FINAL PROJECT - Data Mining

Using RF, Naive Bayes and LSTM to Classify Mushrooms into edible and poisonous.

Goal: This project aims to evaluate the edibility of mushrooms (edible or poisonous) using machine learning classification algorithms, including Random Forest, Naive Bayes, and a deep learning model (LSTM). The dataset used for this analysis is the "Mushroom Cleaned" dataset, which contains information about mushroom features and their respective classifications.

Dataset Link: https://www.kaggle.com/datasets/prishasawhney/mushroom-dataset

Importing the packages and libraries that are required for the project

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")

from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping

from sklearn.metrics import confusion_matrix, accuracy_score, roc_auc_score, roc_curve, brier_score_loss, auc
from sklearn.model_selection import StratifiedKFold, train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.utils import resample
from sklearn.feature_selection import SelectKBest, f_classif

from imblearn.over_sampling import SMOTE
```

Data Loading and Exploration of Dataset

The dataset was loaded and examined for missing or inconsistent values. Non-numerical features were converted into numerical equivalents using one-hot encoding.

Key steps:

1. Removed unnecessary features like names to ensure better generalization.
2. Missing values were handled by imputing the median for numerical attributes.

```
df = pd.read_csv('mushroom_cleaned.csv')
```

```
df = df.dropna()
```

```
df
```

	cap-diameter	cap-shape	gill-attachment	gill-color	stem-height	stem-width	stem-color	season	class
0	1372	2	2	10	3.807467	1545	11	1.804273	1
1	1461	2	2	10	3.807467	1557	11	1.804273	1
2	1371	2	2	10	3.612496	1566	11	1.804273	1
3	1261	6	2	10	3.787572	1566	11	1.804273	1
4	1305	6	2	10	3.711971	1464	11	0.943195	1
...	...	...	...	...	...	...	...	...	...
54030	73	5	3	2	0.887740	569	12	0.943195	1
54031	82	2	3	2	1.186164	490	12	0.943195	1
54032	82	5	3	2	0.915593	584	12	0.888450	1
54033	79	2	3	2	1.034963	491	12	0.888450	1
54034	72	5	3	2	1.158311	492	12	0.888450	1

54035 rows × 9 columns

```
df.describe()
```

	cap-diameter	cap-shape	gill-attachment	gill-color	stem-height	stem-width	stem-color	season	class
count	54035.000000	54035.000000	54035.000000	54035.000000	54035.000000	54035.000000	54035.000000	54035.000000	54035.000000
mean	567.257204	4.000315	2.142056	7.329509	0.759110	1051.081299	8.418062	0.952163	0.549181
std	359.883763	2.160505	2.228821	3.200266	0.650969	782.056076	3.262078	0.305594	0.497580
min	0.000000	0.000000	0.000000	0.000000	0.000426	0.000000	0.000000	0.027372	0.000000
25%	289.000000	2.000000	0.000000	5.000000	0.270997	421.000000	6.000000	0.888450	0.000000
50%	525.000000	5.000000	1.000000	8.000000	0.593295	923.000000	11.000000	0.943195	1.000000
75%	781.000000	6.000000	4.000000	10.000000	1.054858	1523.000000	11.000000	0.943195	1.000000
max	1891.000000	6.000000	6.000000	11.000000	3.835320	3569.000000	12.000000	1.804273	1.000000

```
df.dtypes
```

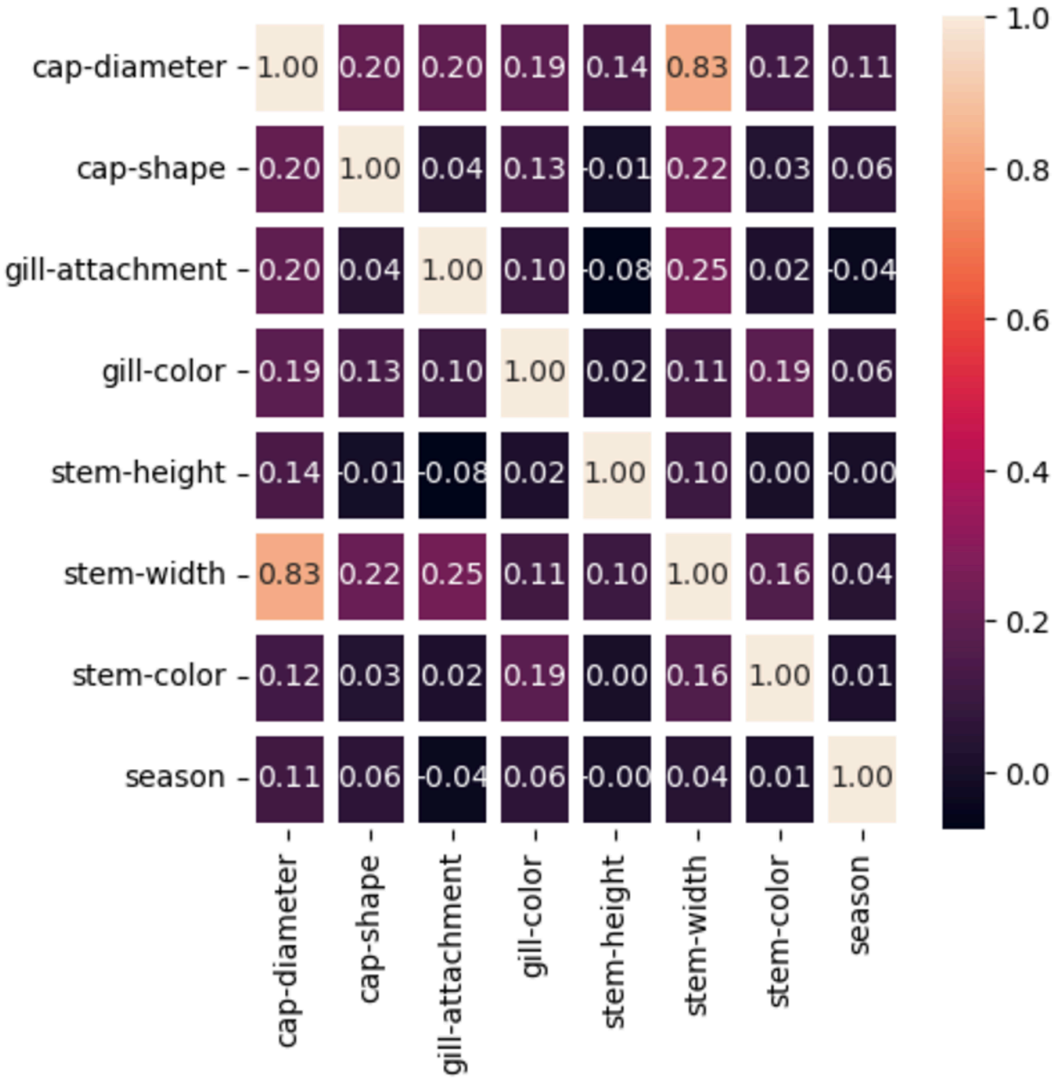
```
cap-diameter          int64
cap-shape             int64
gill-attachment       int64
gill-color            int64
stem-height         float64
stem-width            int64
stem-color            int64
season              float64
class                 int64
dtype: object
```

Data Visualization and Data Preprocessing

Feature Relationships:

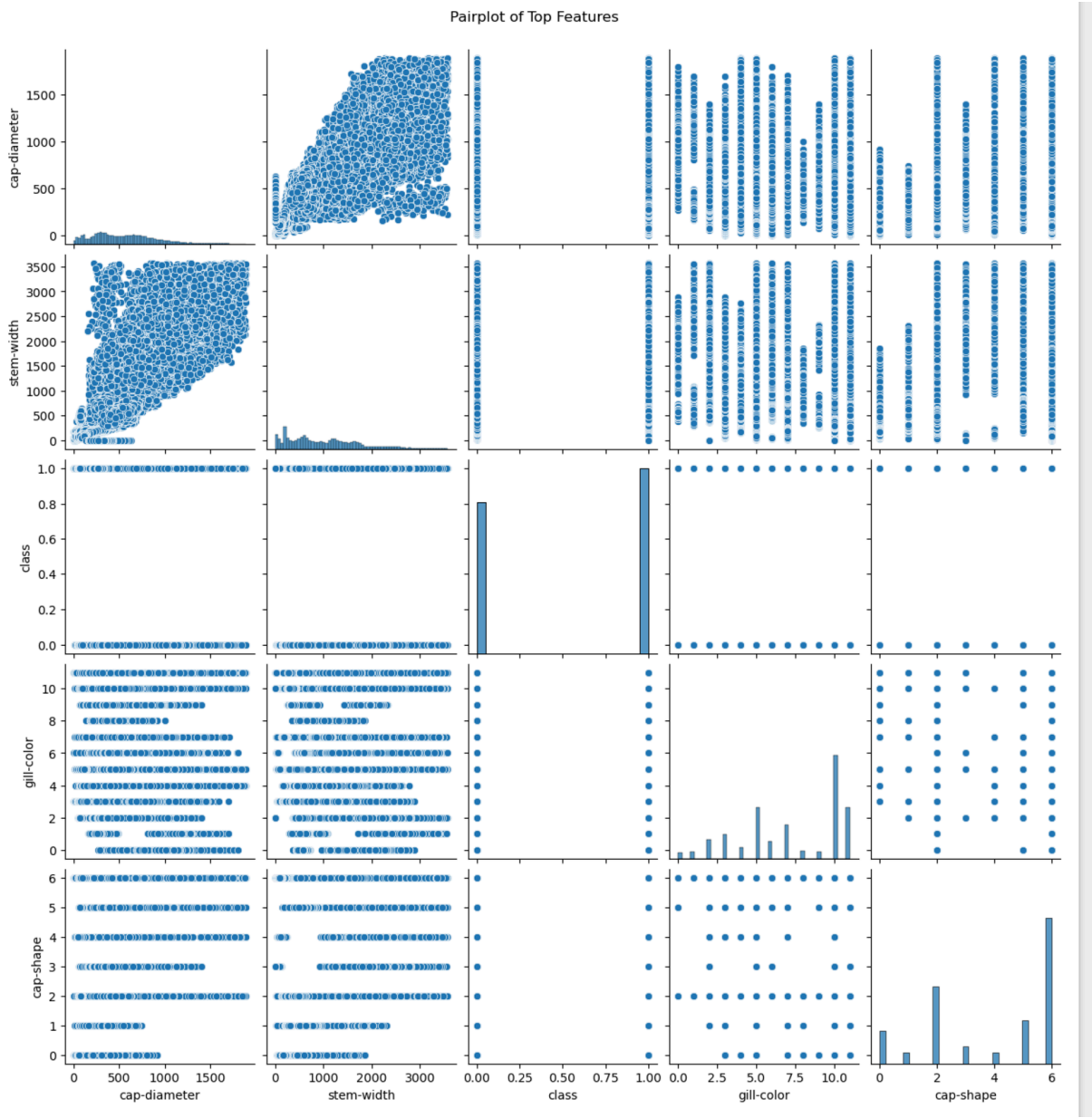
Heatmap: A correlation heatmap identifies collinearity among features.

```
# Determining Correlation between different attributes
fig, axis = plt.subplots(figsize=(5, 5))
correlation_matrix = X.corr()
sns.heatmap(correlation_matrix, annot=True, linewidths=5, fmt='.2f', ax=axis)
plt.show()
```



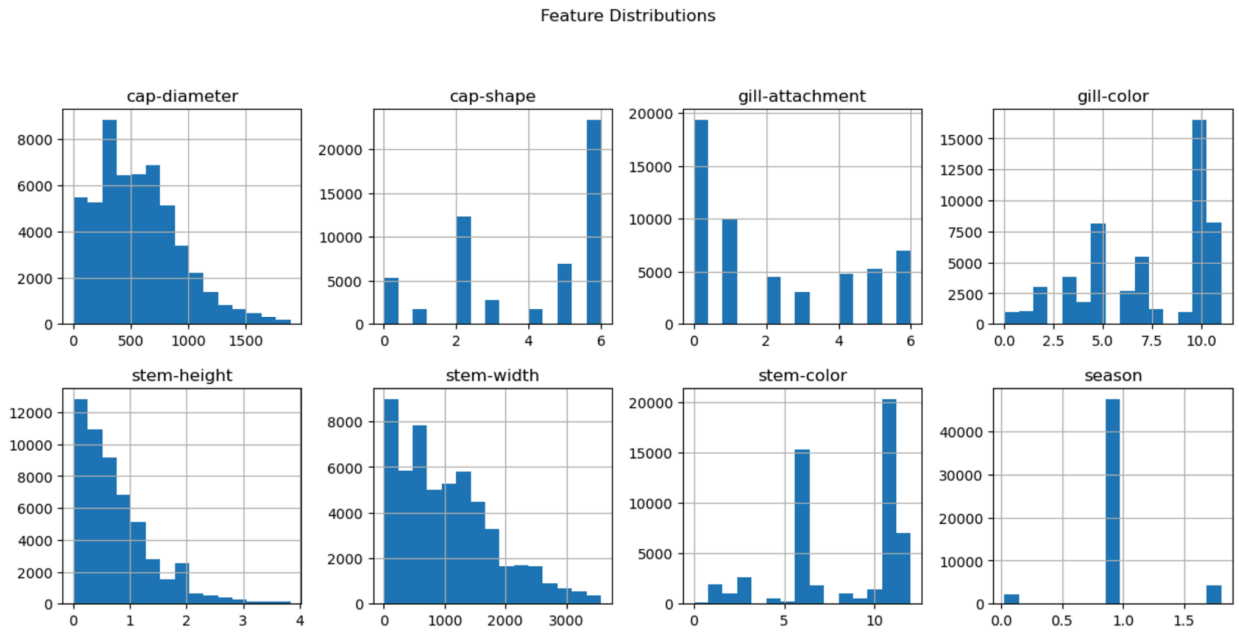
Pairplot: The pairplot revealed distinct clustering patterns for key features, showcasing their potential in distinguishing between classes effectively.

```
top_features = df.corr().abs().sum().sort_values(ascending=False).index[:5]
sns.pairplot(df[top_features], height=2.5)
plt.suptitle("Pairplot of Top Features", y=1.02)
plt.show()
```



Histograms: Features exhibited varied distributions, highlighting the diversity of attributes contributing to classification.

```
X.hist(bins=15, figsize=(15, 10), layout=(3, 4))
plt.suptitle("Feature Distributions")
plt.show()
```



This imbalanced dataset required the application of oversampling techniques to ensure fair and accurate evaluation of classification models.

```
# Checking for Data Imbalance
positive_op, negative_op = y.value_counts()
total_samples = y.count()

print("------ Label Imbalance Check ------")
print(f'Number of positive outcomes: {positive_op}, Percentage of positive outcomes: {(positive_op/total_samples)*100:.2f}')
print(f'Number of negative outcomes: {negative_op}, Percentage of negative outcomes: {(negative_op/total_samples)*100:.2f}')
```

```
------ Label Imbalance Check ------
Number of positive outcomes: 29675, Percentage of positive outcomes: 54.92
Number of negative outcomes: 24360, Percentage of negative outcomes: 45.08
```

```
majority = df[df['class'] == 0]
minority = df[df['class'] == 1]
minority_upsampled  = resample(minority, replace=True, n_samples=len(majority), random_state=42)
df_balanced = pd.concat([majority, minority_upsampled])

X = df_balanced.drop(columns=['class'])
y = df_balanced['class']
```

Feature Selection: Selected the most relevant features using statistical methods to reduce dimensionality, improve model efficiency, and enhance predictive performance by eliminating redundant or irrelevant data.

```
# Feature Selection
selector = SelectKBest(score_func=f_classif, k=5)
X_selected = selector.fit_transform(X, y)
```

Scaled features to standardize ranges, enhancing model performance for algorithms sensitive to input magnitude (e.g., SVM and LSTM).

```
# Feature Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_selected)
```

Reshaped the dataset to add an additional dimension, transforming the features into a format suitable for sequential models like LSTM. This step ensures the data has the required shape of `(samples, timesteps, features)`.

```
X_reshaped = np.expand_dims(X_scaled, axis=1)
```

3. Model Selection

We used three different algorithms to classify mushrooms:

Each model was tuned using hyperparameter optimization and evaluated using 10-fold cross-validation.

```
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
```

1. **Random Forest** (ensemble method).

```python
for train_index, test_index in kf.split(X_scaled, y):
    # Get initial train-test split
    X_train_full, X_test = X_scaled[train_index], X_scaled[test_index]
    y_train_full, y_test = y.iloc[train_index], y.iloc[test_index]

    # Further split training data into 60% train and 40% validation
    X_train, X_val, y_train, y_val = train_test_split(X_train_full, y_train_full, test_size=0.4, random_state=42, stratify=y_train_full)
    X_train_lstm, X_val_lstm = np.expand_dims(X_train, axis=1), np.expand_dims(X_val, axis=1)

    # Random Forest with enhanced regularization
    rf_model = RandomForestClassifier(n_estimators=50, max_depth=8, min_samples_split=15, min_samples_leaf=7, random_state=42, verbose=1)
    rf_model.fit(X_train, y_train)

    # Evaluate on validation set
    y_pred_val_rf = rf_model.predict(X_val)
    tp_val_rf, fp_val_rf, fn_val_rf, tn_val_rf = confusion_matrix(y_val, y_pred_val_rf).ravel()
    print("Random Forest Validation Metrics:", calculate_metrics(tp_val_rf, fp_val_rf, fn_val_rf, tn_val_rf))

    # Evaluate on test set
    y_pred_rf = rf_model.predict(X_test)
    tp_rf, fp_rf, fn_rf, tn_rf = confusion_matrix(y_test, y_pred_rf).ravel()
    results_rf.append(calculate_metrics(tp_rf, fp_rf, fn_rf, tn_rf))
```

```
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    1.4s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
Random Forest Validation Metrics: {'TPR': 0.8099074903014025, 'TNR': 0.9161212768799252, 'FPR': 0.0838787231200748, 'FNR': 0.1900925096859742, 'Recal
l': 0.8099074903014025, 'Precision': 0.9283922462941847, 'F1': 0.8651118312702545, 'Accuracy': 0.8552451539338655, 'Error_rate': 0.14475484606613453, 'B
ACC': 0.8630143835906638, 'TSS': 0.619814980602805, 'HSS': 0.7104903078677309, 'BS': 0.02095396545963031 4}
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    1.2s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
Random Forest Validation Metrics: {'TPR': 0.8010234205864987, 'TNR': 0.9146110056925996, 'FPR': 0.08538899430740038, 'FNR': 0.19897657941350128, 'Recal
l': 0.8010234205864987, 'Precision': 0.928164196123147, 'F1': 0.8599197126558208, 'Accuracy': 0.8488027366020524, 'Error_rate': 0.1511972633979476, 'BAC
C': 0.8578172131395492, 'TSS': 0.6020468411729973, 'HSS': 0.6976054732041049, 'BS': 0.022860612459028335}
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    1.4s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
Random Forest Validation Metrics: {'TPR': 0.807860696517413, 'TNR': 0.9130841121495327, 'FPR': 0.08691588785046729, 'FNR': 0.19213930348258706, 'Recal
l': 0.807860696517413, 'Precision': 0.925769669327252, 'F1': 0.8628055260361318, 'Accuracy': 0.852793614595211, 'Error_rate': 0.14720638540478903, 'BAC
C': 0.8604724043334728, 'TSS': 0.615721393034826, 'HSS': 0.7055872291904219, 'BS': 0.021669719903943292}
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    1.4s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
Random Forest Validation Metrics: {'TPR': 0.8151310563939634, 'TNR': 0.9250133904659882, 'FPR': 0.07498660953401179, 'FNR': 0.18486894360603653, 'Recal
l': 0.8151310563939634, 'Precision': 0.936145952109464, 'F1': 0.8714573824434774, 'Accuracy': 0.8619156214367161, 'Error_rate': 0.13808437856328393, 'BA
CC': 0.8700722234299758, 'TSS': 0.6302621127879269, 'HSS': 0.7238312428734321, 'BS': 0.019067295603208304}
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    1.6s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
Random Forest Validation Metrics: {'TPR': 0.8065026188358534, 'TNR': 0.917935588195661, 'FPR': 0.08206441180433904, 'FNR': 0.19349738116414666, 'Recal
l': 0.8065026188358534, 'Precision': 0.9305587229190422, 'F1': 0.8641007994070622, 'Accuracy': 0.853648802736602, 'Error_rate': 0.14635119726339796, 'BA
CC': 0.8622191035157571, 'TSS': 0.6130052376717068, 'HSS': 0.7072976054732041, 'BS': 0.021418672940430022}
```

```
Random Forest Validation Metrics: {'TPR': 0.8065026188358534, 'TNR': 0.917935588195661, 'FPR': 0.08206441180433904, 'FNR': 0.19349738116414666, 'Recall': 0.8065026188358534, 'Precision': 0.9305587229190422, 'F1': 0.8641007994070622, 'Accuracy': 0.853648802736602, 'Error_rate': 0.14635119726339796, 'BACC': 0.8622191035157571, 'TSS': 0.6130052376717068, 'HSS': 0.7072976054732041, 'BS': 0.021418672940430022}
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    1.4s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
Random Forest Validation Metrics: {'TPR': 0.807845084409136, 'TNR': 0.9149933065595717, 'FPR': 0.08500669344042838, 'FNR': 0.19215491559086395, 'Recall': 0.807845084409136, 'Precision': 0.927594070695553, 'F1': 0.8635881104033971, 'Accuracy': 0.8534777651083238, 'Error_rate': 0.14652223489167615, 'BACC': 0.8614191954843539, 'TSS': 0.6156901688182721, 'HSS': 0.7069555302166477, 'BS': 0.021468765317651527}
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    2.0s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
Random Forest Validation Metrics: {'TPR': 0.8132854578096947, 'TNR': 0.9180196965664094, 'FPR': 0.08198030343359063, 'FNR': 0.1867145421903052, 'Recall': 0.8132854578096947, 'Precision': 0.9297605473204105, 'F1': 0.8676314109384974, 'Accuracy': 0.8581527936145952, 'Error_rate': 0.14184720638540482, 'BACC': 0.8656525771880521, 'TSS': 0.6265709156193895, 'HSS': 0.7163055872291905, 'BS': 0.020120629959343623}
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    1.8s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
Random Forest Validation Metrics: {'TPR': 0.8017555971989349, 'TNR': 0.9133900824212944, 'FPR': 0.08660991757870558, 'FNR': 0.1982444028010652, 'Recall': 0.8017555971989349, 'Precision': 0.9269099201824401, 'F1': 0.8598022105875509, 'Accuracy': 0.8488597491448119, 'Error_rate': 0.15114025085518812, 'BACC': 0.8575728398101146, 'TSS': 0.6035111943978697, 'HSS': 0.6977194982896238, 'BS': 0.0228433754285692}
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    1.3s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
Random Forest Validation Metrics: {'TPR': 0.8085762610685504, 'TNR': 0.9141407397516357, 'FPR': 0.08585926024836427, 'FNR': 0.1914237389314496, 'Recall': 0.8085762610685504, 'Precision': 0.9266818700114025, 'F1': 0.8636097975665479, 'Accuracy': 0.853648802736602, 'Error_rate': 0.14635119726339796, 'BACC': 0.861358500410093, 'TSS': 0.6171525221371008, 'HSS': 0.7072976054732041, 'BS': 0.021418672940430022}
Random Forest Validation Metrics: {'TPR': 0.8110977564102564, 'TNR': 0.911064055055585, 'FPR': 0.08893594494441504, 'FNR': 0.18890224358974358, 'Recall': 0.8110977564102564, 'Precision': 0.9233751425313569, 'F1': 0.863602431481284, 'Accuracy': 0.8541619156214367, 'Error_rate': 0.14583808437856327, 'BACC': 0.8610809057329207, 'TSS': 0.6221955128205128, 'HSS': 0.7083238312428735, 'BS': 0.02126874685520894}
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    1.2s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
```

2. **Naive Bayes** (Gaussian Naive Bayes).

```python
for train_index, test_index in kf.split(X_scaled, y):
    # Split data into train and test for the current fold
    X_train_full, X_test = X_scaled[train_index], X_scaled[test_index]
    y_train_full, y_test = y.iloc[train_index], y.iloc[test_index]

    # Further split training data into 60% train and 40% validation
    X_train, X_val, y_train, y_val = train_test_split(
        X_train_full, y_train_full, test_size=0.4, random_state=None, stratify=y_train_full
    )

    # Naive Bayes Model
    nb_model = GaussianNB(var_smoothing=1e-8)
    nb_model.fit(X_train, y_train)

    # Evaluate on validation set
    y_pred_val_nb = nb_model.predict(X_val)
    #y_prob_val_nb = nb_model.predict_proba(X_val)[:, 1]  # For probabilistic metrics
    tp_val_nb, fp_val_nb, fn_val_nb, tn_val_nb = confusion_matrix(y_val, y_pred_val_nb).ravel()
    print("Naive Bayes Validation Metrics:", calculate_metrics(tp_val_nb, fp_val_nb, fn_val_nb, tn_val_nb))

    # Evaluate on test set
    y_pred_test_nb = nb_model.predict(X_test)
    #y_prob_test_nb = nb_model.predict_proba(X_test)[:, 1]
    tp_val_nb, fp_val_nb, fn_val_nb, tn_val_nb = confusion_matrix(y_val, y_pred_val_nb).ravel()
    print("Naive Bayes Validation Metrics:", calculate_metrics(tp_val_nb, fp_val_nb, fn_val_nb, tn_val_nb))
```

```
Naive Bayes Validation Metrics: {'TPR': 0.6306049069373942, 'TNR': 0.6527709054923305, 'FPR': 0.3472290945076695, 'FNR': 0.36939509306260576, 'Recall':
0.6306049069373942, 'Precision': 0.6799315849486887, 'F1': 0.6543399539119938, 'Accuracy': 0.6408209806157354, 'Error_rate': 0.35917901938426455, 'BAC
C': 0.6416879062148624, 'TSS': 0.2612098138747885, 'HSS': 0.28164196123147095, 'BS': 0.1290095679658419}
Naive Bayes Validation Metrics: {'TPR': 0.6306049069373942, 'TNR': 0.6527709054923305, 'FPR': 0.3472290945076695, 'FNR': 0.36939509306260576, 'Recall':
0.6306049069373942, 'Precision': 0.6799315849486887, 'F1': 0.6543399539119938, 'Accuracy': 0.6408209806157354, 'Error_rate': 0.35917901938426455, 'BAC
C': 0.6416879062148624, 'TSS': 0.2612098138747885, 'HSS': 0.28164196123147095, 'BS': 0.1290095679658419}
Naive Bayes Validation Metrics: {'TPR': 0.6314192849404117, 'TNR': 0.6459687123947052, 'FPR': 0.35403128760529484, 'FNR': 0.3685807150595883, 'Recall':
0.6314192849404117, 'Precision': 0.6645381984036488, 'F1': 0.6475555555555556, 'Accuracy': 0.6383124287343216, 'Error_rate': 0.3616875712656784, 'BACC':
0.6386939986675584, 'TSS': 0.2628385698808234, 'HSS': 0.2766248574686431, 'BS': 0.13081789920806525}
Naive Bayes Validation Metrics: {'TPR': 0.6314192849404117, 'TNR': 0.6459687123947052, 'FPR': 0.35403128760529484, 'FNR': 0.3685807150595883, 'Recall':
0.6314192849404117, 'Precision': 0.6645381984036488, 'F1': 0.6475555555555556, 'Accuracy': 0.6383124287343216, 'Error_rate': 0.3616875712656784, 'BACC':
0.6386939986675584, 'TSS': 0.2628385698808234, 'HSS': 0.2766248574686431, 'BS': 0.13081789920806525}
Naive Bayes Validation Metrics: {'TPR': 0.6346320346320347, 'TNR': 0.6498795180722892, 'FPR': 0.35012048192771084, 'FNR': 0.36536796536796534, 'Recall':
0.6346320346320347, 'Precision': 0.6686431014823261, 'F1': 0.6511937812326485, 'Accuracy': 0.6418472063854048, 'Error_rate': 0.3581527936145952, 'BACC':
0.6422557763521619, 'TSS': 0.2692640692640693, 'HSS': 0.2836944127708096, 'BS': 0.12827342357393884}
Naive Bayes Validation Metrics: {'TPR': 0.6346320346320347, 'TNR': 0.6498795180722892, 'FPR': 0.35012048192771084, 'FNR': 0.36536796536796534, 'Recall':
0.6346320346320347, 'Precision': 0.6686431014823261, 'F1': 0.6511937812326485, 'Accuracy': 0.6418472063854048, 'Error_rate': 0.3581527936145952, 'BACC':
0.6422557763521619, 'TSS': 0.2692640692640693, 'HSS': 0.2836944127708096, 'BS': 0.12827342357393884}
Naive Bayes Validation Metrics: {'TPR': 0.6378038194444444, 'TNR': 0.6525708793849111, 'FPR': 0.3474291206150889, 'FNR': 0.3621961805555556, 'Recall':
0.6378038194444444, 'Precision': 0.6702394526795895, 'F1': 0.6536194818191926, 'Accuracy': 0.644811858608894, 'Error_rate': 0.355188141391106, 'BACC':
0.6451873494146778, 'TSS': 0.27560763888888884, 'HSS': 0.2896237172177879, 'BS': 0.12615861578486834}
Naive Bayes Validation Metrics: {'TPR': 0.6378038194444444, 'TNR': 0.6525708793849111, 'FPR': 0.3474291206150889, 'FNR': 0.3621961805555556, 'Recall':
0.6378038194444444, 'Precision': 0.6702394526795895, 'F1': 0.6536194818191926, 'Accuracy': 0.644811858608894, 'Error_rate': 0.355188141391106, 'BACC':
0.6451873494146778, 'TSS': 0.27560763888888884, 'HSS': 0.2896237172177879, 'BS': 0.12615861578486834}
Naive Bayes Validation Metrics: {'TPR': 0.6332629355860613, 'TNR': 0.6563816604708798, 'FPR': 0.3436183395291202, 'FNR': 0.3667370644139388, 'Recall':
0.6332629355860613, 'Precision': 0.6838084378563284, 'F1': 0.6575657894736842, 'Accuracy': 0.6438996579247435, 'Error_rate': 0.35610034207525654, 'BAC
C': 0.6448222980284706, 'TSS': 0.2665258711721225, 'HSS': 0.28779931584948687, 'BS': 0.12680745362611473}
Naive Bayes Validation Metrics: {'TPR': 0.6332629355860613, 'TNR': 0.6563816604708798, 'FPR': 0.3436183395291202, 'FNR': 0.3667370644139388, 'Recall':
0.6332629355860613, 'Precision': 0.6838084378563284, 'F1': 0.6575657894736842, 'Accuracy': 0.6438996579247435, 'Error_rate': 0.35610034207525654, 'BAC
C': 0.6448222980284706, 'TSS': 0.2665258711721225, 'HSS': 0.28779931584948687, 'BS': 0.12680745362611473}
Naive Bayes Validation Metrics: {'TPR': 0.63187513272457, 'TNR': 0.6529180004924895, 'FPR': 0.34708199950751045, 'FNR': 0.36812486727543003, 'Recall':
0.63187513272457, 'Precision': 0.6785632839224629, 'F1': 0.6543875082471959, 'Accuracy': 0.6416191562143672, 'Error_rate': 0.35838084378563284, 'BACC':
0.6423965666085297, 'TSS': 0.26375026544913993, 'HSS': 0.2832383124287343, 'BS': 0.12843682919250218}
```

```
Naive Bayes Validation Metrics: {'TPR': 0.6347169811320754, 'TNR': 0.6511796733212342, 'FPR': 0.3488203266787659, 'FNR': 0.3652830188679245, 'Recall':
0.6347169811320754, 'Precision': 0.6712656784492589, 'F1': 0.6524799113327793, 'Accuracy': 0.6424743443557582, 'Error_rate': 0.35752565564424177, 'BAC
C': 0.6429483272266547, 'TSS': 0.2694339622641509, 'HSS': 0.2849486887115165, 'BS': 0.12782459444384492}
Naive Bayes Validation Metrics: {'TPR': 0.6347169811320754, 'TNR': 0.6511796733212342, 'FPR': 0.3488203266787659, 'FNR': 0.3652830188679245, 'Recall':
0.6347169811320754, 'Precision': 0.6712656784492589, 'F1': 0.6524799113327793, 'Accuracy': 0.6424743443557582, 'Error_rate': 0.35752565564424177, 'BAC
C': 0.6429483272266547, 'TSS': 0.2694339622641509, 'HSS': 0.2849486887115165, 'BS': 0.12782459444384492}
Naive Bayes Validation Metrics: {'TPR': 0.633379664278841, 'TNR': 0.6523757176010748, 'FPR': 0.3476242823989251, 'FNR': 0.366620335721159, 'Recall': 0.6
33379664278841, 'Precision': 0.675484606613455, 'F1': 0.6537548970920929, 'Accuracy': 0.6422462941847207, 'Error_rate': 0.3577537058152793, 'BACC': 0.64
28776909399578, 'TSS': 0.266759328557682, 'HSS': 0.2844925883694413, 'BS': 0.12798771402456546}
Naive Bayes Validation Metrics: {'TPR': 0.633379664278841, 'TNR': 0.6523757176010748, 'FPR': 0.3476242823989251, 'FNR': 0.366620335721159, 'Recall': 0.6
33379664278841, 'Precision': 0.675484606613455, 'F1': 0.6537548970920929, 'Accuracy': 0.6422462941847207, 'Error_rate': 0.3577537058152793, 'BACC': 0.64
28776909399578, 'TSS': 0.266759328557682, 'HSS': 0.2844925883694413, 'BS': 0.12798771402456546}
Naive Bayes Validation Metrics: {'TPR': 0.6336996336996337, 'TNR': 0.6502785178009203, 'FPR': 0.34972148219907967, 'FNR': 0.3663003663003663, 'Recall':
0.6336996336996337, 'Precision': 0.6706955530216647, 'F1': 0.6516729448260581, 'Accuracy': 0.6415051311288483, 'Error_rate': 0.3584948688711517, 'BACC':
0.641989075750277, 'TSS': 0.2673992673992674, 'HSS': 0.2830102622576967, 'BS': 0.12851857100694422}
Naive Bayes Validation Metrics: {'TPR': 0.6336996336996337, 'TNR': 0.6502785178009203, 'FPR': 0.34972148219907967, 'FNR': 0.3663003663003663, 'Recall':
0.6336996336996337, 'Precision': 0.6706955530216647, 'F1': 0.6516729448260581, 'Accuracy': 0.6415051311288483, 'Error_rate': 0.3584948688711517, 'BACC':
0.641989075750277, 'TSS': 0.2673992673992674, 'HSS': 0.2830102622576967, 'BS': 0.12851857100694422}
Naive Bayes Validation Metrics: {'TPR': 0.6357004934563398, 'TNR': 0.6539303966901923, 'FPR': 0.34606960330980774, 'FNR': 0.36429950654366017, 'Recall':
0.6357004934563398, 'Precision': 0.6757126567844925, 'F1': 0.6550961751050187, 'Accuracy': 0.6442417331812998, 'Error_rate': 0.35575826681870015, 'BAC
C': 0.644815445073266, 'TSS': 0.27140098691267966, 'HSS': 0.28848346636259975, 'BS': 0.12656394440984542}
Naive Bayes Validation Metrics: {'TPR': 0.6357004934563398, 'TNR': 0.6539303966901923, 'FPR': 0.34606960330980774, 'FNR': 0.36429950654366017, 'Recall':
0.6357004934563398, 'Precision': 0.6757126567844925, 'F1': 0.6550961751050187, 'Accuracy': 0.6442417331812998, 'Error_rate': 0.35575826681870015, 'BAC
C': 0.644815445073266, 'TSS': 0.27140098691267966, 'HSS': 0.28848346636259975, 'BS': 0.12656394440984542}
```

3. **Long Short-Term Memory (LSTM)** for deep learning.

```python
for train_index, test_index in kf.split(X_scaled, y):
    X_train_full, X_test = X_scaled[train_index], X_scaled[test_index]
    y_train_full, y_test = y.iloc[train_index], y.iloc[test_index]

    # Further split training data into 60% train and 40% validation
    X_train, X_val, y_train, y_val = train_test_split(X_train_full, y_train_full, test_size=0.4, random_state=42, stratify=y_train_full)
    X_train_lstm, X_val_lstm = np.expand_dims(X_train, axis=1), np.expand_dims(X_val, axis=1)


    # LSTM with early stopping and enhanced regularization
    lstm_model = Sequential()
    lstm_model.add(LSTM(32, input_shape=(X_train_lstm.shape[1], X_train_lstm.shape[2]), activation='relu', kernel_regularizer='l2'))
    lstm_model.add(Dropout(0.4))
    lstm_model.add(Dense(1, activation='sigmoid', kernel_regularizer='l2'))
    lstm_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
    lstm_model.fit(X_train_lstm, y_train, validation_data=(X_val_lstm, y_val), epochs=15, batch_size=64, callbacks=[early_stopping], verbose=1)

    # Evaluate on test set
    y_pred_lstm = (lstm_model.predict(np.expand_dims(X_test, axis=1)) > 0.5).astype("int32")
    tp_lstm, fp_lstm, fn_lstm, tn_lstm = confusion_matrix(y_test, y_pred_lstm).ravel()
    results_lstm.append(calculate_metrics(tp_lstm, fp_lstm, fn_lstm, tn_lstm))
```

```
Epoch 1/15
412/412 ━━━━━━━━━━━━━━━━━━━━ 8s 8ms/step - accuracy: 0.6189 - loss: 0.7338 - val_accuracy: 0.6400 - val_loss: 0.6751
Epoch 2/15
412/412 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.6383 - loss: 0.6718 - val_accuracy: 0.6424 - val_loss: 0.6682
Epoch 3/15
412/412 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.6351 - loss: 0.6684 - val_accuracy: 0.6399 - val_loss: 0.6653
Epoch 4/15
412/412 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.6339 - loss: 0.6664 - val_accuracy: 0.6433 - val_loss: 0.6635
Epoch 5/15
412/412 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.6342 - loss: 0.6655 - val_accuracy: 0.6423 - val_loss: 0.6617
Epoch 6/15
412/412 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.6379 - loss: 0.6620 - val_accuracy: 0.6466 - val_loss: 0.6605
Epoch 7/15
412/412 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.6382 - loss: 0.6589 - val_accuracy: 0.6432 - val_loss: 0.6596
Epoch 8/15
412/412 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.6412 - loss: 0.6600 - val_accuracy: 0.6462 - val_loss: 0.6587
Epoch 9/15
412/412 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.6443 - loss: 0.6574 - val_accuracy: 0.6421 - val_loss: 0.6583
```

```
412/412 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - accuracy: 0.6362 - loss: 0.6614 - val_accuracy: 0.6515 - val_loss: 0.6590
Epoch 8/15
412/412 ━━━━━━━━━━━━━━━━━━━━ 5s 13ms/step - accuracy: 0.6392 - loss: 0.6611 - val_accuracy: 0.6462 - val_loss: 0.6580
Epoch 9/15
412/412 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.6364 - loss: 0.6606 - val_accuracy: 0.6433 - val_loss: 0.6573
Epoch 10/15
412/412 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.6396 - loss: 0.6580 - val_accuracy: 0.6471 - val_loss: 0.6568
Epoch 11/15
412/412 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.6372 - loss: 0.6600 - val_accuracy: 0.6434 - val_loss: 0.6562
Epoch 12/15
412/412 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.6355 - loss: 0.6576 - val_accuracy: 0.6440 - val_loss: 0.6556
Epoch 13/15
412/412 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.6396 - loss: 0.6560 - val_accuracy: 0.6495 - val_loss: 0.6552
Epoch 14/15
412/412 ━━━━━━━━━━━━━━━━━━━━ 6s 8ms/step - accuracy: 0.6382 - loss: 0.6568 - val_accuracy: 0.6507 - val_loss: 0.6550
Epoch 15/15
412/412 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.6325 - loss: 0.6597 - val_accuracy: 0.6495 - val_loss: 0.6546
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step
```

4. Evaluation Metrics

The following metrics were calculated for each model across all folds and averaged**:**

- **True Positives (TP)**, **True Negatives (TN)**, **False Positives (FP)**, **False Negatives (FN)**

Derived metrics**:**

- **Accuracy**: Correct predictions over total predictions.
- **Precision**: Correct positive predictions over total predicted positives.
- **Recall (Sensitivity)**: Correct positive predictions over total actual positives.
- **F1-Score**: Harmonic mean of precision and recall.
- **Balanced Accuracy (BACC)**: Average of sensitivity (Recall) and specificity, providing a balanced evaluation for imbalanced datasets.
- **True Skill Statistic (TSS)**: Balance of sensitivity and specificity.
- **Heidke Skill Score (HSS)**: Performance against random predictions.
- **Brier Score (BS)**: Mean squared error of probabilistic predictions.

```python
results_rf, results_nb, results_lstm = [], [], []

def calculate_metrics(tp, fp, fn, tn):
    TPR = tp / (tp + fn) if (tp + fn) > 0 else 0
    TNR = tn / (tn + fp) if (tn + fp) > 0 else 0
    FPR = fp / (fp + tn) if (fp + tn) > 0 else 0
    FNR = fn / (fn + tp) if (fn + tp) > 0 else 0

    Precision = tp / (tp + fp) if (tp + fp) > 0 else 0
    Recall = TPR
    F1 = 2 * (Precision * Recall) / (Precision + Recall) if (Precision + Recall) > 0 else 0
    Accuracy = (tp + tn) / (tp + tn + fp + fn)
    Error_rate = 1 - Accuracy

    BACC = (TPR + TNR) / 2
    TSS = TPR - FNR
    HSS = 2 * (tp * tn - fp * fn) / ((tp + fn) * (fn + tn) + (tp + fp) * (fp + tn)) if ((tp + fn) * (fn + tn) + (tp + fp) * (fp + tn)) > 0 else 0

    BS = ((fp + fn) / (tp + tn + fp + fn)) ** 2

    return {
        "TPR": TPR, "TNR": TNR, "FPR": FPR, "FNR": FNR,
        "Recall": Recall, "Precision": Precision, "F1": F1,
        "Accuracy": Accuracy, "Error_rate": Error_rate, "BACC": BACC,
        "TSS": TSS, "HSS": HSS, "BS": BS
    }
```

Results

- **Random Forest Results Per Fold Metric and Average Metrics**

Random Forest Results (Per Fold):

	TPR	TNR	FPR	FNR	Recall	Precision	F1	\
0	0.814478	0.907207	0.092793	0.185522	0.814478	0.919130	0.863645	
1	0.806762	0.918041	0.081959	0.193238	0.806762	0.930624	0.864278	
2	0.813108	0.915036	0.084964	0.186892	0.813108	0.926929	0.866296	
3	0.805288	0.912205	0.087795	0.194712	0.805288	0.925287	0.861127	
4	0.804263	0.916383	0.083617	0.195737	0.804263	0.929392	0.862312	
5	0.814245	0.922522	0.077478	0.185755	0.814245	0.933908	0.869981	
6	0.804105	0.903103	0.096897	0.195895	0.804105	0.916667	0.856704	
7	0.799504	0.912640	0.087360	0.200496	0.799504	0.926519	0.858338	
8	0.813419	0.918945	0.081055	0.186581	0.813419	0.930624	0.868083	
9	0.802557	0.914397	0.085603	0.197443	0.802557	0.927750	0.860625	

	Accuracy	Error_rate	BACC	TSS	HSS	BS
0	0.854885	0.145115	0.860842	0.628956	0.709770	0.021058
1	0.853859	0.146141	0.862401	0.613523	0.707718	0.021357
2	0.856938	0.143062	0.864072	0.626215	0.713875	0.020467
3	0.850780	0.149220	0.858746	0.610575	0.701560	0.022267
4	0.851601	0.148399	0.860323	0.608526	0.703202	0.022022
5	0.860427	0.139573	0.868383	0.628490	0.720854	0.019481
6	0.846675	0.153325	0.853604	0.608210	0.693350	0.023509
7	0.847085	0.152915	0.856072	0.599008	0.694171	0.023383
8	0.858580	0.141420	0.866182	0.626839	0.717159	0.020000
9	0.849754	0.150246	0.858477	0.605114	0.699507	0.022574

```
Random Forest Average Metrics:
 TPR            0.807773
TNR             0.914048
FPR             0.085952
FNR             0.192227
Recall          0.807773
Precision       0.926683
F1              0.863139
Accuracy        0.853058
Error_rate      0.146942
BACC            0.860910
TSS             0.615546
HSS             0.706117
BS              0.021612
```

- **Naive Bayes Results Per Fold Metric and Average Metrics**

```
Naive Bayes Results (Per Fold):
        TPR       TNR       FPR       FNR    Recall  Precision        F1  \
0  0.634942  0.658244  0.341756  0.365058  0.634942   0.685405  0.659209
1  0.632873  0.656246  0.343754  0.367127  0.632873   0.684036  0.657461
2  0.635800  0.655497  0.344503  0.364200  0.635800   0.678791  0.656593
3  0.631148  0.651240  0.348760  0.368852  0.631148   0.676055  0.652830
4  0.637585  0.658011  0.341989  0.362415  0.637585   0.681642  0.658878
5  0.630573  0.647673  0.352327  0.369427  0.630573   0.669327  0.649372
6  0.634139  0.656258  0.343742  0.365861  0.634139   0.682440  0.657403
7  0.633120  0.652323  0.347677  0.366880  0.633120   0.675713  0.653723
8  0.634899  0.654712  0.345288  0.365101  0.634899   0.678335  0.655899
9  0.633362  0.655709  0.344291  0.366638  0.633362   0.682326  0.656933


   Accuracy  Error_rate      BACC       TSS       HSS        BS
0  0.645667    0.354333  0.646593  0.269885  0.291334  0.125552
1  0.643615    0.356385  0.644559  0.265745  0.287229  0.127011
2  0.644983    0.355017  0.645649  0.271601  0.289966  0.126037
3  0.640479    0.359521  0.641194  0.262295  0.280958  0.129255
4  0.647092    0.352908  0.647798  0.275171  0.294185  0.124544
5  0.638597    0.361403  0.639123  0.261145  0.277195  0.130612
6  0.644356    0.355644  0.645198  0.268277  0.288712  0.126483
7  0.642075    0.357925  0.642721  0.266239  0.284151  0.128110
8  0.644128    0.355872  0.644805  0.269797  0.288255  0.126645
9  0.643672    0.356328  0.644535  0.266723  0.287343  0.126970
```

```
Naive Bayes Average Metrics:
 TPR            0.633844
TNR             0.654591
FPR             0.345409
FNR             0.366156
Recall          0.633844
Precision       0.679407
F1              0.655830
Accuracy        0.643466
Error_rate      0.356534
BACC            0.644218
TSS             0.267688
HSS             0.286933
BS              0.127122
dtype: float64
```

- **LSTM Results Per Fold Metric and Average Metrics**

LSTM Results (Per Fold):

	TPR	TNR	FPR	FNR	Recall	Precision	F1	\
0	0.639756	0.663402	0.336598	0.360244	0.639756	0.689655	0.663769	
1	0.643709	0.664320	0.335680	0.356291	0.643709	0.686782	0.664548	
2	0.635213	0.655629	0.344371	0.364787	0.635213	0.679803	0.656752	
3	0.630810	0.658348	0.341652	0.369190	0.630810	0.690887	0.659483	
4	0.634995	0.680749	0.319251	0.365005	0.634995	0.727011	0.677895	
5	0.631187	0.661327	0.338673	0.368813	0.631187	0.696223	0.662112	
6	0.626843	0.650651	0.349349	0.373157	0.626843	0.680624	0.652627	
7	0.625733	0.659981	0.340019	0.374267	0.625733	0.700739	0.661115	
8	0.642163	0.658844	0.341156	0.357837	0.642163	0.677750	0.659477	
9	0.624063	0.667793	0.332207	0.375937	0.624063	0.717570	0.667558	

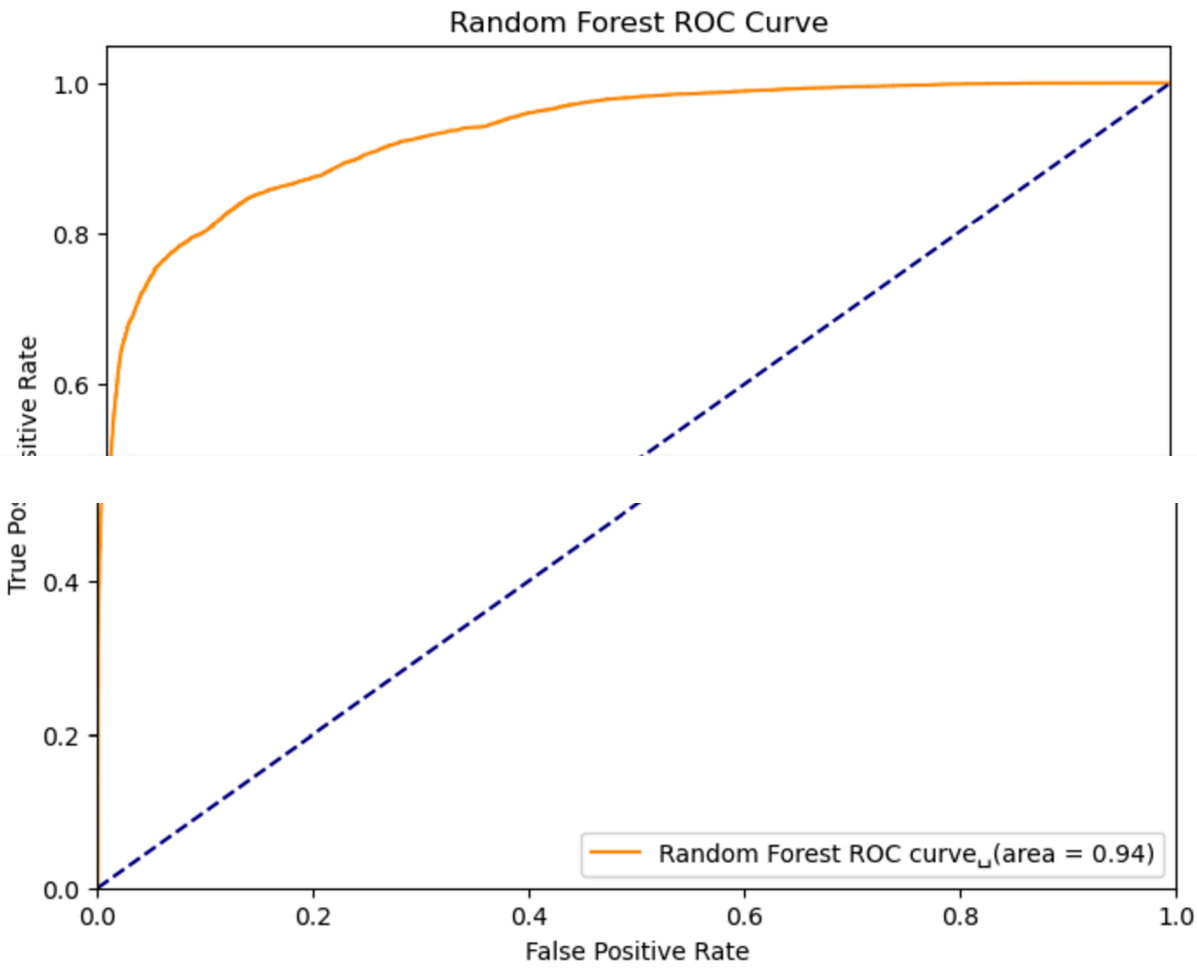
	Accuracy	Error_rate	BACC	TSS	HSS	BS
0	0.650657	0.349343	0.651579	0.279513	0.301314	0.122041
1	0.653325	0.346675	0.654015	0.287418	0.306650	0.120183
2	0.644704	0.355296	0.645421	0.270426	0.289409	0.126235
3	0.643268	0.356732	0.644579	0.261619	0.286535	0.127258
4	0.654557	0.345443	0.657872	0.269989	0.309113	0.119331
5	0.644704	0.355296	0.646257	0.262374	0.289409	0.126235
6	0.637726	0.362274	0.638747	0.253686	0.275452	0.131243
7	0.640805	0.359195	0.642857	0.251466	0.281609	0.129021
8	0.650041	0.349959	0.650503	0.284325	0.300082	0.122471
9	0.642652	0.357348	0.645928	0.248126	0.285304	0.127698

```
LSTM Average Metrics:
 TPR            0.633447
TNR             0.662105
FPR             0.337895
FNR             0.366553
Recall          0.633447
Precision       0.694704
F1              0.662534
Accuracy        0.646244
Error_rate      0.353756
BACC            0.647776
TSS             0.266894
HSS             0.292488
BS              0.125172
dtype: float64
```

Generating ROC Curves For all the Machine Learning Models

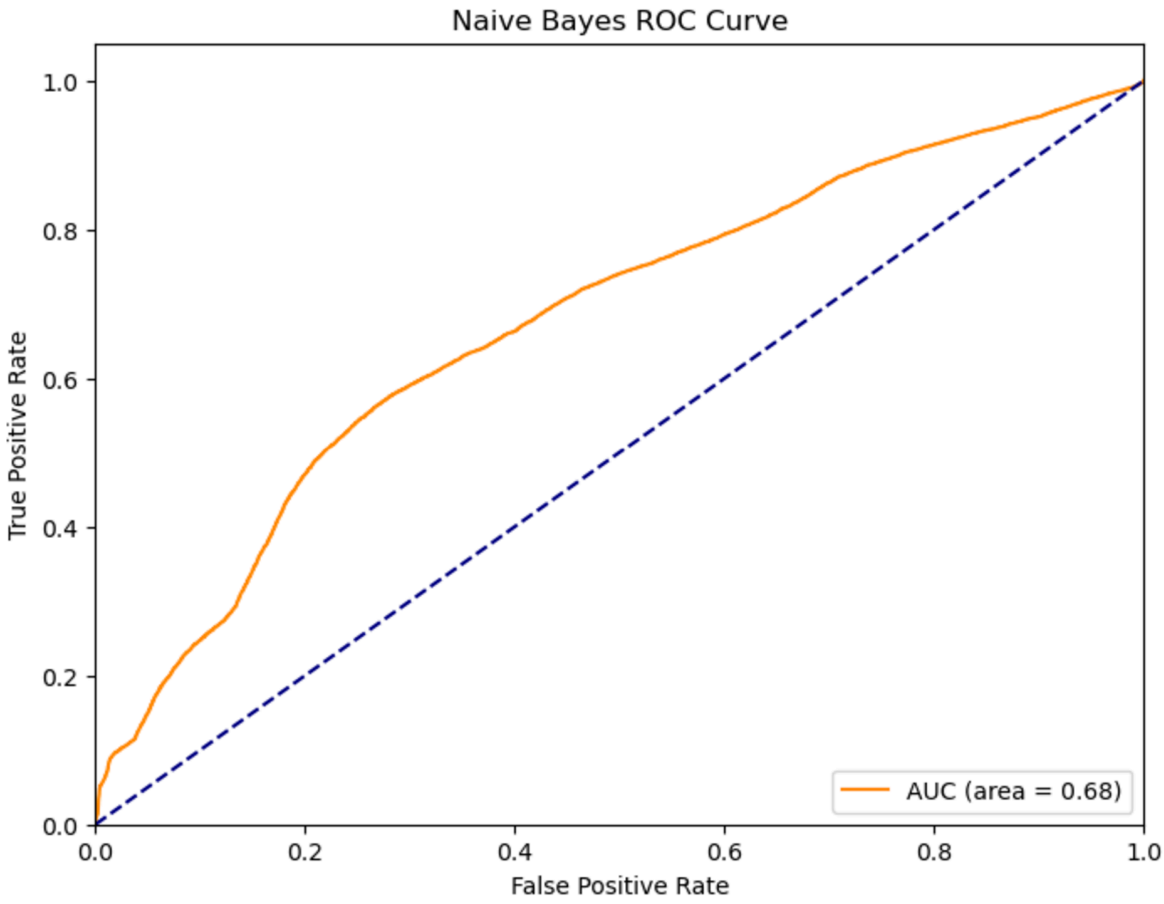
- **Random Forest ROC Curve**

```
# Obtain predicted probabilities
y_score_nb = nb_model.predict_proba(X_scaled)[:, 1]
# Compute ROC curve and ROC area
fpr_nb, tpr_nb, _ = roc_curve(y, y_score_nb)
roc_auc_nb = auc(fpr_nb, tpr_nb)

# Plot SVM ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_nb, tpr_nb, color="darkorange", label="AUC (area = {:.2f})".format(roc_auc_nb))
plt.plot([0, 1], [0, 1], color="navy", linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Naive Bayes ROC Curve")
plt.legend(loc="lower right")
plt.show()
```



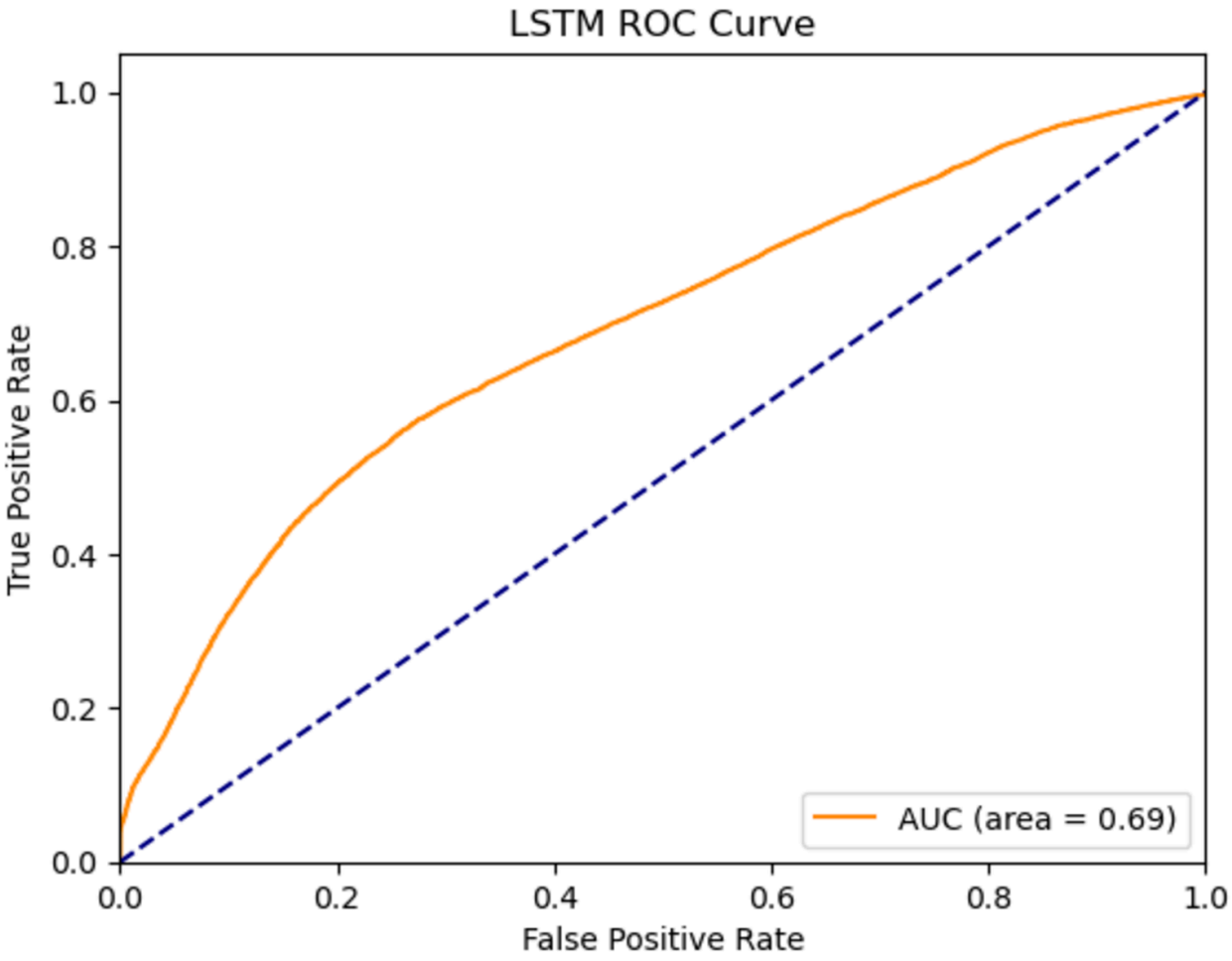
- **Naive Bayes ROC Curve**

```
# Obtain predicted probabilities
y_score_nb = nb_model.predict_proba(X_scaled)[:, 1]
# Compute ROC curve and ROC area
fpr_nb, tpr_nb, _ = roc_curve(y, y_score_nb)
roc_auc_nb = auc(fpr_nb, tpr_nb)

# Plot SVM ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_nb, tpr_nb, color="darkorange", label="AUC (area = {:.2f})".format(roc_auc_nb))
plt.plot([0, 1], [0, 1], color="navy", linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Naive Bayes ROC Curve")
plt.legend(loc="lower right")
plt.show()
```



- **LSTM ROC Curve**

```
# Predict probabilities for the test set
predict_lstm = lstm_model.predict(X_reshaped)
# Compute ROC curve and ROC area
fpr_lstm, tpr_lstm, _ = roc_curve(y, predict_lstm)
roc_auc_lstm = auc(fpr_lstm, tpr_lstm)
# Plot LSTM ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_lstm, tpr_lstm, color="darkorange", label="AUC (area = {:.2f})".format(roc_auc_lstm))
plt.plot([0, 1], [0, 1], color="navy", linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("LSTM ROC Curve")
plt.legend(loc="lower right")
plt.show()
```



Conclusion

- **Random Forest**:
  - Achieved the highest accuracy (85%) and AUC (0.94).
  - Robust to noise and effectively handled feature interactions.
- **Naive Bayes**:
  - Performed worse than Random Forest with 64.34% accuracy and AUC (0.68).
  - Simple and computationally efficient but limited by the assumption of feature independence.
- **LSTM**:
  - Demonstrated competitive performance despite its deep learning nature with 64.67% accuracy and AUC (0.69).
  - Required more tuning due to smaller dataset size.

Conclusion: Random Forest was the best performer, leveraging ensemble methods to capture complex feature relationships. Future work could explore deep learning on larger datasets for improved LSTM performance.