validation Results " ==========

Random Forest Results:

Fold	Accuracy	Recall	Precision	F1_Score	TPR	TNR	FPR	FNR	P	N	ER	BACC	TSS	HSS	BS	BSS	AUC
1	0.991	1	0.98	0.99	1	0.984	0.016	0	48	62	0.009	0.992	0.984	0.984	0.009	0.963	0.992
2	1	1	1	1	1	1	0	0	48	62	0	1	1	1	0	1	1
3	0.982	0.958	1	0.979	0.958	1	0	0.042	48	62	0.018	0.979	0.958	0.958	0.018	0.926	0.979
4	0.991	0.979	1	0.989	0.979	1	0	0.021	48	62	0.009	0.99	0.979	0.979	0.009	0.963	0.99
5	1	1	1	1	1	1	0	0	49	61	0	1	1	1	0	1	1
6	0.982	1	0.961	0.98	1	0.967	0.033	0	49	61	0.018	0.984	0.967	0.967	0.018	0.926	0.984
7	0.991	1	0.98	0.99	1	0.984	0.016	0	49	61	0.009	0.992	0.984	0.984	0.009	0.963	0.992
8	0.991	1	0.98	0.99	1	0.984	0.016	0	48	61	0.009	0.992	0.984	0.984	0.009	0.963	0.992
9	0.991	0.979	1	0.989	0.979	1	0	0.021	48	61	0.009	0.99	0.979	0.979	0.009	0.963	0.99
10	0.982	0.979	0.979	0.979	0.979	0.984	0.016	0.021	48	61	0.018	0.981	0.963	0.963	0.018	0.926	0.981

Figure: 9

The table of Random Forest for 10 folds and metric values are displayed to user.

LSTM Results:

Fold	Accuracy	Recall	Precision	F1_Score	TPR	TNR	FPR	FNR	P	N	ER	BACC	TSS	HSS	BS	BSS	AUC
1	0.918	0.938	0.882	0.909	0.938	0.903	0.097	0.062	48	62	0.082	0.92	0.841	0.841	0.118	0.522	0.975
2	0.882	0.854	0.872	0.863	0.854	0.903	0.097	0.146	48	62	0.118	0.879	0.757	0.757	0.115	0.531	0.964
3	0.9	0.854	0.911	0.882	0.854	0.935	0.065	0.146	48	62	0.1	0.895	0.79	0.79	0.119	0.518	0.965
4	0.864	0.875	0.824	0.848	0.875	0.855	0.145	0.125	48	62	0.136	0.865	0.73	0.73	0.131	0.469	0.947
5	0.873	0.796	0.907	0.848	0.796	0.934	0.066	0.204	49	61	0.127	0.865	0.73	0.73	0.122	0.507	0.976
6	0.773	0.633	0.816	0.713	0.633	0.885	0.115	0.367	49	61	0.227	0.759	0.518	0.518	0.156	0.367	0.894
7	0.882	0.878	0.86	0.869	0.878	0.885	0.115	0.122	49	61	0.118	0.881	0.763	0.763	0.115	0.533	0.967
8	0.899	0.833	0.93	0.879	0.833	0.951	0.049	0.167	48	61	0.101	0.892	0.784	0.784	0.126	0.49	0.961
9	0.908	0.917	0.88	0.898	0.917	0.902	0.098	0.083	48	61	0.092	0.909	0.818	0.818	0.118	0.52	0.964
10	0.899	0.854	0.911	0.882	0.854	0.934	0.066	0.146	48	61	0.101	0.894	0.789	0.789	0.108	0.562	0.976

Figure: 10

The table of LSTM for 10 folds and metric values are displayed to user.

KNN Results:

Fold	Accuracy	Recall	Precision	F1_Score	TPR	TNR	FPR	FNR	P	N	ER	BACC	TSS	HSS	BS	BSS	AUC
1	0.982	1	0.96	0.98	1	0.968	0.032	0	48	62	0.018	0.984	0.968	0.968	0.018	0.926	0.984
2	1	1	1	1	1	1	0	0	48	62	0	1	1	1	0	1	1
3	1	1	1	1	1	1	0	0	48	62	0	1	1	1	0	1	1
4	1	1	1	1	1	1	0	0	48	62	0	1	1	1	0	1	1
5	1	1	1	1	1	1	0	0	49	61	0	1	1	1	0	1	1
6	1	1	1	1	1	1	0	0	49	61	0	1	1	1	0	1	1
7	1	1	1	1	1	1	0	0	49	61	0	1	1	1	0	1	1
8	1	1	1	1	1	1	0	0	48	61	0	1	1	1	0	1	1
9	1	1	1	1	1	1	0	0	48	61	0	1	1	1	0	1	1
10	1	1	1	1	1	1	0	0	48	61	0	1	1	1	0	1	1

Figure: 11

The table of KNN for 10 folds and metric values are displayed to user.

SVM Results:

Fold	Accuracy	Recall	Precision	F1_Score	TPR	TNR	FPR	FNR	P	N	ER	BACC	TSS	HSS	BS	BSS	AUC
1	1	1	1	1	1	1	0	0	48	62	0	1	1	1	0	1	1
2	1	1	1	1	1	1	0	0	48	62	0	1	1	1	0	1	1
3	1	1	1	1	1	1	0	0	48	62	0	1	1	1	0	1	1
4	1	1	1	1	1	1	0	0	48	62	0	1	1	1	0	1	1
5	1	1	1	1	1	1	0	0	49	61	0	1	1	1	0	1	1
6	1	1	1	1	1	1	0	0	49	61	0	1	1	1	0	1	1
7	1	1	1	1	1	1	0	0	49	61	0	1	1	1	0	1	1
8	1	1	1	1	1	1	0	0	48	61	0	1	1	1	0	1	1
9	1	1	1	1	1	1	0	0	48	61	0	1	1	1	0	1	1
10	1	1	1	1	1	1	0	0	48	61	0	1	1	1	0	1	1

Figure: 12

The table of SVM for 10 folds and metric values are displayed to user.

```
Best Parameters for models:
Parameters for Random Forest: {'n_estimators': 100, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_depth': None}
Parameters for KNN: {'n_neighbors': 3, 'p': 2, 'weights': 'uniform'}
Parameters for SVM: {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}


======================================== " Average Metrics of 10 folds for each model " ========================================
```

Metric	Random Forest	LSTM	KNN	SVM
Accuracy	0.99	0.88	0.998	1
Recall	0.99	0.843	1	1
Precision	0.988	0.879	0.996	1
F1_Score	0.989	0.859	0.998	1
TPR	0.99	0.843	1	1
TNR	0.99	0.909	0.997	1
FPR	0.01	0.091	0.003	0
FNR	0.01	0.157	0	0
P	48.3	48.3	48.3	48.3
N	61.4	61.4	61.4	61.4
ER	0.01	0.12	0.002	0
BACC	0.99	0.876	0.998	1
TSS	0.98	0.752	0.997	1
HSS	0.98	0.752	0.997	1
BS	0.01	0.123	0.002	0
BSS	0.959	0.502	0.993	1
AUC	0.99	0.959	0.998	1

Figure: 13

The table of Average metrics of all models for 10 folds are displayed to user.

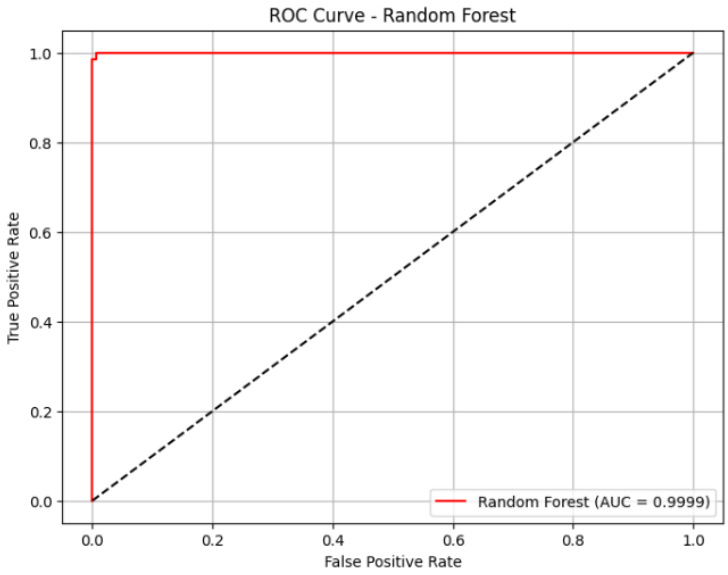

Figure: 14

The ROC curve for AUC of Random Forest is plotted and displayed to the user.

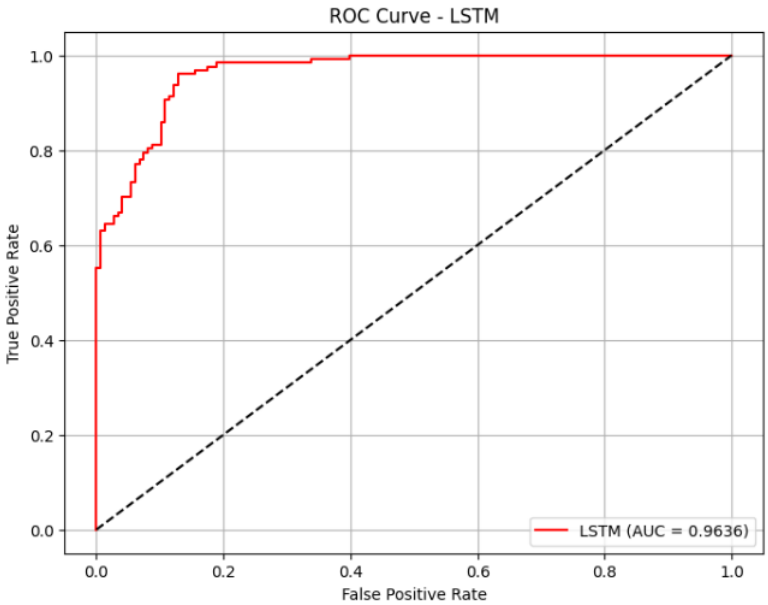

Figure: 15

The ROC curve for AUC of LSTM is plotted and displayed to the user.

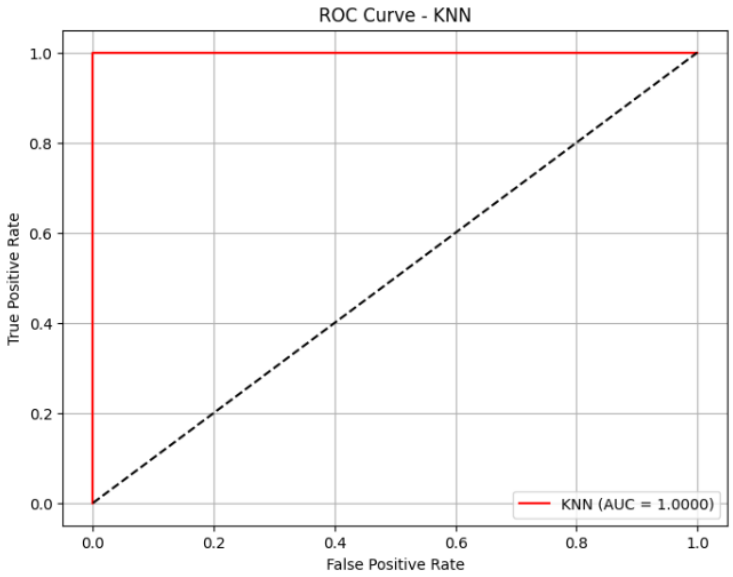

Figure: 16

The ROC curve for AUC of KNN is plotted and displayed to the user.

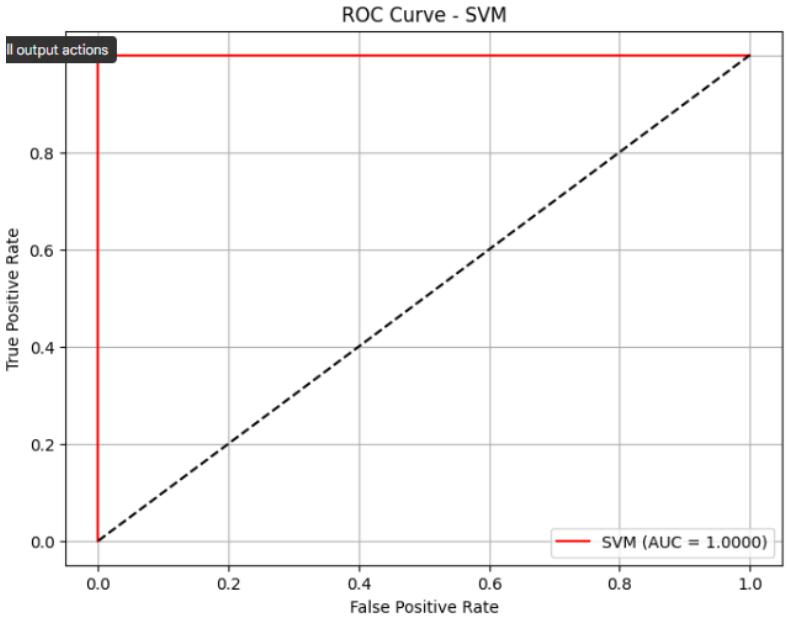

Figure: 17

The ROC curve for AUC of SVM is plotted and displayed to the user.

```
================================================= " Insights " =================================================

1. Best and Worst Models for Each Metric:
+-----------+---------------+---------------+
| Metric    | Best Model    | Worst Model   |
+===========+===============+===============+
| Accuracy  | SVM           | LSTM          |
+-----------+---------------+---------------+
| Recall    | KNN           | LSTM          |
+-----------+---------------+---------------+
| Precision | SVM           | LSTM          |
+-----------+---------------+---------------+
| F1_Score  | SVM           | LSTM          |
+-----------+---------------+---------------+
| TPR       | KNN           | LSTM          |
+-----------+---------------+---------------+
| TNR       | SVM           | LSTM          |
+-----------+---------------+---------------+
| FPR       | LSTM          | SVM           |
+-----------+---------------+---------------+
| FNR       | LSTM          | KNN           |
+-----------+---------------+---------------+
| P         | Random Forest | Random Forest |
+-----------+---------------+---------------+
| N         | Random Forest | Random Forest |
+-----------+---------------+---------------+
| ER        | LSTM          | SVM           |
+-----------+---------------+---------------+
| BACC      | SVM           | LSTM          |
+-----------+---------------+---------------+
| TSS       | SVM           | LSTM          |
+-----------+---------------+---------------+
| HSS       | SVM           | LSTM          |
+-----------+---------------+---------------+
| BS        | LSTM          | SVM           |
+-----------+---------------+---------------+
| BSS       | SVM           | LSTM          |
+-----------+---------------+---------------+
| AUC       | SVM           | LSTM          |
+-----------+---------------+---------------+
```

```
2. Overall Best and Worst Models Based on Accuracy:
   - The overall best model is 'SVM', which achieved the highest average accuracy.
   - The overall worst model is 'LSTM', which achieved the lowest average accuracy.

3. Recommendations:
   - SVM performed best. Explore different kernel functions or adjust the regularization parameter (C).
   - LSTM performed worst. Adjust network layers or explore other sequence models.
   - For all models, consider hyperparameter tuning and better feature selection for improved results.
4. Oberservations:
While SVM achieved the highest accuracy during evaluation, it exhibited signs of overfitting,
    which may limit its generalizability in real-world scenarios. On the other hand, LSTM demonstrated robust
    performance with better adaptability, making it the preferred choice when real-world application and reliability are key priorities.
================================================================================================================================================================================================================
================================================================================================================================================================================================================
Thank you for using this program!
Hope the results were insightful. Feel free to revisit the program anytime for further analysis.
If you have any questions or need assistance, don't hesitate to reach out!
================================================================================================================================================================================================================
```

Figure: 18

Dynamic Insights are generated as best and worst model for each metric based on the performance of the models, yet SVM has best accuracy and AUC, here it is considered as best model but in real world it is overfitting and is not considered a good sign for any model, so if ever required in real world LSTM would be a better option or enhance these rest of models to improve their effectiveness on the real-world scenarios.

CONCLUSION

The performance of four models, namely the Random Forest, LSTM, KNN, and SVM, have been analyzed in this work with respect to the task of banknote authentication. Among all, SVM exhibited the maximum accuracy and AUC from the tenfold cross-validation. However, its capability for generalization on unseen samples is dubious because of signs of overfitting. On the other hand, LSTM's performance and adaptability could be a better choice for deployment in the real world.

Model improvement through hyperparameter tuning and better feature selection in real-life applications is recommended. In addition, alternative approaches or refinement of the existing models may lead to even better results in dynamic environments.

SOURCE CODE AND REPOSITORY

- The source code (**.py** file), (**.ipynb** file) and data sets (**.csv** files) will be attached to the zip file.

- **Link for GitHub Repository:** https://github.com/Yash3561/Final_Project_DM