CS 634 Data Mining Final Term Project -- Tap47

github link: https://github.com/tap4725/Final_term_project

1. Introduction

This notebook provides the report of my work on utilizing 3 machine learning models on loan data for binary classification problem. Machine learning models i chose: 1) Random Forest, 2) KNN, 3) LSTM.

2. Abstract

The notebook includes code for evaluating model performance using various metrics. Key observations:

- Custom Metrics Calculation: The notebook defines a function (matrics_cal) to calculate performance metrics such as accuracy, precision, recall, F1 score, Brier score, AUC, and Brier Skill Score (BSS).
- Model Training and Evaluation: 3 models are defined and evaluated using 10-fold cross-validation (KFold). For each fold, metrics are calculated and stored in a list, which is later converted to a DataFrame.
- Performance Metrics: Metrics like AUC, precision, and Brier score suggest an evaluation process aimed at assessing probabilistic and classification performance.

3. Data Visualization and Preprocessing

In [3]:
```
import pandas as pd

df = pd.read_csv("data/loan_data.csv")
df.describe()
```

Out[3]:

	person_age	person_income	person_emp_exp	loan_amnt	loan_int_rate	loan_pe
count	45000.000000	4.500000e+04	45000.000000	45000.000000	45000.000000	
mean	27.764178	8.031905e+04	5.410333	9583.157556	11.006606	
std	6.045108	8.042250e+04	6.063532	6314.886691	2.978808	
min	20.000000	8.000000e+03	0.000000	500.000000	5.420000	
25%	24.000000	4.720400e+04	1.000000	5000.000000	8.590000	
50%	26.000000	6.704800e+04	4.000000	8000.000000	11.010000	
75%	30.000000	9.578925e+04	8.000000	12237.250000	12.990000	
max	144.000000	7.200766e+06	125.000000	35000.000000	20.000000	

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45000 entries, 0 to 44999
Data columns (total 14 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   person_age                      45000 non-null  float64
 1   person_gender                   45000 non-null  object
 2   person_education                45000 non-null  object
 3   person_income                   45000 non-null  float64
 4   person_emp_exp                  45000 non-null  int64
 5   person_home_ownership           45000 non-null  object
 6   loan_amnt                       45000 non-null  float64
 7   loan_intent                     45000 non-null  object
 8   loan_int_rate                   45000 non-null  float64
 9   loan_percent_income             45000 non-null  float64
 10  cb_person_cred_hist_length      45000 non-null  float64
 11  credit_score                    45000 non-null  int64
 12  previous_loan_defaults_on_file  45000 non-null  object
 13  loan_status                     45000 non-null  int64
dtypes: float64(6), int64(3), object(5)
memory usage: 4.8+ MB
```

As we can see there are multiple columns with string values are present. which we need to encode into their numerical representation.

In [5]:
```
df["loan_status"].unique()
```

Out[5]:
```
array([1, 0])
```

as we can see it is binary classification problem.

In [6]:
```
df.isnull().any()
```

Out[6]:
```
person_age                        False
person_gender                     False
person_education                  False
person_income                     False
person_emp_exp                    False
person_home_ownership             False
loan_amnt                         False
loan_intent                       False
loan_int_rate                     False
loan_percent_income               False
cb_person_cred_hist_length        False
credit_score                      False
previous_loan_defaults_on_file    False
loan_status                       False
dtype: bool
```

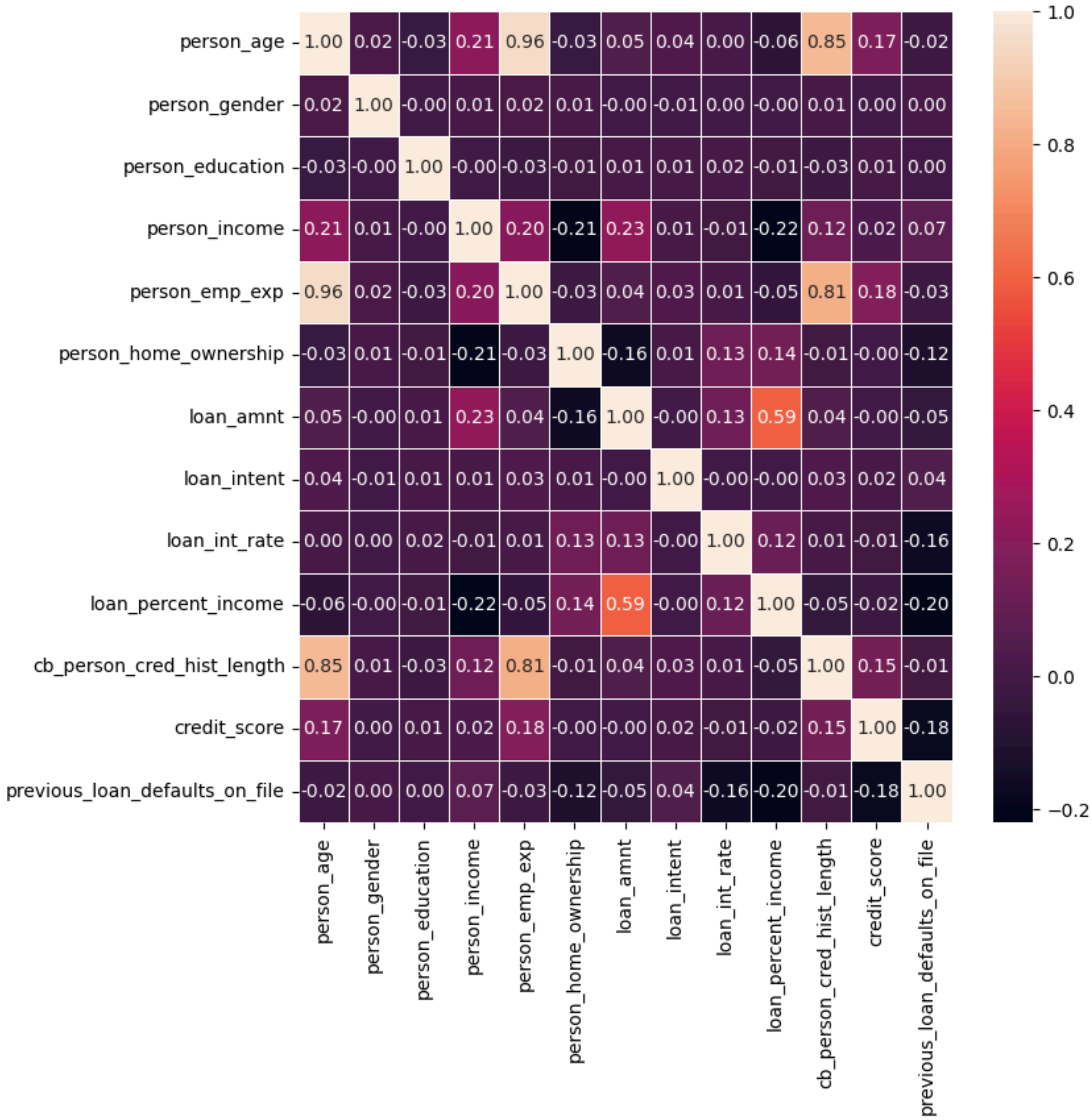
there are no null or missing values in the dataset.

In [7]:
```
df = df.groupby('loan_status').sample(frac=0.2, random_state=42)
df["loan_status"].value_counts()
```

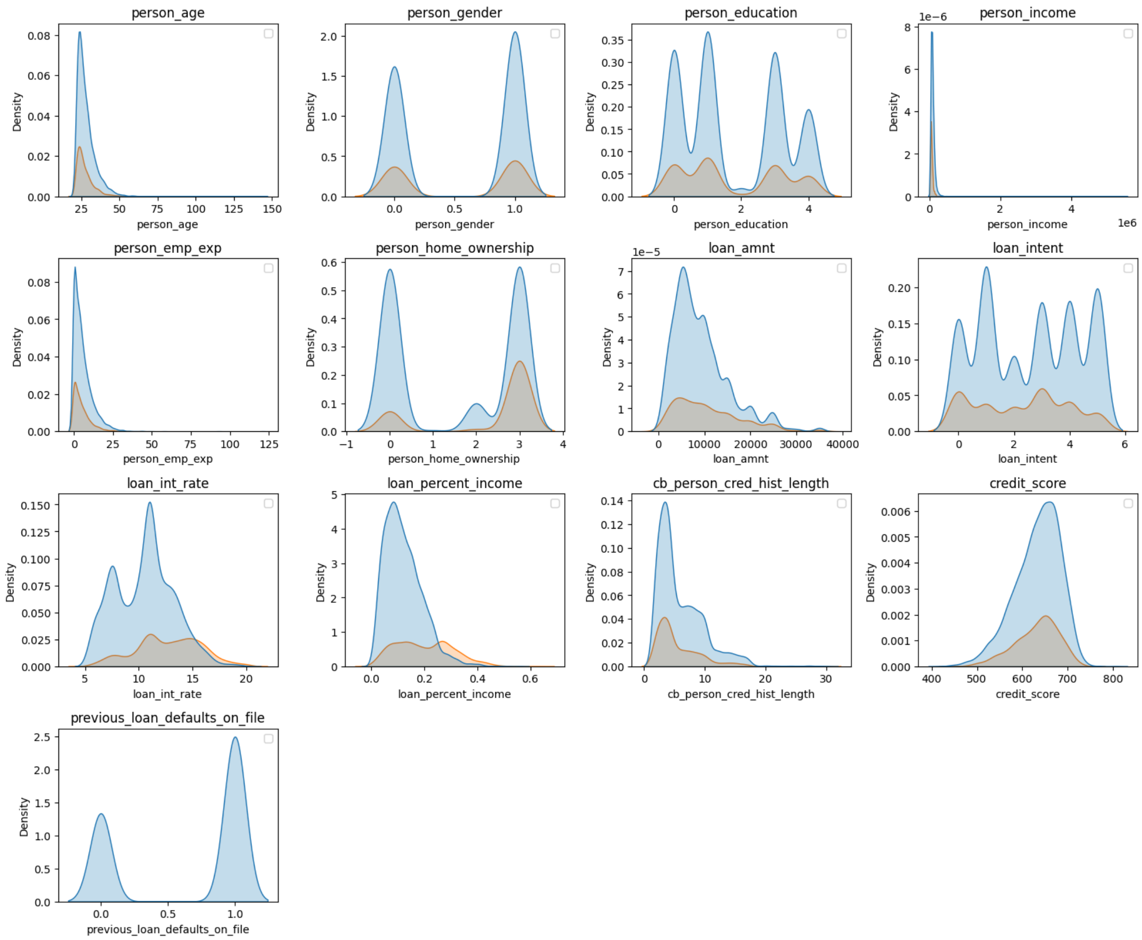
Out[7]:
```
loan_status
0    7000
1    2000
Name: count, dtype: int64
```

taking sample of the dataset since its too big for the project.

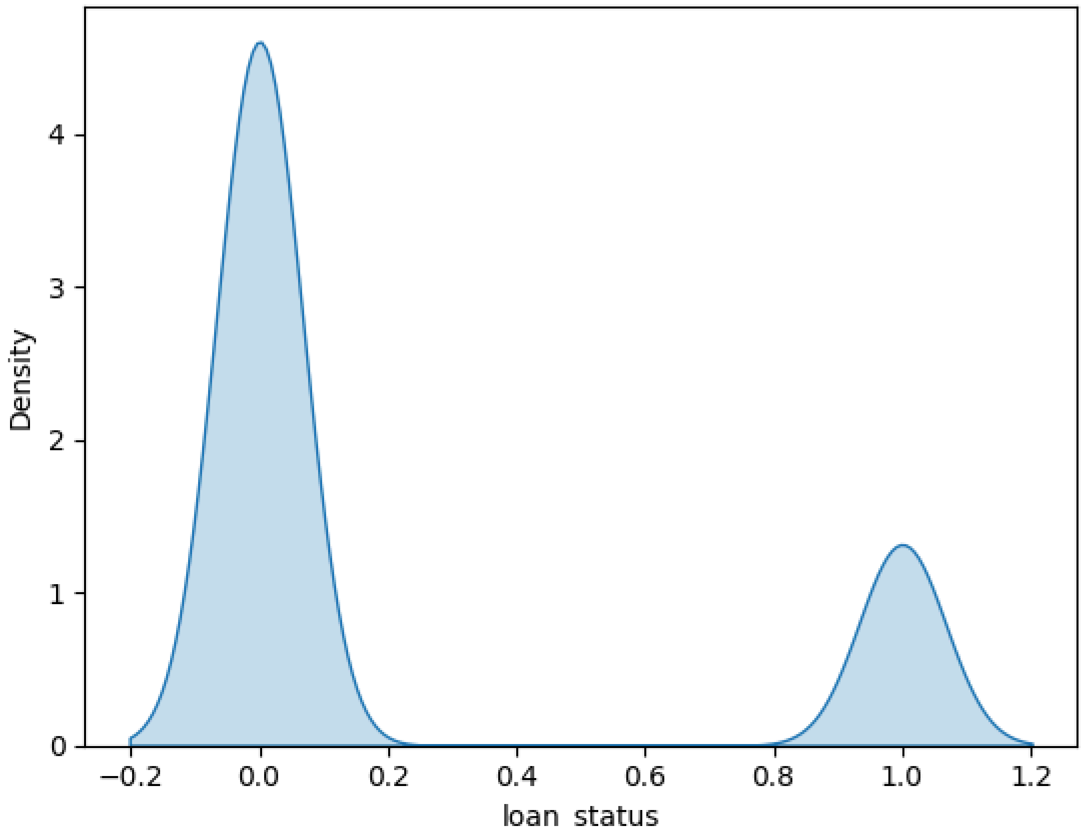
Here is the correlation charts of the data.



Here is the histogram of features based on the target value.



Density graph of labels.



4. Metrics Calculations and Common Training functions.

In [8]:
```
def matrics_cal(y_test, y_pred, y_proba = None):
    matrics = {}
    matrics["TP"] = sum(np.where(y_test & y_pred, 1, 0))
    matrics["TN"] = sum(np.where( (y_test == 0) & (y_pred == 0), 1, 0))
    matrics["FP"] = sum(np.where( (y_test == 0) & (y_pred == 1), 1, 0))
    matrics["FN"] = sum(np.where( (y_test == 1) & (y_pred == 0), 1, 0))

    matrics["TPR"] =  round(matrics["TP"] / (matrics["TP"] +  matrics["FN"]),3)
    matrics["TNR"] =  round(matrics["TN"] / (matrics["TN"] +  matrics["FP"]),3)
    matrics["FPR"] =  round(matrics["FP"] / (matrics["FP"] +  matrics["TN"]),3)
    matrics["FNR"] =  round(matrics["FN"] / (matrics["TP"] +  matrics["FN"]),3)

    matrics["Accuracy"] = round((matrics["TP"] + matrics["TN"]) / (matrics["TP"] +
    matrics["Precision"] = round(matrics["TP"] / (matrics["TP"] +  matrics["FP"]),3
    matrics["F1"] = 2 * round(((matrics["Precision"] * matrics["TPR"]) / (matrics["

    matrics["brier_score"] = round(brier_score_loss(y_test, y_proba),3)
    matrics["AUC"] =  round(roc_auc_score(y_test, y_proba),3)
    reference_prob = np.mean(y_test)
    reference_brier_score = brier_score_loss(y_test, [reference_prob] * len(y_test)
    matrics["BSS"] = round(1 - (matrics["brier_score"] / reference_brier_score),3)

    return matrics
```

In [9]:
```
def train(clf, X, y):

    kf = KFold(n_splits=10, shuffle=True, random_state=42)
    metrics_list = []

    for i, (train_index, test_index) in enumerate(kf.split(X), start=1):
        # Splitting the data
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        clf.fit(X_train, y_train)

        y_pred = clf.predict(X_test)
        y_pred_proba = clf.predict_proba(X_test)[:, 1]


        mat = matrics_cal(y_test, y_pred, y_pred_proba)
        print(f"Fold {i}: {mat}")

        metrics_list.append(mat)

    return metrics_list, y_pred_proba
```

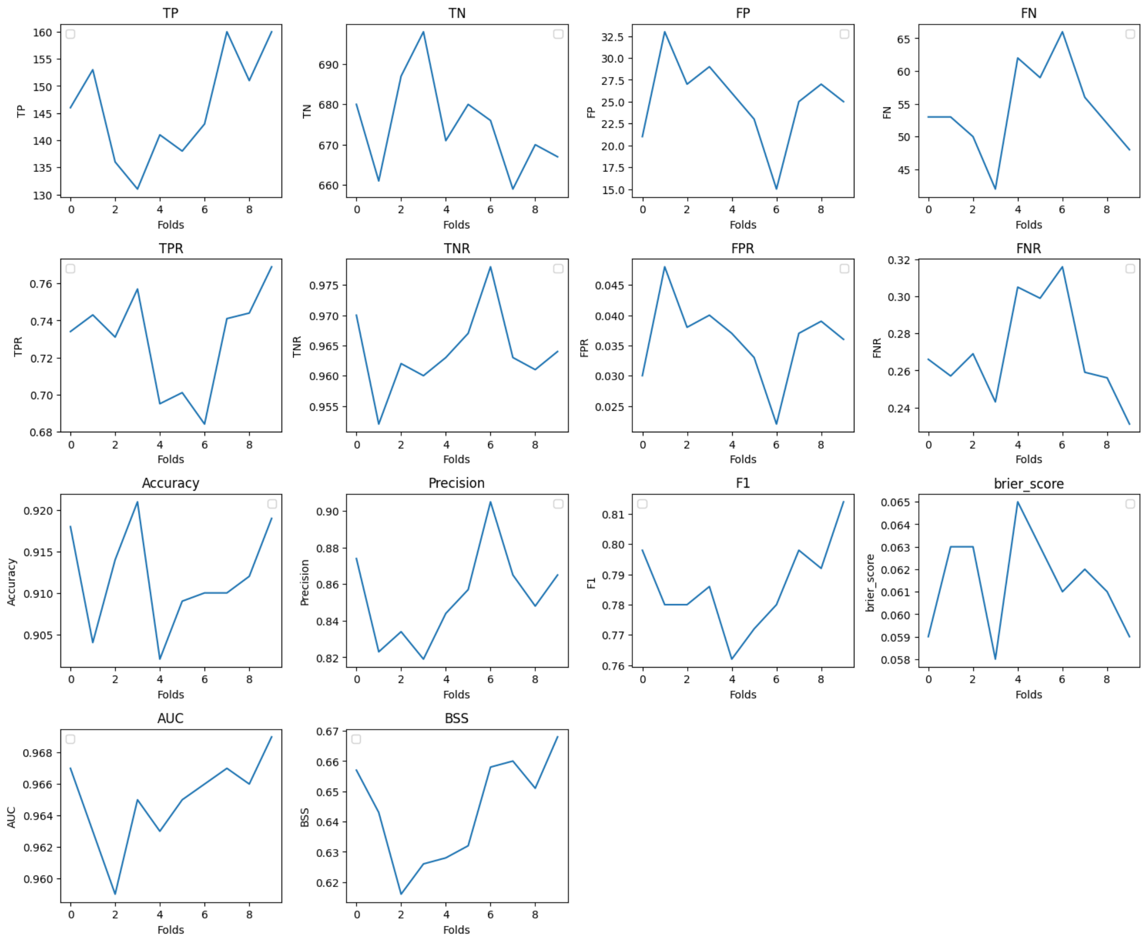
In [10]:
```
def plot_matrics(matrics):
    plt.figure(figsize=(15,15))
    for ax, col in enumerate(matrics.columns):
        plt.subplot(5,4, ax+1)
        plt.title(col)
        sns.lineplot(data=matrics, x=matrics.index, y=col)
        plt.xlabel("Folds")
        plt.legend()
```

```
plt.tight_layout()
```

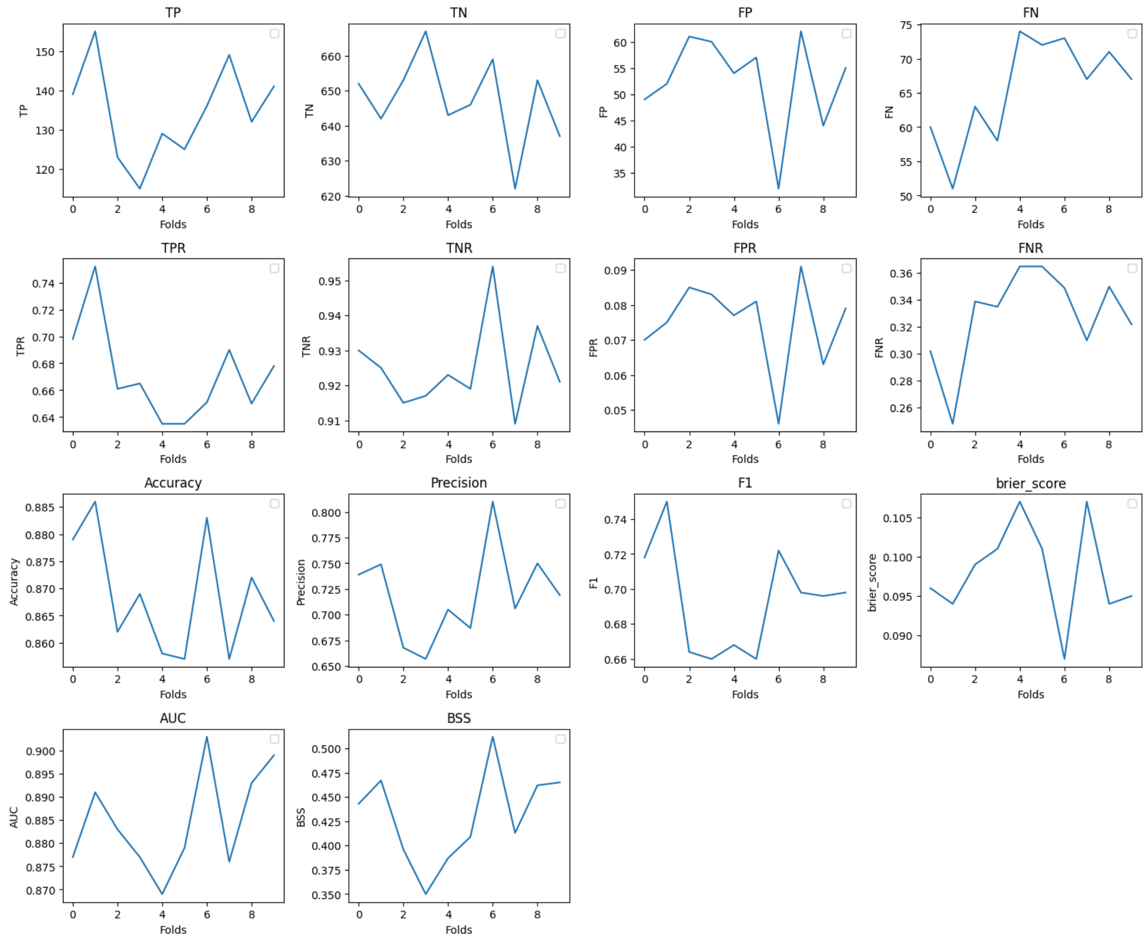
5 Training.

Below are the plots of metrics for each fold in training.

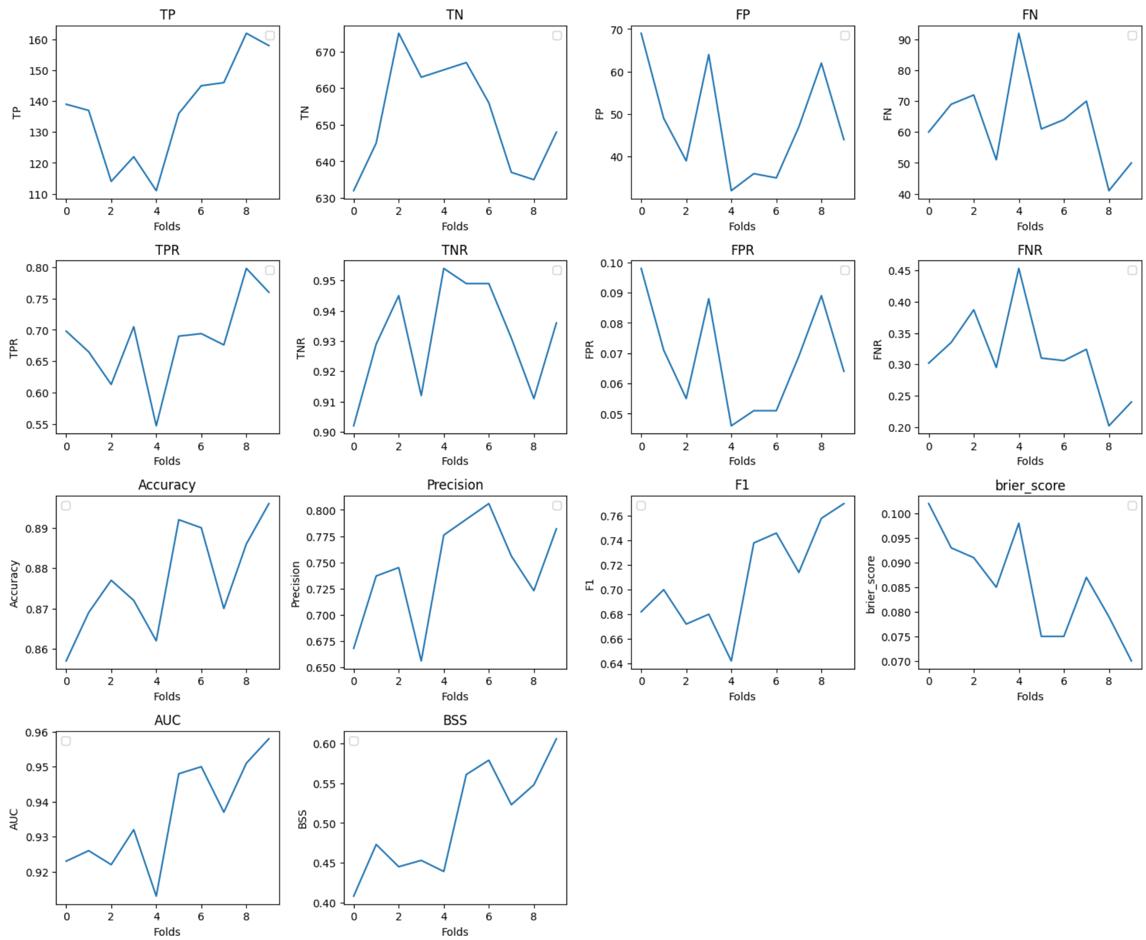
5.1 Random Forest



5.2 KNN


TP
TN
FP
FN
TPR
TNR
FPR
FNR
Accuracy
Precision
F1
brier_score
AUC
BSS
Folds

5.3 LSTM



6 Metrics Comparison

6.1 Random forest

	TP	TN	FP	FN	TPR	TNR	FPR	FNR	Accuracy	Precision	F1	brier_score	AUC	BSS
0	146	680	21	53	0.734	0.970	0.030	0.266	0.918	0.874	0.798	0.059	0.967	0.657
1	153	661	33	53	0.743	0.952	0.048	0.257	0.904	0.823	0.780	0.063	0.963	0.643
2	136	687	27	50	0.731	0.962	0.038	0.269	0.914	0.834	0.780	0.063	0.959	0.616
3	131	698	29	42	0.757	0.960	0.040	0.243	0.921	0.819	0.786	0.058	0.965	0.626
4	141	671	26	62	0.695	0.963	0.037	0.305	0.902	0.844	0.762	0.065	0.963	0.628

```
5       138     680     23      59      0.701   0.967   0.033
0.299   0.909           0.857           0.772   0.063
0.965   0.632
6       143     676     15      66      0.684   0.978   0.022
0.316   0.910           0.905           0.780   0.061
0.966   0.658
7       160     659     25      56      0.741   0.963   0.037
0.259   0.910           0.865           0.798   0.062
0.967   0.660
8       151     670     27      52      0.744   0.961   0.039
0.256   0.912           0.848           0.792   0.061
0.966   0.651
9       160     667     25      48      0.769   0.964   0.036
0.231   0.919           0.865           0.814   0.059
0.969   0.668
```

6.2 KNN

	TP	TN	FP	FN	TPR	TNR	FPR	FNR	Accuracy	Precision	F1	brier_score	AUC	BSS
0	139	652	49	60	0.698	0.930	0.070	0.302	0.879	0.739	0.718	0.096	0.877	0.443
1	155	642	52	51	0.752	0.925	0.075	0.248	0.886	0.749	0.750	0.094	0.891	0.467
2	123	653	61	63	0.661	0.915	0.085	0.339	0.862	0.668	0.664	0.099	0.883	0.396
3	115	667	60	58	0.665	0.917	0.083	0.335	0.869	0.657	0.660	0.101	0.877	0.350
4	129	643	54	74	0.635	0.923	0.077	0.365	0.858	0.705	0.668	0.107	0.869	0.387
5	125	646	57	72	0.635	0.919	0.081	0.365	0.857	0.687	0.660	0.101	0.879	0.409
6	136	659	32	73	0.651	0.954	0.046	0.349	0.883	0.810	0.722	0.087	0.903	0.512
7	149	622	62	67	0.690	0.909	0.091	0.310	0.857	0.706	0.698	0.107	0.876	0.413
8	132	653	44	71	0.650	0.937	0.063	0.350	0.872	0.750	0.696	0.094	0.893	0.462
9	141	637	55	67	0.678	0.921	0.079	0.322	0.864	0.719	0.698	0.095	0.899	0.465

6.3 LSTM

	TP	TN	FP	FN	TPR	TNR	FPR	FNR	Accuracy	Precision	F1	brier_score	AUC	BSS
0	139	632	69	60	0.698	0.902	0.098	0.302	0.857	0.668	0.682	0.102	0.923	0.408
1	137	645	49	69	0.665	0.929	0.071	0.335	0.869	0.737	0.700	0.093	0.926	0.473
2	114	675	39	72	0.613	0.945	0.055	0.387	0.877	0.745	0.672	0.091	0.922	0.445
3	122	663	64	51	0.705	0.912	0.088	0.295	0.872	0.656	0.680	0.085	0.932	0.453

```
4       111     665     32      92      0.547   0.954   0.046
0.453   0.862           0.776           0.642   0.098
0.913   0.439
5       136     667     36      61      0.690   0.949   0.051
0.310   0.892           0.791           0.738   0.075
0.948   0.561
6       145     656     35      64      0.694   0.949   0.051
0.306   0.890           0.806           0.746   0.075
0.950   0.579
7       146     637     47      70      0.676   0.931   0.069
0.324   0.870           0.756           0.714   0.087
0.937   0.523
8       162     635     62      41      0.798   0.911   0.089
0.202   0.886           0.723           0.758   0.079
0.951   0.548
9       158     648     44      50      0.760   0.936   0.064
0.240   0.896           0.782           0.770   0.070
0.958   0.606
```

6.4 Average Comparison

	Random Forest	KNN	LSTM
TP	145.9000	134.4000	137.0000
TN	674.9000	647.4000	652.3000
FP	25.1000	52.6000	47.7000
FN	54.1000	65.6000	63.0000
TPR	0.7299	0.6715	0.6846
TNR	0.9640	0.9250	0.9318
FPR	0.0360	0.0750	0.0682
FNR	0.2701	0.3285	0.3154
Accuracy	0.9119	0.8687	0.8771
Precision	0.8534	0.7190	0.7440
F1	0.7862	0.6934	0.7102
brier_score	0.0614	0.0981	0.0855
AUC	0.9650	0.8847	0.9360
BSS	0.6439	0.4304	0.5035

6.5 Foldwise Comparison

6.5.1 Fold 1:

	Random Forest	KNN	LSTM
TP	146.000	139.000	139.000
TN	680.000	652.000	632.000
FP	21.000	49.000	69.000
FN	53.000	60.000	60.000
TPR	0.734	0.698	0.698

TNR	0.970	0.930	0.902
FPR	0.030	0.070	0.098
FNR	0.266	0.302	0.302
Accuracy	0.918	0.879	0.857
Precision	0.874	0.739	0.668
F1	0.798	0.718	0.682
brier_score	0.059	0.096	0.102
AUC	0.967	0.877	0.923
BSS	0.657	0.443	0.408

Fold 2:

	Random Forest	KNN	LSTM
TP	153.000	155.000	137.000
TN	661.000	642.000	645.000
FP	33.000	52.000	49.000
FN	53.000	51.000	69.000
TPR	0.743	0.752	0.665
TNR	0.952	0.925	0.929
FPR	0.048	0.075	0.071
FNR	0.257	0.248	0.335
Accuracy	0.904	0.886	0.869
Precision	0.823	0.749	0.737
F1	0.780	0.750	0.700
brier_score	0.063	0.094	0.093
AUC	0.963	0.891	0.926
BSS	0.643	0.467	0.473

Fold 3:

	Random Forest	KNN	LSTM
TP	136.000	123.000	114.000
TN	687.000	653.000	675.000
FP	27.000	61.000	39.000
FN	50.000	63.000	72.000
TPR	0.731	0.661	0.613
TNR	0.962	0.915	0.945
FPR	0.038	0.085	0.055
FNR	0.269	0.339	0.387
Accuracy	0.914	0.862	0.877
Precision	0.834	0.668	0.745
F1	0.780	0.664	0.672
brier_score	0.063	0.099	0.091
AUC	0.959	0.883	0.922
BSS	0.616	0.396	0.445

Fold 4:

	Random Forest	KNN	LSTM
TP	131.000	115.000	122.000
TN	698.000	667.000	663.000
FP	29.000	60.000	64.000
FN	42.000	58.000	51.000
TPR	0.757	0.665	0.705
TNR	0.960	0.917	0.912
FPR	0.040	0.083	0.088
FNR	0.243	0.335	0.295
Accuracy	0.921	0.869	0.872
Precision	0.819	0.657	0.656
F1	0.786	0.660	0.680
brier_score	0.058	0.101	0.085
AUC	0.965	0.877	0.932
BSS	0.626	0.350	0.453

Fold 5:

	Random Forest	KNN	LSTM
TP	141.000	129.000	136.000
TN	671.000	643.000	633.000
FP	26.000	54.000	64.000
FN	62.000	74.000	67.000
TPR	0.695	0.635	0.670
TNR	0.963	0.923	0.908
FPR	0.037	0.077	0.092
FNR	0.305	0.365	0.330
Accuracy	0.902	0.858	0.854
Precision	0.844	0.705	0.680
F1	0.762	0.668	0.674
brier_score	0.065	0.107	0.100
AUC	0.963	0.869	0.915
BSS	0.628	0.387	0.428

Fold 6:

	Random Forest	KNN	LSTM
TP	138.000	125.000	134.000
TN	680.000	646.000	674.000
FP	23.000	57.000	29.000
FN	59.000	72.000	63.000
TPR	0.701	0.635	0.680
TNR	0.967	0.919	0.959
FPR	0.033	0.081	0.041
FNR	0.299	0.365	0.320

Accuracy	0.909	0.857	0.898
Precision	0.857	0.687	0.822
F1	0.772	0.660	0.744
brier_score	0.063	0.101	0.075
AUC	0.965	0.879	0.949
BSS	0.632	0.409	0.561

Fold 7:

	Random Forest	KNN	LSTM
TP	143.000	136.000	138.000
TN	676.000	659.000	664.000
FP	15.000	32.000	27.000
FN	66.000	73.000	71.000
TPR	0.684	0.651	0.660
TNR	0.978	0.954	0.961
FPR	0.022	0.046	0.039
FNR	0.316	0.349	0.340
Accuracy	0.910	0.883	0.891
Precision	0.905	0.810	0.836
F1	0.780	0.722	0.738
brier_score	0.061	0.087	0.078
AUC	0.966	0.903	0.950
BSS	0.658	0.512	0.563

Fold 8:

	Random Forest	KNN	LSTM
TP	160.000	149.000	139.000
TN	659.000	622.000	642.000
FP	25.000	62.000	42.000
FN	56.000	67.000	77.000
TPR	0.741	0.690	0.644
TNR	0.963	0.909	0.939
FPR	0.037	0.091	0.061
FNR	0.259	0.310	0.356
Accuracy	0.910	0.857	0.868
Precision	0.865	0.706	0.768
F1	0.798	0.698	0.700
brier_score	0.062	0.107	0.088
AUC	0.967	0.876	0.938
BSS	0.660	0.413	0.518

Fold 9:

	Random Forest	KNN	LSTM
TP	151.000	132.000	152.000
TN	670.000	653.000	653.000
FP	27.000	44.000	44.000

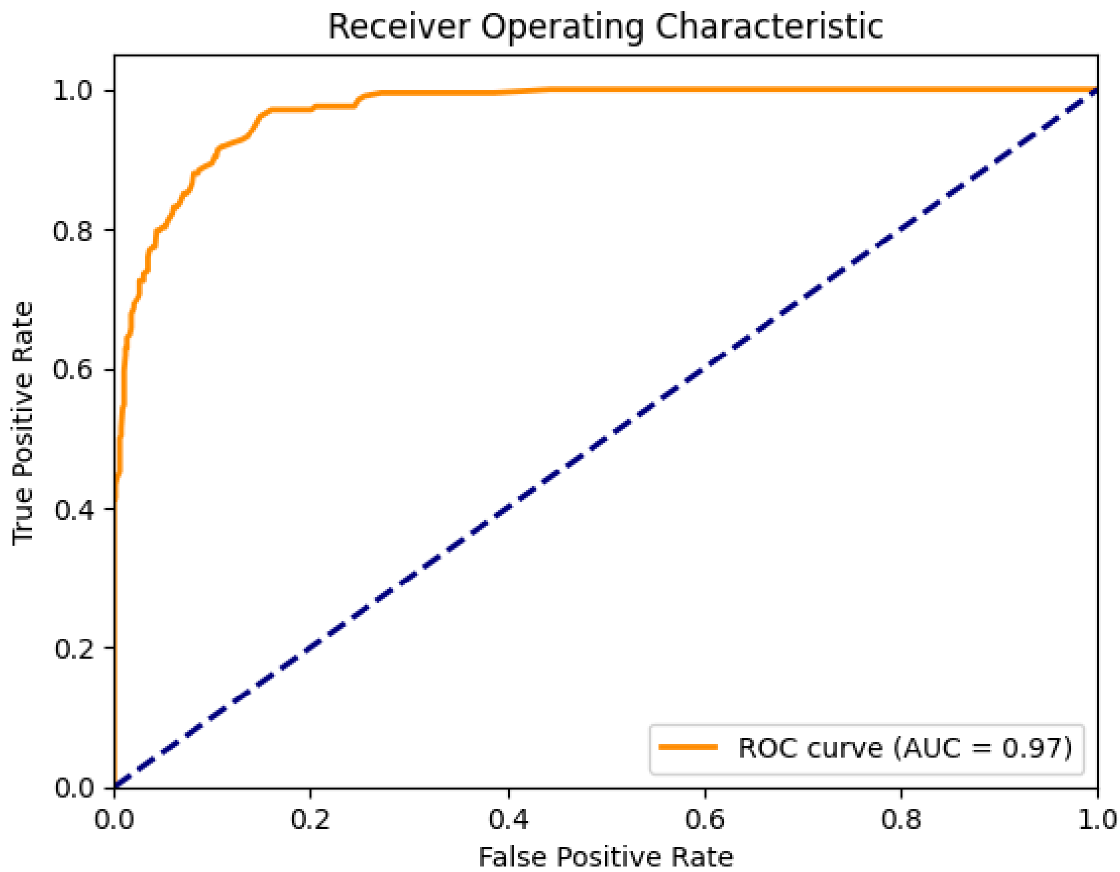
FN	52.000	71.000	51.000
TPR	0.744	0.650	0.749
TNR	0.961	0.937	0.937
FPR	0.039	0.063	0.063
FNR	0.256	0.350	0.251
Accuracy	0.912	0.872	0.894
Precision	0.848	0.750	0.776
F1	0.792	0.696	0.762
brier_score	0.061	0.094	0.075
AUC	0.966	0.893	0.950
BSS	0.651	0.462	0.571

Fold 10:

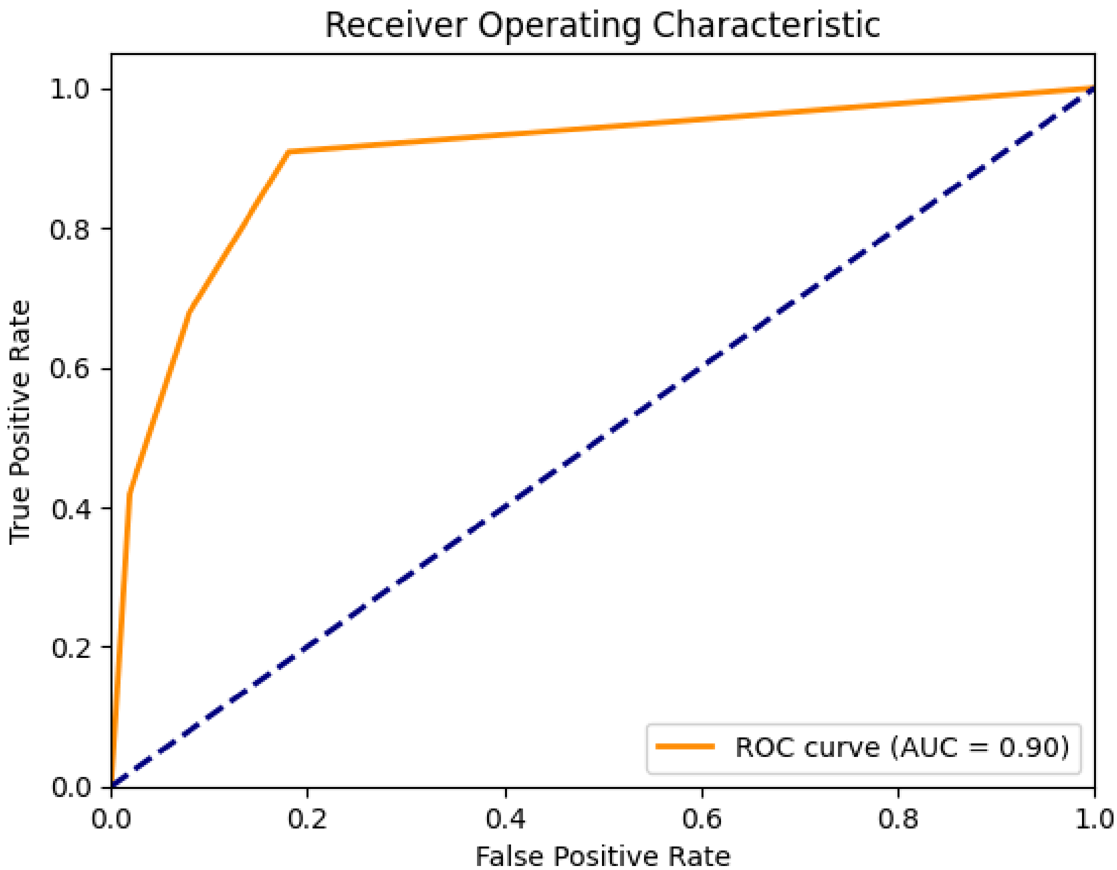
	Random Forest	KNN	LSTM
TP	160.000	141.000	154.000
TN	667.000	637.000	657.000
FP	25.000	55.000	35.000
FN	48.000	67.000	54.000
TPR	0.769	0.678	0.740
TNR	0.964	0.921	0.949
FPR	0.036	0.079	0.051
FNR	0.231	0.322	0.260
Accuracy	0.919	0.864	0.901
Precision	0.865	0.719	0.815
F1	0.814	0.698	0.776
brier_score	0.059	0.095	0.070
AUC	0.969	0.899	0.959
BSS	0.668	0.465	0.606

6.6 ROC curves

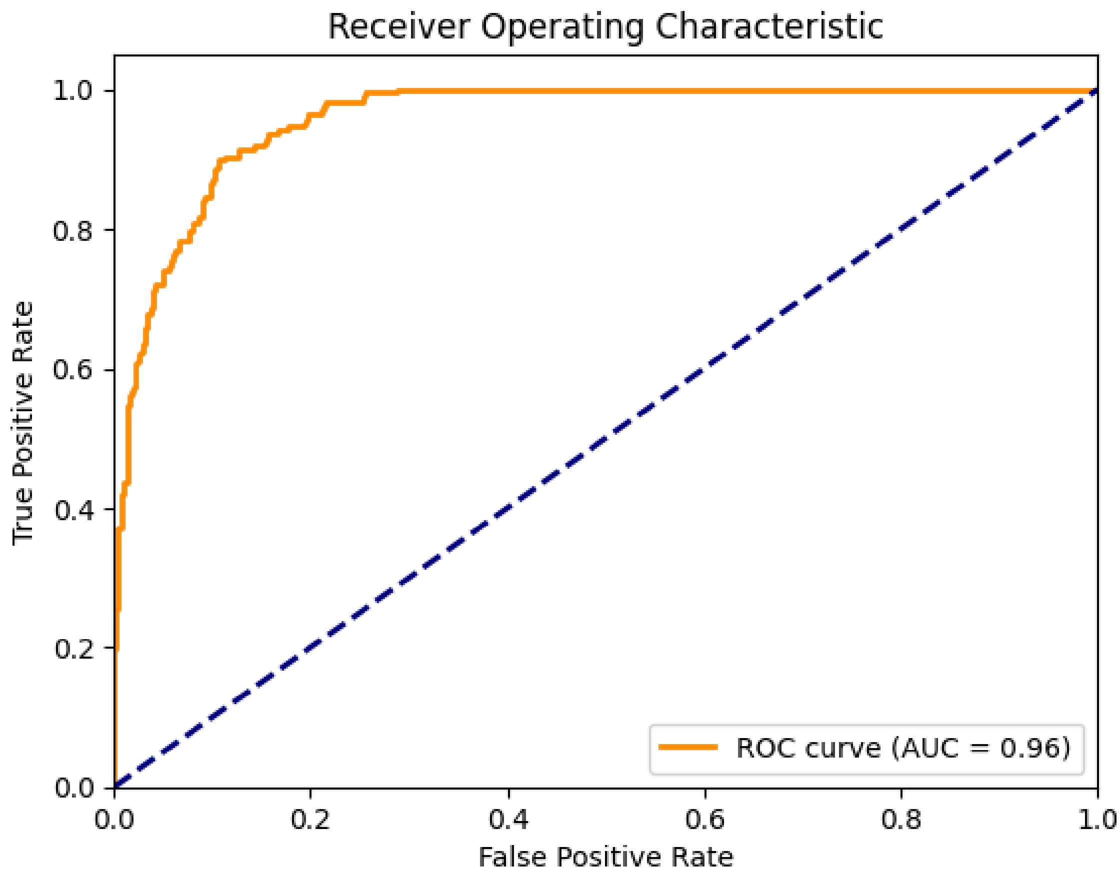
6.6.1 Random Forest


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
ROC curve (AUC = 0.97)
0.0
0.2
0.4
0.6
0.8
1.0

6.6.2 KNN


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
ROC curve (AUC = 0.90)
0.0
0.2
0.4
0.6
0.8
1.0

6.6.3 LSTM



7. Conculsion

Random Forest outperforms KNN and LSTM on all critical measures. It has the best accuracy (91.19%), AUC (0.965), and precision (0.8534) while achieving the lowest Brier score (0.0614) and FPR (0.0360), showing good classification and calibration. LSTM performs moderately, with higher precision (0.7440) and F1-score (0.7102) than KNN, but it falls short of Random Forest in accuracy (87.71%) and calibration. KNN performs poorly, with the lowest accuracy (86.87%), AUC (0.8847), and calibration scores. Overall, Random Forest is the most trustworthy option, with LSTM coming in second.