Name: Deepesh Singh Tomar
NJIT UCID: dt394 Email Id: dt394@njit.edu
Date: 21th April 2024
Professor: Dr. Yasser Abduallah
CS 634104

Data Mining Final Project Report

Trojan Detection

Abstract:

This paper presents a comprehensive machine learning pipeline that uses a network traffic dataset to detect Trojan activity. The method makes use of a number of machine learning methods, such as Random Forest, Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and Long Short-Term Memory (LSTM) networks. The procedure is described in the document, starting with data collection and preprocessing and ending with algorithm implementation and performance assessment..

Introduction:

Cyber risks such as Trojan infections are becoming more common, thus it's important to have efficient detection systems that can quickly and correctly identify them. This paper describes how to use a machine learning pipeline that uses many classification methods to extract Trojan activity information from network traffic data. Different performance indicators are used to compare the efficacy of each algorithm.

Data Acquisition: The study employs data named "Trojan_Detection.csv," which consists of aspects of network traffic that may indicate malicious or benign activity. Flow IDs, source and destination IP addresses, timestamps, and class labels designating the type of communication (Trojan or benign) are examples of attributes.

Data Preprocessing:
The steps in preprocessing include:

1. Data cleaning is the process of eliminating irrelevant columns from the model that don't improve its predictive power, such as "Flow ID," "Source IP," "Destination IP," and "Timestamp."
2. 'Trojan' and 'Benign' are examples of categorical labels that are transformed into a binary format for processing through feature encoding.
3. Feature selection improves model performance and lowers overfitting by keeping features that are highly linked with the target variable. This is done by using correlation analysis.
4. Data normalization: To guarantee consistency and enhance algorithm convergence, features are standardized to have a zero mean and a unit variance.
5. Train-Test Split: To facilitate model validation, the dataset is split into training and testing sets.

Algorithm Implementation:
Four different machine learning algorithms are implemented by the script:

- Random Forest
- Support Vector Machine (SVM)
- K-Nearest Neighbors (KNN)
- Long Short-Term Memory (LSTM) Network

The dataset is used to train each model, and GridSearchCV is used to fine-tune the hyperparameters to the ideal values. In order to capture temporal dependencies in network traffic that other models might miss, LSTM—a model built to handle sequence data—is also incorporated.

Explanation of the Process:

- **Training of Models:** The preprocessed training dataset is used to train each model.
- **Model Evaluation:** To generalize the performance of the models and reduce the impact of any imbalance in the dataset, Stratified K-Fold cross-validation is used.
- **Metrics Calculation:** To evaluate the effectiveness of each model, performance metrics for each test fold are computed, including True Positive Rate (TPR), False Positive Rate (FPR), Precision, F1 Measure, and others.
- **ROC Curve Analysis:** To illustrate each model's capacity for diagnosis, the Receiver Operating Characteristic (ROC) curve is plotted.

Here are some screenshots of the Python code:

Read transactions from a CSV file and convert them into a list of transactions.

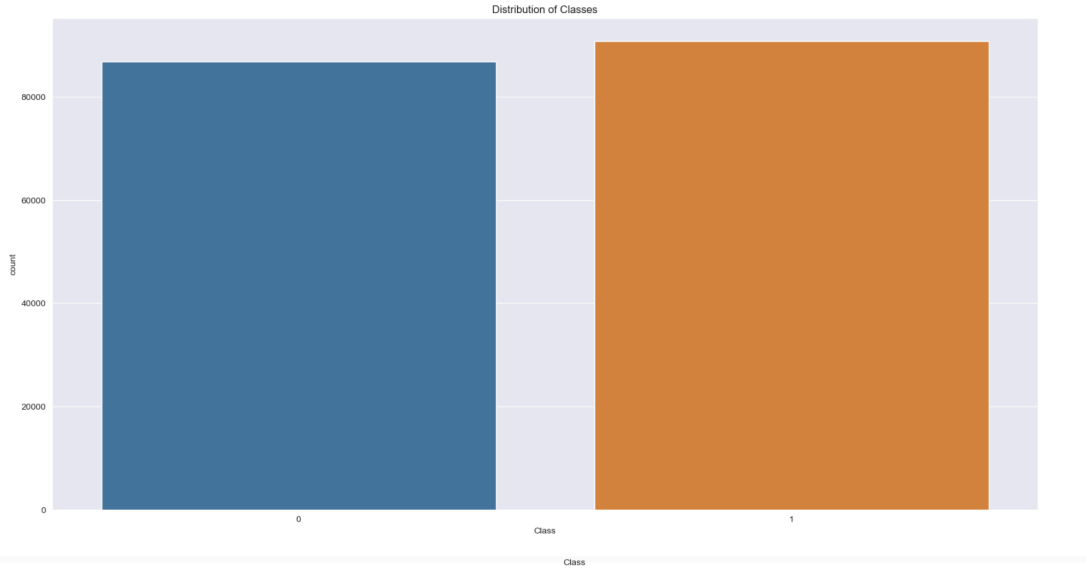
```
In [2]: df1 = pd.read_csv("Trojan_Detection.csv")
        df1.head()
        df1.describe()
```

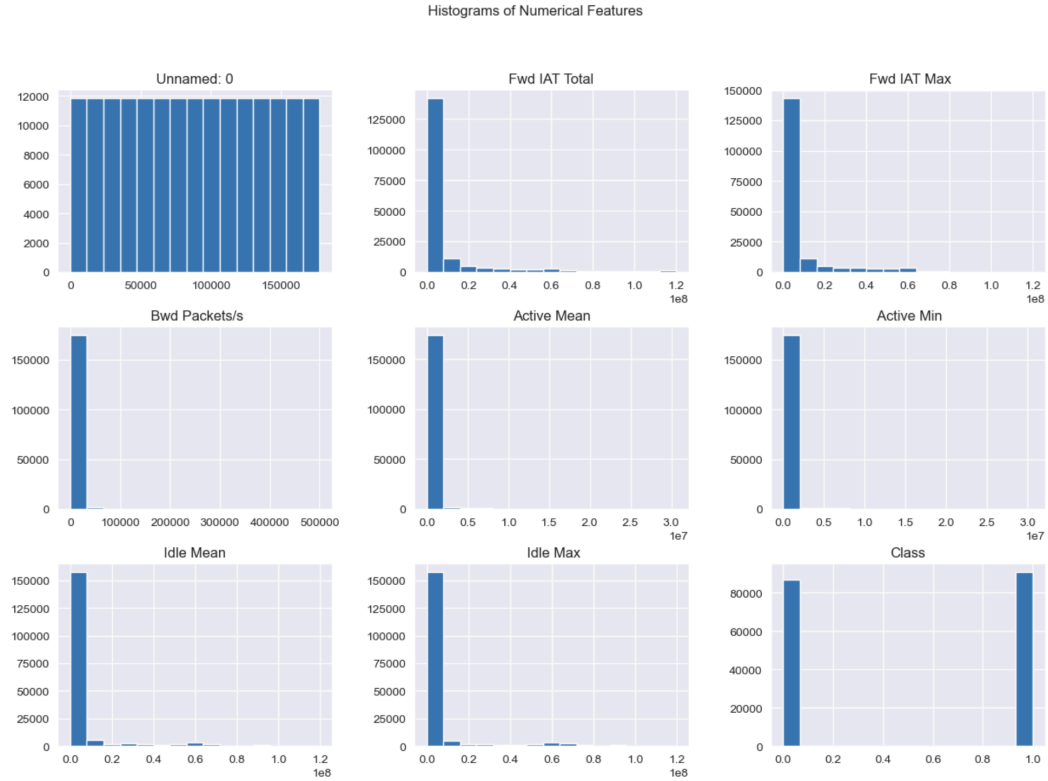
Out[2]:

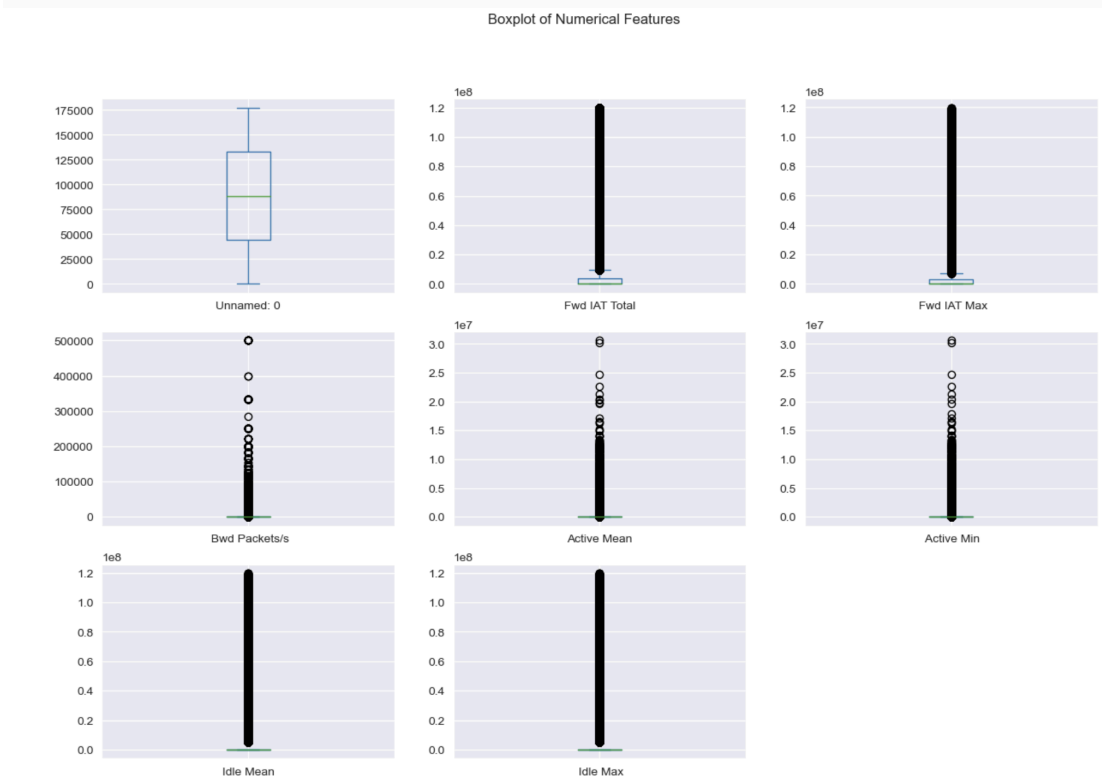
	Unnamed: 0	Source Port	Destination Port	Protocol	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max
count	177482.000000	177482.000000	177482.000000	177482.000000	1.774820e+05	177482.000000	177482.000000	177482.000000	1.774820e+05	177482.000000
mean	88740.500000	38633.298729	5763.328969	8.295917	1.163310e+07	6.743929	9.908261	811.919496	1.023884e+04	230.168969
std	51234.784576	18435.832044	15155.981517	4.540501	2.302798e+07	29.655515	106.401351	6055.745405	1.517666e+05	392.025958
min	0.000000	0.000000	0.000000	0.000000	2.000000e+00	1.000000	0.000000	0.000000	0.000000e+00	0.000000
25%	44370.250000	34156.000000	80.000000	6.000000	4.690050e+04	1.000000	0.000000	0.000000	0.000000e+00	0.000000
50%	88740.500000	43105.500000	443.000000	6.000000	4.854025e+05	2.000000	1.000000	31.000000	3.100000e+01	31.000000
75%	133110.750000	52135.000000	443.000000	6.000000	1.062035e+07	5.000000	4.000000	451.000000	4.950000e+02	355.000000
max	177481.000000	65530.000000	65267.000000	17.000000	1.199991e+08	4502.000000	12951.000000	846614.000000	1.868497e+07	1460.000000

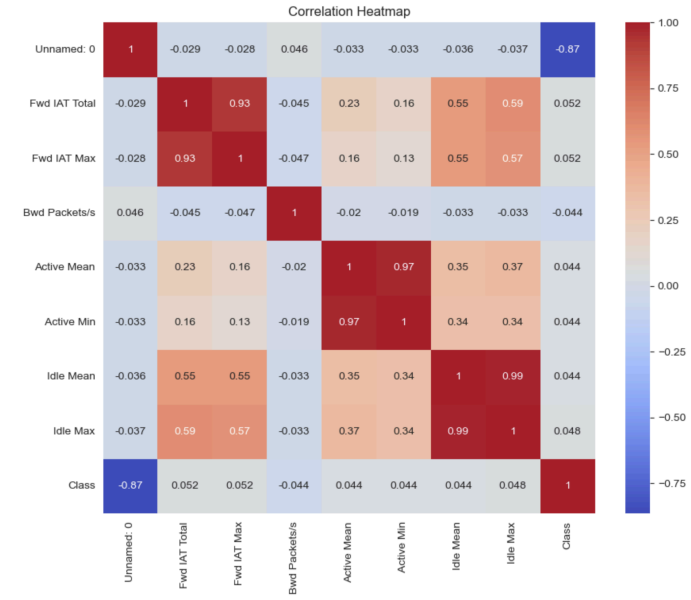
8 rows × 81 columns

Different Visualizations:-






Boxplot of Numerical Features
Unnamed: 0
Fwd IAT Total
Fwd IAT Max
Bwd Packets/s
Active Mean
Active Min
Idle Mean
Idle Max


Correlation Heatmap
Unnamed: 0
Fwd IAT Total
Fwd IAT Max
Bwd Packets/s
Active Mean
Active Min
Idle Mean
Idle Max
Class

Code Snippets:-

In [20]:
```python
from sklearn.metrics import confusion_matrix

def calculate_performance_metrics(y_true, y_pred):
    TN, FP, FN, TP = confusion_matrix(y_true, y_pred).ravel()

    TPR = TP / (TP + FN)
    TNR = TN / (TN + FP)
    PPV = TP / (TP + FP)
    NPV = TN / (TN + FN)
    FPR = FP / (TN + FP)
    FNR = FN / (TP + FN)
    FDR = FP / (TP + FP)
    ACC = (TP + TN) / (TP + FP + FN + TN)
    F1 = 2 * TP / (2 * TP + FP + FN)
    BACC = (TPR + TNR) / 2
    TSS = TPR - FPR
    HSS = 2 * (TP * TN - FP * FN) / ((TP + FN) * (FN + TN) + (TP + FP) * (FP + TN))

    return {
        'Sensitivity': TPR,
        'Specificity': TNR,
        'Precision': PPV,
        'NPV': NPV,
        'FPR': FPR,
        'FNR': FNR,
        'FDR': FDR,
        'Accuracy': ACC,
        'F1': F1,
        'BACC': BACC,
        'TSS': TSS,
        'HSS': HSS,

    }
```

In [22]:
```python
import numpy as np
from sklearn.model_selection import KFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB

# Initialize KFold and models
kf = KFold(n_splits=10, shuffle=True, random_state=42)

model_list = [KNeighborsClassifier(n_neighbors=5), RandomForestClassifier(n_estimators=100, random_state=42), Gaussi

# Initialize a list to store the models' accuracy for each fold
models_performance_accumulator = []

# Iterate through each fold for cross-validation
for fold_number, (train_indices, test_indices) in enumerate(kf.split(X), 1):
    # Split data into training and test subsets for this fold
    training_features = X.iloc[train_indices]
    test_features = X.iloc[test_indices]
    training_labels = y.iloc[train_indices]
    test_labels = y.iloc[test_indices]

    # Store performance metrics for each model during this fold
    single_fold_performance = []

    for individual_model in model_list:
        individual_model.fit(training_features, training_labels)
        predicted_labels = individual_model.predict(test_features)
        metrics_result = calculate_performance_metrics(test_labels, predicted_labels)
        single_fold_performance.append(metrics_result)
    models_performance_accumulator.append(single_fold_performance)
```

In [23]:
```
from sklearn.model_selection import KFold
from sklearn.metrics import confusion_matrix, accuracy_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Bidirectional, Dense
from tensorflow.keras.utils import to_categorical

number_of_splits = 10
kfold_validator = KFold(n_splits=number_of_splits, shuffle=True, random_state=42)

# List to accumulate the performance metrics for each fold
models_performance_accumulation = []

for fold_number, (train_indices, test_indices) in enumerate(kfold_validator.split(X), 1):
    features_for_training, features_for_testing = X.iloc[train_indices], X.iloc[test_indices]
    labels_for_training, labels_for_testing = y.iloc[train_indices], y.iloc[test_indices]
    features_for_training_reshaped = features_for_training.values.reshape(features_for_training.shape[0], 1, feature
    features_for_testing_reshaped = features_for_testing.values.reshape(features_for_testing.shape[0], 1, features_f
    # Encoding the labels as categorical (one-hot encoding)
    labels_for_training_encoded = to_categorical(labels_for_training)
    labels_for_testing_encoded = to_categorical(labels_for_testing)
    # Create a Sequential model and add bidirectional LSTM layer and dense layer
    lstm_model = Sequential()
    lstm_model.add(Bidirectional(LSTM(units=64, input_shape=(1, features_for_training.shape[1]))))
    lstm_model.add(Dense(labels_for_training_encoded.shape[1], activation='softmax'))
    lstm_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    # Train the LSTM model with the training data
    lstm_model.fit(features_for_training_reshaped, labels_for_training_encoded, epochs=10, batch_size=32, verbose=0)
    # Predicting on the test data using the trained model
    predictions_probabilities = lstm_model.predict(features_for_testing_reshaped)
    predictions = np.argmax(predictions_probabilities, axis=-1)
    # Calculate the accuracy and store it in the accumulator
    accuracy_of_model = accuracy_score(labels_for_testing, predictions)
    models_performance_accumulation.append(accuracy_of_model)
mean_accuracy = np.mean(models_performance_accumulation)
print(f"Bidirectional LSTM Model Mean Accuracy across folds: {mean_accuracy}")
```

```
555/555 [==============================] - 0s 301us/step
555/555 [==============================] - 0s 289us/step
555/555 [==============================] - 0s 292us/step
555/555 [==============================] - 0s 287us/step
555/555 [==============================] - 0s 295us/step
555/555 [==============================] - 0s 295us/step
555/555 [==============================] - 0s 288us/step
555/555 [==============================] - 0s 286us/step
555/555 [==============================] - 0s 289us/step
555/555 [==============================] - 0s 289us/step
Bidirectional LSTM Model Mean Accuracy across folds: 0.569094379170806
```

Bidirectional LSTM Model

In [30]:
```
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import confusion_matrix
from keras.models import Sequential
from keras.layers import LSTM, Dense

# K-Fold setup
n_splits = 10
kf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=42)

# List to store metrics for each model
lstm_metrics_list = []

for train_index, test_index in kf.split(X, y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Preparing input for LSTM (assuming features are already scaled/normalized)
    X_train_lstm = np.expand_dims(X_train, axis=-1)  # Adding a dimension for LSTM compatibility
    X_test_lstm = np.expand_dims(X_test, axis=-1)

    # LSTM model setup
    model = Sequential()
    model.add(LSTM(64, input_shape=(X_train_lstm.shape[1], X_train_lstm.shape[2])))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

    model.fit(X_train_lstm, y_train, epochs=10, batch_size=32, verbose=0)

    predictions = model.predict(X_test_lstm) > 0.5
    performance_metrics = calculate_performance_metrics(y_test, predictions)
    lstm_metrics_list.append(performance_metrics)
```

```
555/555 [==============================] - 0s 591us/step
555/555 [==============================] - 0s 627us/step
555/555 [==============================] - 0s 527us/step
555/555 [==============================] - 1s 511us/step
555/555 [==============================] - 0s 514us/step
555/555 [==============================] - 0s 530us/step
555/555 [==============================] - 0s 510us/step
555/555 [==============================] - 0s 517us/step
555/555 [==============================] - 0s 544us/step
555/555 [==============================] - 0s 537us/step
```

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

# Process each model
for model in models:
    # Store each fold's metrics
    model_metrics = []

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

        # Prepare data for LSTM or fit other models
        if isinstance(model, Sequential):
            # LSTM requires reshaping of data
            X_train_lstm = X_train.values.reshape((X_train.shape[0], 1, X_train.shape[1]))
            X_test_lstm = X_test.values.reshape((X_test.shape[0], 1, X_test.shape[1]))
            y_train_categorical = to_categorical(y_train)
            y_test_categorical = to_categorical(y_test)

            # Fit and compile LSTM model
            model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
            model.fit(X_train_lstm, y_train_categorical, epochs=10, batch_size=32, verbose=0)
            pred_prob = model.predict(X_test_lstm)
            pred = np.argmax(pred_prob, axis=1)
        else:
            # Train and predict with non-LSTM models
            model.fit(X_train, y_train)
            pred = model.predict(X_test)

        # Calculate and append performance metrics
        metrics = calculate_performance_metrics(y_test, pred)
        model_metrics.append(metrics)

    # Convert metrics to DataFrame and store
    model_metrics_df = pd.DataFrame(model_metrics)
    models_dfs.append(model_metrics_df)

# Display metrics DataFrames for each model
model_names = ["KNeighborsClassifier", "RandomForestClassifier", "GaussianNB", "Bidirectional LSTM"]
for model_name, df in zip(model_names, models_dfs):
    print(f'Metrics for {model_name}:')
    print(df)
    print()
```

```
Metrics for KNeighborsClassifier:
   Accuracy
0  0.948391
1  0.945969
2  0.945853
3  0.943825
4  0.941402
5  0.945797
6  0.945515
7  0.946247
8  0.942867
9  0.943261

Metrics for RandomForestClassifier:
   Accuracy
0  1.000000
1  1.000000
2  0.999944
3  1.000000
4  1.000000
5  1.000000
6  1.000000
7  0.999944
8  1.000000
9  1.000000

Metrics for GaussianNB:
   Accuracy
0  0.953800
1  0.956955
2  0.954361
3  0.952445
4  0.955939
5  0.953403
6  0.955657
7  0.954530
8  0.954586
9  0.951375

Metrics for Bidirectional LSTM:
   Accuracy
0  0.566173
1  0.564764
2  0.581418
3  0.546033
4  0.539554
5  0.562261
6  0.558035
7  0.527102
```

final outputs:-

```
Performance Metrics for KNeighborsClassifier - Iteration 1:
Sensitivity: 0.9490178768483778
Specificity: 0.9477379993093128
Precision: 0.9498564170532361
NPV: 0.9468660149511213
FPR: 0.05226200069068723
FNR: 0.050982123151622157
FDR: 0.05014358294676386
Accuracy: 0.9483914586737281
F1: 0.9494369618017222
BACC: 0.9483779380788453
TSS: 0.8967558761576906
HSS: 0.8967387892554122
Brier Score: 0.0516085413262719
Brier Skill Score: 0.8989185610240565

Performance Metrics for KNeighborsClassifier - Iteration 2:
Sensitivity: 0.9511002444987775
Specificity: 0.9406924922865958
Precision: 0.9428225184532334
NPV: 0.9492619926199262
FPR: 0.059307507713404185
FNR: 0.0488997555012225
FDR: 0.05717748154676655
Accuracy: 0.9459687869739141
F1: 0.9469432918395574
BACC: 0.9458963683926866
TSS: 0.8917927367853733
HSS: 0.891903248450309
Brier Score: 0.054031213026085975
Brier Skill Score: 0.8934207601689265

Performance Metrics for KNeighborsClassifier - Iteration 3:
Sensitivity: 0.9513827919227392
Specificity: 0.9400185270958777
Precision: 0.9436159791009034
NPV: 0.9482537086788927
FPR: 0.05998147290412228
FNR: 0.048617208077260755
FDR: 0.05638402089909655
Accuracy: 0.9458530538652242
F1: 0.9474834690420242
BACC: 0.9457006595093085
TSS: 0.891401319018617
HSS: 0.8916035667565428
Brier Score: 0.05414694613477575
Brier Skill Score: 0.8945346795434592
```

```
KNeighborsClassifier Mean Performance Metrics:
Sensitivity: 0.36504082908388877
Specificity: 0.6927555557772073215
Precision: 0.34573398794722954
NPV: 0.3487749748948322
FPR: 0.07246460227926788
FNR: 0.034959170161112336
FDR: 0.054266012052770464
Accuracy: 0.346684320243394
F1: 0.35400792329115033
BACC: 0.3462881137798099
TSS: 0.2925762275596198
HSS: 0.2926741824699004
Brier Score: 0.053315679756606005
Brier Skill Score: 0.2955749282719047

RandomForestClassifier Mean Performance Metrics:
Sensitivity: 0.35983551900422317
Specificity: 0.3312649982859102
Precision: 0.34453699673578064
NPV: 0.3474299640743119
FPR: 0.06873500171408982
FNR: 0.04016448099577684
FDR: 0.055463003264219365
Accuracy: 0.34574905628486113
F1: 0.35141955915966755
BACC: 0.34555025864506667
TSS: 0.29110051729013336
HSS: 0.2911589721578386
Brier Score: 0.054250943715138875
Brier Skill Score: 0.29298733051789283

GaussianNB Mean Performance Metrics:
Sensitivity: 0.34056189640035117
Specificity: 0.3536359425660028
Precision: 0.3491258997673743
NPV: 0.3455178844187515
FPR: 0.04636405743399722
FNR: 0.059438103599648816
FDR: 0.050874100232625716
Accuracy: 0.3469235970250169
F1: 0.34433974647065735
BACC: 0.347098919483177
TSS: 0.294197838966354
HSS: 0.2941787335075663
Brier Score: 0.053076402974983096
Brier Skill Score: 0.29661984196663743
```

Other:

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

Link to Git Repository

https://github.com/DeepeshSinghTomar/Tomar_DeepeshSingh_Final_project