In [1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn as sk
```

In [2]:
```python
df = pd.read_csv('credit_risk_dataset.csv')
```

In [3]:
```python
df
```

Out[3]:

	person_age	person_income	person_home_ownership	person_emp_length	
0	22	59000	RENT	123.0	
1	21	9600	OWN	5.0	
2	25	9600	MORTGAGE	1.0	
3	23	65500	RENT	4.0	
4	24	54400	RENT	8.0	
...	...	...	...	...	
32576	57	53000	MORTGAGE	1.0	
32577	54	120000	MORTGAGE	4.0	
32578	65	76000	RENT	3.0	HOMEIMF
32579	56	150000	MORTGAGE	5.0	
32580	66	42000	RENT	2.0	

32581 rows × 12 columns

In [4]:
```python
df.shape
```

Out[4]: (32581, 12)

In [7]:
```python
df.head()
```

Out[7]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	l
0	22	59000	RENT	123.0	PERSONAL	
1	21	9600	OWN	5.0	EDUCATION	
2	25	9600	MORTGAGE	1.0	MEDICAL	
3	23	65500	RENT	4.0	MEDICAL	
4	24	54400	RENT	8.0	MEDICAL	

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32581 entries, 0 to 32580
Data columns (total 12 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   person_age                  32581 non-null  int64  
 1   person_income               32581 non-null  int64  
 2   person_home_ownership       32581 non-null  object 
 3   person_emp_length           31686 non-null  float64
 4   loan_intent                 32581 non-null  object 
 5   loan_grade                  32581 non-null  object 
 6   loan_amnt                   32581 non-null  int64  
 7   loan_int_rate               29465 non-null  float64
 8   loan_status                 32581 non-null  int64  
 9   loan_percent_income         32581 non-null  float64
 10  cb_person_default_on_file   32581 non-null  object 
 11  cb_person_cred_hist_length  32581 non-null  int64  
dtypes: float64(3), int64(5), object(4)
memory usage: 3.0+ MB
```

In [9]:
```
data_rows = df.shape[0]
data_colunms = df.shape[1]

print(f'This dataset have {data_rows} rows and {data_colunms} colum
```

```
This dataset have 32581 rows and 12 columns.
```

In [10]:
```
df.isnull().sum()
```

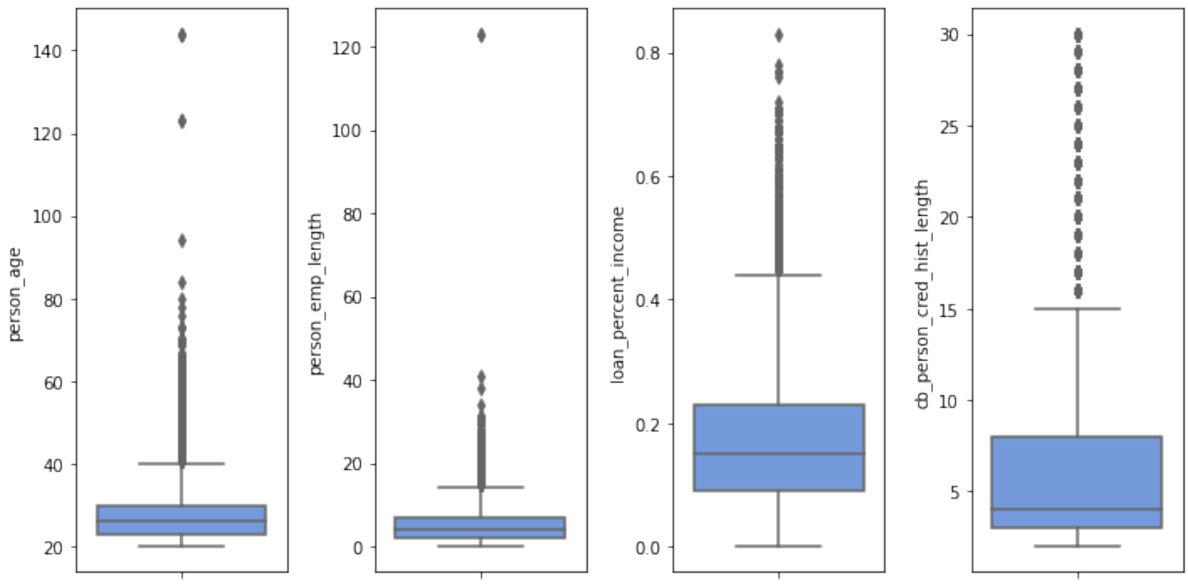
Out[10]:
```
person_age                       0
person_income                    0
person_home_ownership            0
person_emp_length              895
loan_intent                      0
loan_grade                       0
loan_amnt                        0
loan_int_rate                 3116
loan_status                      0
loan_percent_income              0
cb_person_default_on_file        0
cb_person_cred_hist_length       0
dtype: int64
```

In [11]:
```
df.describe()
```

Out[11]:

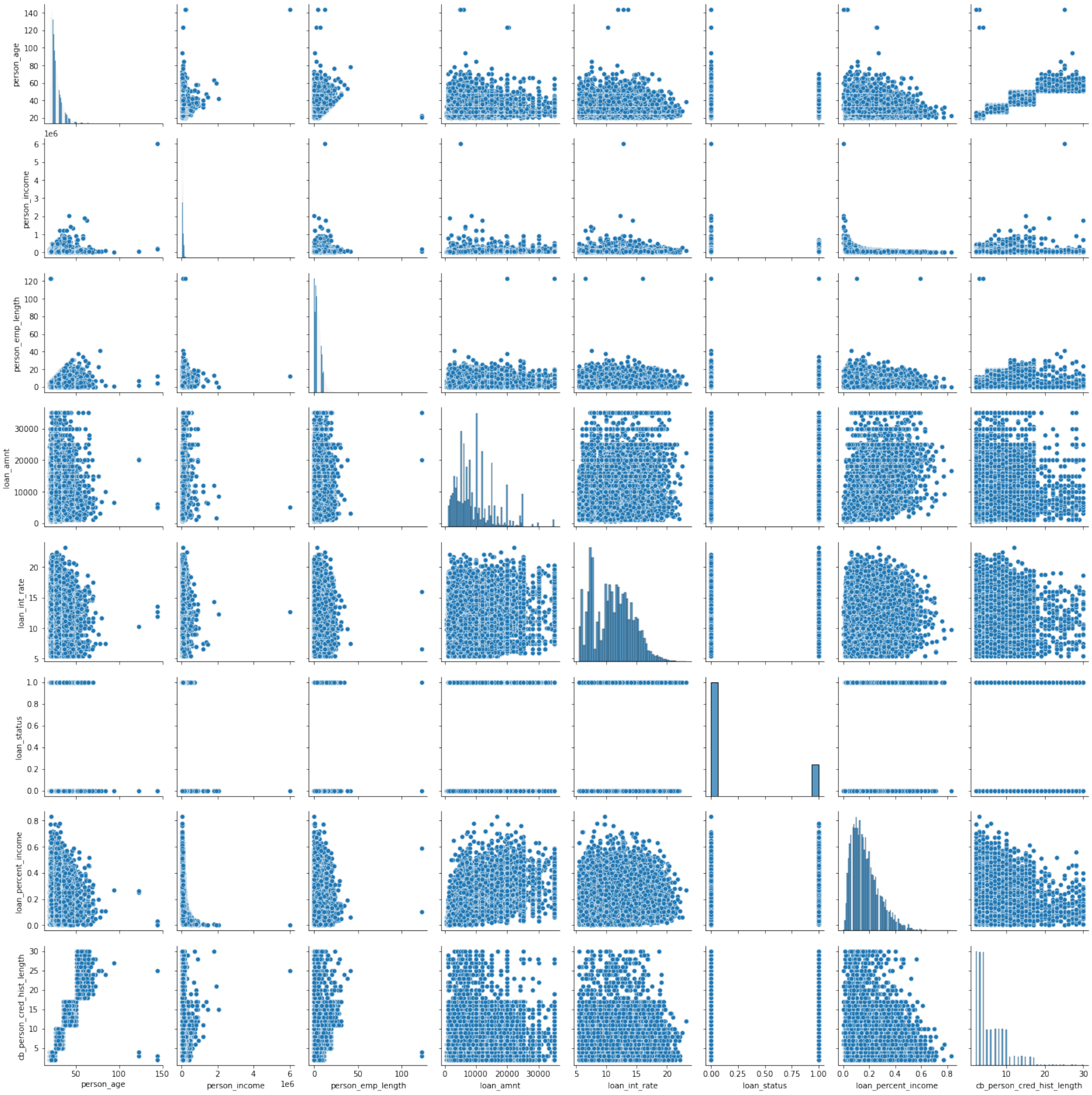
	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loa
count	32581.000000	3.258100e+04	31686.000000	32581.000000	29465.000000	3258
mean	27.734600	6.607485e+04	4.789686	9589.371106	11.011695	
std	6.348078	6.198312e+04	4.142630	6322.086646	3.240459	
min	20.000000	4.000000e+03	0.000000	500.000000	5.420000	
25%	23.000000	3.850000e+04	2.000000	5000.000000	7.900000	
50%	26.000000	5.500000e+04	4.000000	8000.000000	10.990000	
75%	30.000000	7.920000e+04	7.000000	12200.000000	13.470000	
max	144.000000	6.000000e+06	123.000000	35000.000000	23.220000	

In [12]:
```
features = ['person_age','person_emp_length','loan_percent_income',
plt.figure(figsize=(10,5))
for i in range(0,len(features)):
    plt.subplot(1, len(features), i + 1)
    sns.boxplot(y=df[features[i]], color='CornflowerBlue', orient='
    plt.tight_layout()
```



In [13]:
```
sns.pairplot(df)
```

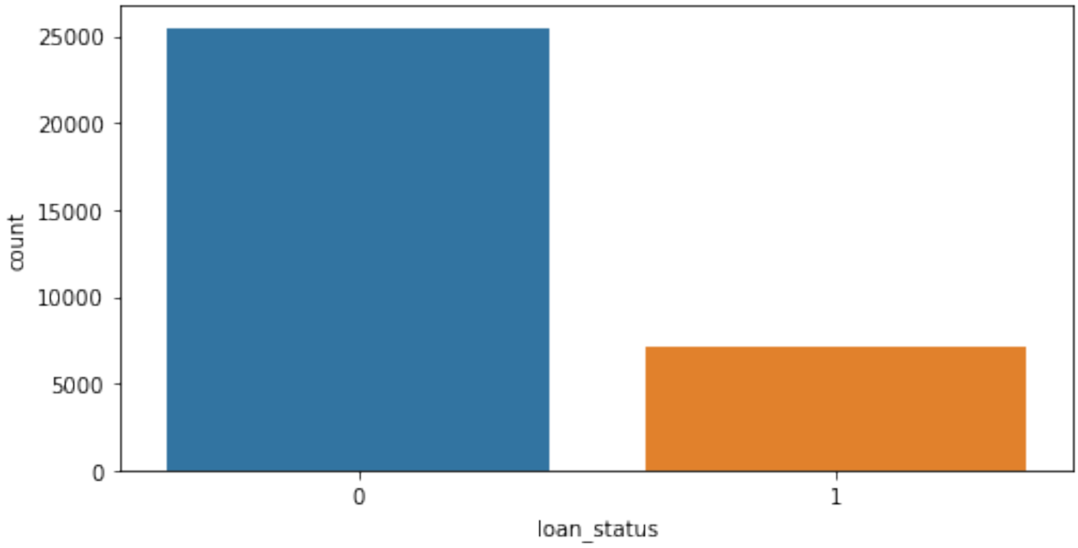
Out[13]: <seaborn.axisgrid.PairGrid at 0x7fdc6b67b5e0>



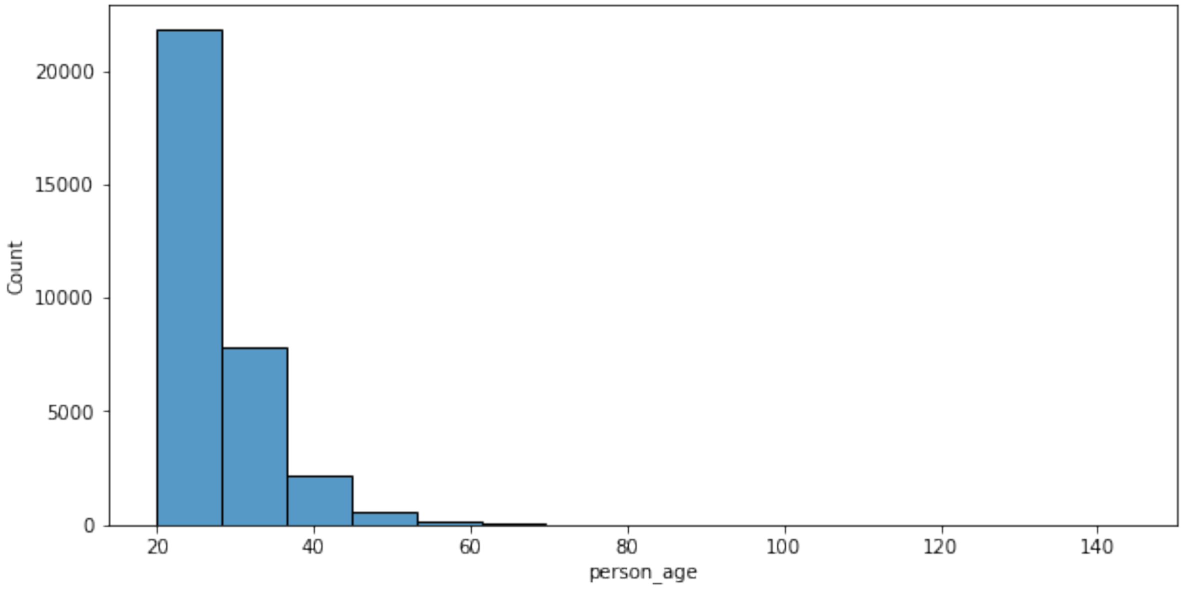
In [14]:
```
df.loan_status.value_counts()
```

Out[14]:
```
0    25473
1     7108
Name: loan_status, dtype: int64
```

In [15]:
```
plt.figure(figsize=(8,4))
sns.countplot(x='loan_status', data=df)
plt.show()
```



In [16]:
```
plt.figure(figsize=(10,5))
sns.histplot(data=df, x= 'person_age', bins=15)
plt.show()
```

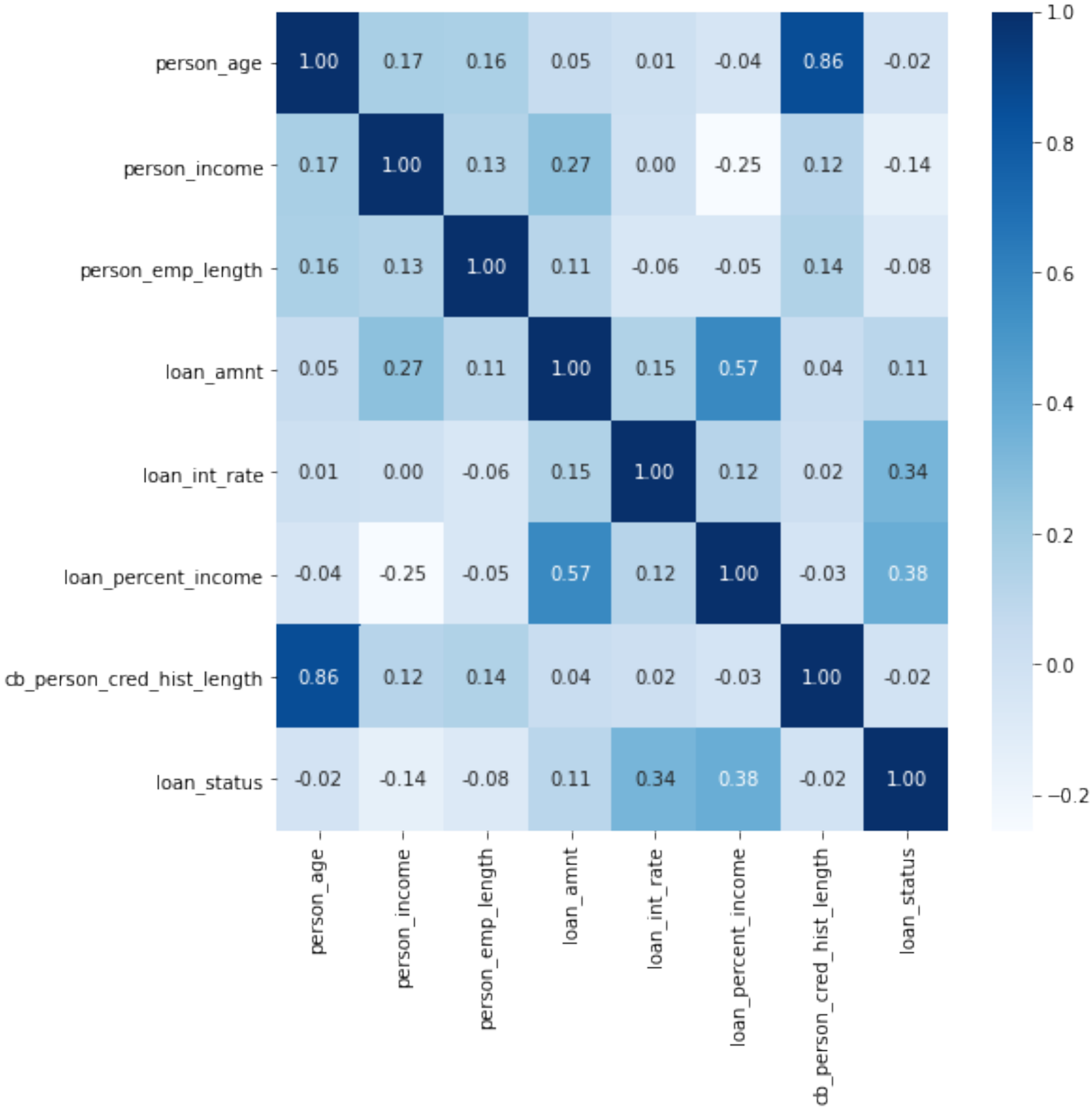


In [3]:
```
variables = ['person_age','person_income','person_emp_length', 'loa
credit_risk_corr = df[variables].corr()
credit_risk_corr
```

Out[3]:

	person_age	person_income	person_emp_length	loan_amnt	lo
person_age	1.000000	0.173202	0.163106	0.050787	
person_income	0.173202	1.000000	0.134268	0.266820	
person_emp_length	0.163106	0.134268	1.000000	0.113082	
loan_amnt	0.050787	0.266820	0.113082	1.000000	
loan_int_rate	0.012580	0.000792	-0.056405	0.146813	
loan_percent_income	-0.042411	-0.254471	-0.054111	0.572612	
cb_person_cred_hist_length	0.859133	0.117987	0.144699	0.041967	
loan_status	-0.021629	-0.144449	-0.082489	0.105376	

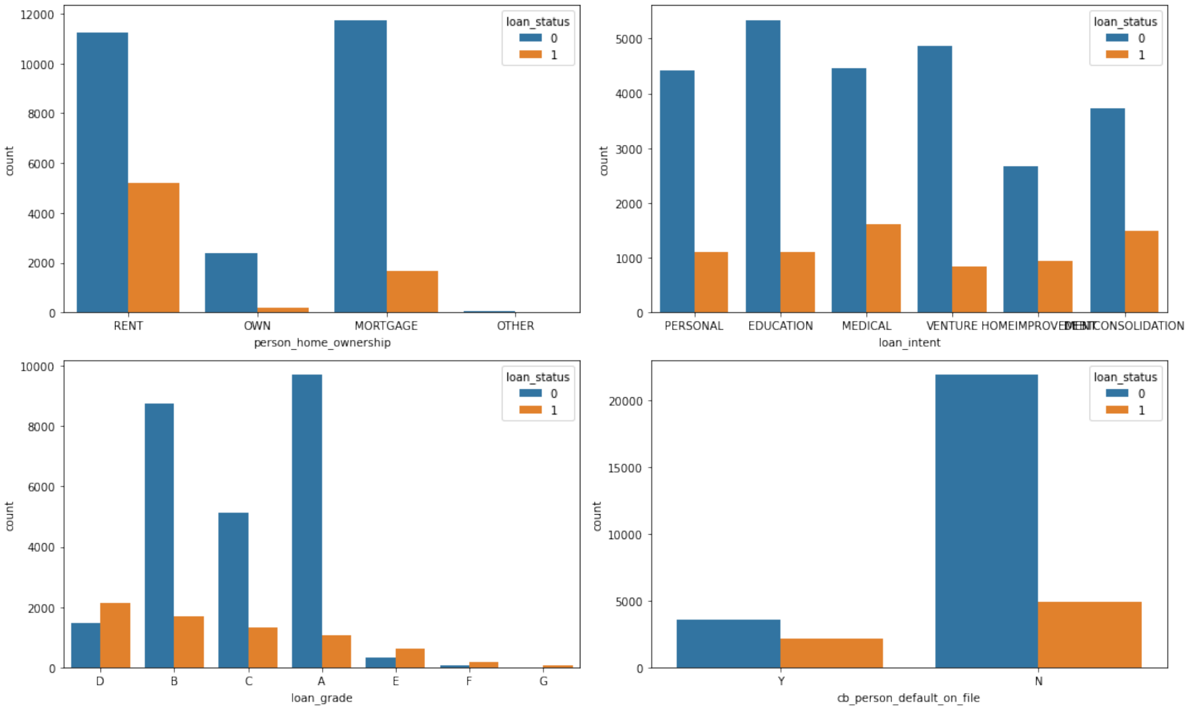
In [4]:
```
plt.figure(figsize=(8,8))
sns.heatmap(credit_risk_corr, cmap='Blues', annot=True, fmt='.2f')
plt.show()
```



In [5]:
```
cat_cols = ['person_home_ownership', 'loan_intent', 'loan_grade', '
for i in cat_cols:
    print(f'Total row of variable {i}')
    print(df[i].value_counts())
    print()
```

```
Total row of variable person_home_ownership
RENT        16446
MORTGAGE    13444
OWN          2584
OTHER         107
Name: person_home_ownership, dtype: int64

Total row of variable loan_intent
EDUCATION            6453
MEDICAL              6071
VENTURE              5719
PERSONAL             5521
DEBTCONSOLIDATION    5212
HOMEIMPROVEMENT      3605
Name: loan_intent, dtype: int64

Total row of variable loan_grade
A    10777
B    10451
C     6458
D     3626
E      964
F      241
G       64
Name: loan_grade, dtype: int64

Total row of variable cb_person_default_on_file
N    26836
Y     5745
Name: cb_person_default_on_file, dtype: int64
```

In [6]:
```
plt.figure(figsize=(15,9))
for i in range(0,len(cat_cols)):
    plt.subplot(2,2,i + 1)
    sns.countplot(data= df, x = cat_cols[i], hue='loan_status')
    plt.tight_layout()
```



In [7]:
```
df.drop(df.loc[df['person_emp_length'] == 123].index, inplace=True)
```

In [8]:
```
df.drop(df.loc[df['person_age'] >= 123].index, inplace=True)
```

In [9]:
```
df.loc[df['person_age'] >= 123].index
```

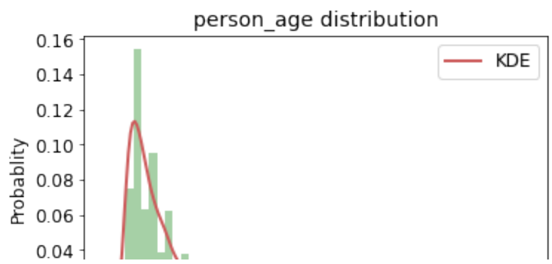
Out[9]:
```
Int64Index([], dtype='int64')
```

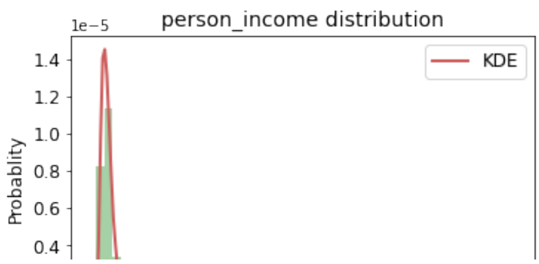
In [10]:
```
num_cols = pd.DataFrame(df[df.select_dtypes(include=['float', 'int'
```

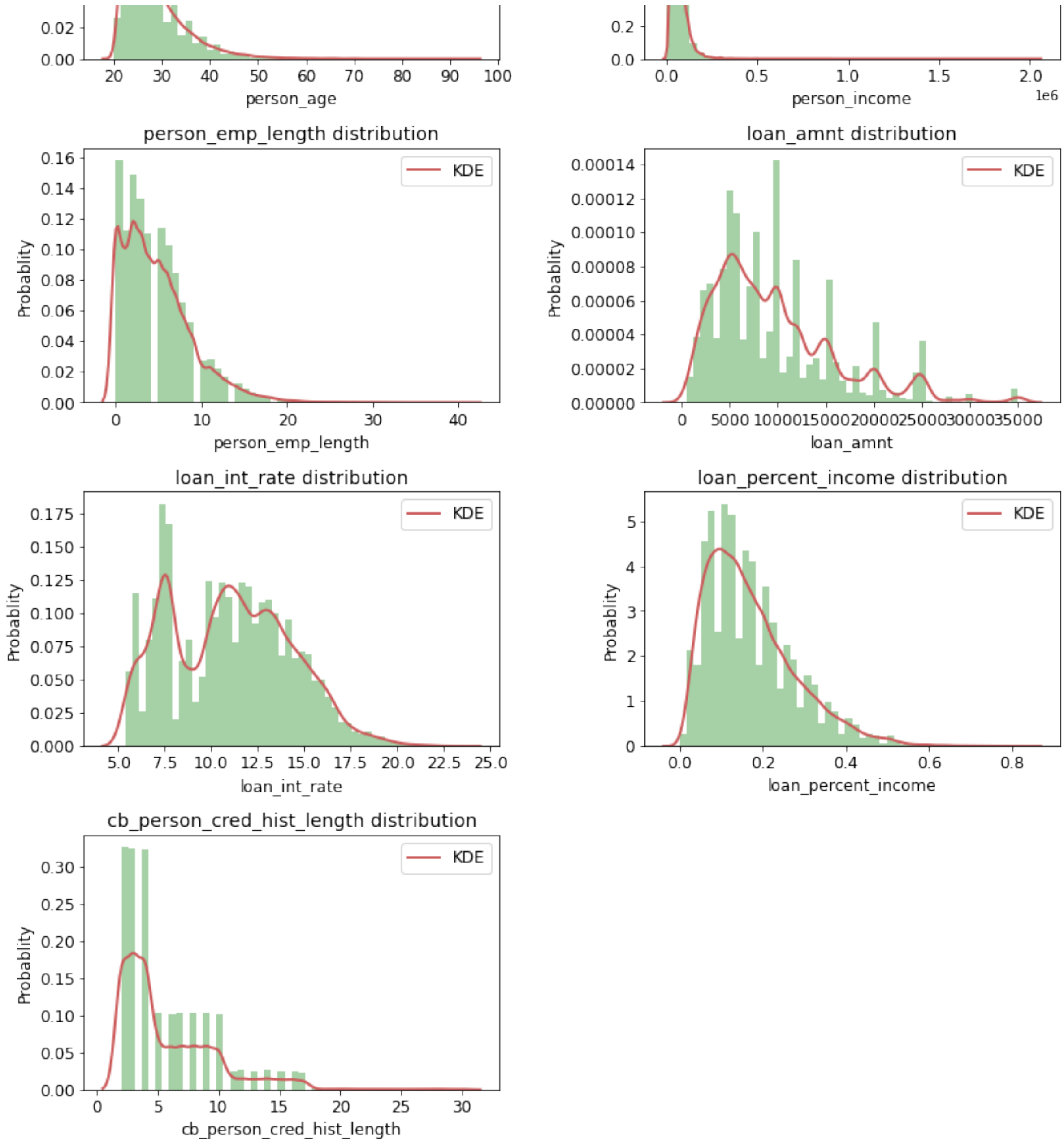
In [15]:
```
num_cols_hist = num_cols.drop(['loan_status'], axis=1)
plt.figure(figsize=(12,16))

for i, col in enumerate(num_cols_hist.columns):
    idx = int('42'+ str(i+1))
    plt.subplot(idx)
    sns.distplot(num_cols_hist[col], color='forestgreen',
                 kde_kws={'color': 'indianred', 'lw': 2, 'label': '
    plt.title(col+' distribution', fontsize=14)
    plt.ylabel('Probablity', fontsize=12)
    plt.xlabel(col, fontsize=12)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    plt.legend(['KDE'], prop={"size":12})
```

```
plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=0.95, h
                    wspace=0.35)
plt.show()
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.p
y:2551: FutureWarning: `distplot` is a deprecated function and wil
l be removed in a future version. Please adapt your code to use ei
ther `displot` (a figure-level function with similar flexibility)
or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.p
y:2551: FutureWarning: `distplot` is a deprecated function and wil
l be removed in a future version. Please adapt your code to use ei
ther `displot` (a figure-level function with similar flexibility)
or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.p
y:2551: FutureWarning: `distplot` is a deprecated function and wil
l be removed in a future version. Please adapt your code to use ei
ther `displot` (a figure-level function with similar flexibility)
or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.p
y:2551: FutureWarning: `distplot` is a deprecated function and wil
l be removed in a future version. Please adapt your code to use ei
ther `displot` (a figure-level function with similar flexibility)
or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.p
y:2551: FutureWarning: `distplot` is a deprecated function and wil
l be removed in a future version. Please adapt your code to use ei
ther `displot` (a figure-level function with similar flexibility)
or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.p
y:2551: FutureWarning: `distplot` is a deprecated function and wil
l be removed in a future version. Please adapt your code to use ei
ther `displot` (a figure-level function with similar flexibility)
or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.p
y:2551: FutureWarning: `distplot` is a deprecated function and wil
l be removed in a future version. Please adapt your code to use ei
ther `displot` (a figure-level function with similar flexibility)
or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```







In [11]:
```
df = df[df['person_age']<=100]
df = df[df['person_emp_length']<=60]
df = df[df['person_income']<=4e6]
```

In [12]:
```
mean_person_emp_length = df['person_emp_length'].mean()
mean_loan_int_rate = df['loan_int_rate'].mean()

df['person_emp_length'] = df['person_emp_length'].fillna(mean_perso
df['loan_int_rate'] = df['loan_int_rate'].fillna(mean_loan_int_rate
```

In [13]:
```
from sklearn.preprocessing import LabelEncoder
```

In [14]:
```
label_encoder = LabelEncoder()
df["loan_intent"]=label_encoder.fit_transform(df["loan_intent"])
df["person_home_ownership"]=label_encoder.fit_transform(df["person_
df["loan_grade"]=label_encoder.fit_transform(df["loan_grade"])
df["cb_person_default_on_file"]=label_encoder.fit_transform(df["cb_
```

In [15]:
```
df.head()
```

Out[15]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	lo
1	21	9600	2	5.0	1	
2	25	9600	0	1.0	3	
3	23	65500	3	4.0	3	
4	24	54400	3	8.0	3	
5	21	9900	2	2.0	5	

In [16]:
```
df["person_age"].max()
```

Out[16]: 94

In [17]:
```
df.isnull().sum()
```

Out[17]:
```
person_age                    0
person_income                 0
person_home_ownership         0
person_emp_length             0
loan_intent                   0
loan_grade                    0
loan_amnt                     0
loan_int_rate                 0
loan_status                   0
loan_percent_income           0
cb_person_default_on_file     0
cb_person_cred_hist_length    0
dtype: int64
```

In [18]:
```
from scipy.stats import uniform, randint
from sklearn import model_selection,linear_model, metrics
from sklearn.metrics import auc, accuracy_score, confusion_matrix,
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
from sklearn.metrics import plot_confusion_matrix
```

In [19]:
```
R_tree=RandomForestClassifier()
lg = LogisticRegression(random_state=42)
N_bayes=GaussianNB()
XGB_model = XGBClassifier(learning_rate=0.1, max_depth=10, scale_po
knn = KNeighborsClassifier(n_neighbors=150)
```

In [20]:
```
label = df['loan_status'] # labels
features = df.drop('loan_status',axis=1) # features
x_train, x_test, y_train, y_test = model_selection.train_test_split

print('The train dataset has {} data\nThe test dataset has {} data'
      format(x_train.shape[0], x_test.shape[0]))
```

```
The train dataset has 22175 data
The test dataset has 9504 data
```

In [24]:
```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(R_tree, x_train, y_train, cv = 10, scoring
print('Cross-validation scores:{}'.format(scores))
```

```
Cross-validation scores:[0.9220018  0.93688007 0.93462579 0.932371
51 0.93146979 0.9300857
 0.932341   0.93279206 0.93053676 0.92061344]
```

In [25]:
```
scores = cross_val_score(knn, x_train, y_train, cv = 10, scoring='a
print('Cross-validation scores:{}'.format(scores))
```

```
Cross-validation scores:[0.83453562 0.8367899  0.83769161 0.840396
75 0.83047791 0.83942264
 0.83400992 0.8488949  0.83897158 0.8285972 ]
```

In [24]:
```
scores = cross_val_score(N_bayes, x_train, y_train, cv = 10, scorin
print('Cross-validation scores:{}'.format(scores))
```

```
Cross-validation scores:[0.81514878 0.81875564 0.82506763 0.816501
35 0.82100992 0.82318448
 0.82995038 0.81145692 0.81686964 0.81867388]
```

In [26]:
```
scores = cross_val_score(lg, x_train, y_train, cv = 10, scoring='ac
print('Cross-validation scores:{}'.format(scores))
```

```
Cross-validation scores:[0.80342651 0.8097385  0.80297565 0.812443
64 0.80613165 0.81235904
 0.81326116 0.8105548  0.8015336  0.80514208]
```

In [29]:
```
scores = cross_val_score(XGB_model, x_train, y_train, cv = 10, scor
print('Cross-validation scores:{}'.format(scores))
```

```
/opt/anaconda3/lib/python3.8/site-packages/xgboost/sklearn.py:1224
: UserWarning: The use of label encoder in XGBClassifier is deprec
ated and will be removed in a future release. To remove this warni
ng, do the following: 1) Pass option use_label_encoder=False when
constructing XGBClassifier object; and 2) Encode your labels (y) a
s integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

Cross-validation scores:[0.93327322 0.9418395  0.93958521 0.935527
5  0.94274121 0.9391069
 0.93594948 0.94091114 0.92918358 0.92737934]
```

In [21]:
```
def model_assess(model, name='Default'):
    model.fit(x_train, y_train)
    preds = model.predict(x_test)
    preds_proba = model.predict_proba(x_test)
    print(name, '\n',classification_report(y_test, model.predict(x_
    print(confusion_matrix(y_test,preds))
```

In [22]:
```
N_bayes=GaussianNB()
model_assess(N_bayes, name='Naive bayes')

lg = LogisticRegression(random_state=42)
model_assess(lg, 'Logistic Regression')

R_tree=RandomForestClassifier()
model_assess(R_tree,'RandomForest Classifier')

XGB_model = XGBClassifier(learning_rate=0.1, max_depth=10, scale_po
model_assess(XGB_model,'Xgboost')

knn = KNeighborsClassifier(n_neighbors=150)
model_assess(knn, name='KNN')
```

```
Naive bayes
              precision    recall  f1-score   support

           0       0.85      0.96      0.90      7456
           1       0.70      0.37      0.48      2048

    accuracy                           0.83      9504
   macro avg       0.77      0.66      0.69      9504
weighted avg       0.81      0.83      0.81      9504
```

```
[[7135  321]
 [1296  752]]
Logistic Regression
              precision    recall  f1-score   support

           0       0.81      0.98      0.89      7456
           1       0.71      0.15      0.25      2048

    accuracy                           0.80      9504
   macro avg       0.76      0.57      0.57      9504
weighted avg       0.79      0.80      0.75      9504

[[7327  129]
 [1738  310]]
RandomForest Classifier
              precision    recall  f1-score   support

           0       0.93      0.99      0.96      7456
           1       0.97      0.71      0.82      2048

    accuracy                           0.93      9504
   macro avg       0.95      0.85      0.89      9504
weighted avg       0.94      0.93      0.93      9504

[[7411   45]
 [ 584 1464]]

/opt/anaconda3/lib/python3.8/site-packages/xgboost/sklearn.py:1224
: UserWarning: The use of label encoder in XGBClassifier is deprec
ated and will be removed in a future release. To remove this warni
ng, do the following: 1) Pass option use_label_encoder=False when
constructing XGBClassifier object; and 2) Encode your labels (y) a
s integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

Xgboost
              precision    recall  f1-score   support

           0       0.93      0.99      0.96      7456
           1       0.95      0.75      0.84      2048

    accuracy                           0.94      9504
   macro avg       0.94      0.87      0.90      9504
weighted avg       0.94      0.94      0.93      9504

[[7373   83]
 [ 517 1531]]
KNN
              precision    recall  f1-score   support

           0       0.85      0.96      0.90      7456
           1       0.71      0.38      0.49      2048

    accuracy                           0.83      9504
```

```
   macro avg       0.78      0.67      0.70      9504
weighted avg       0.82      0.83      0.81      9504

[[7144  312]
 [1276  772]]
```

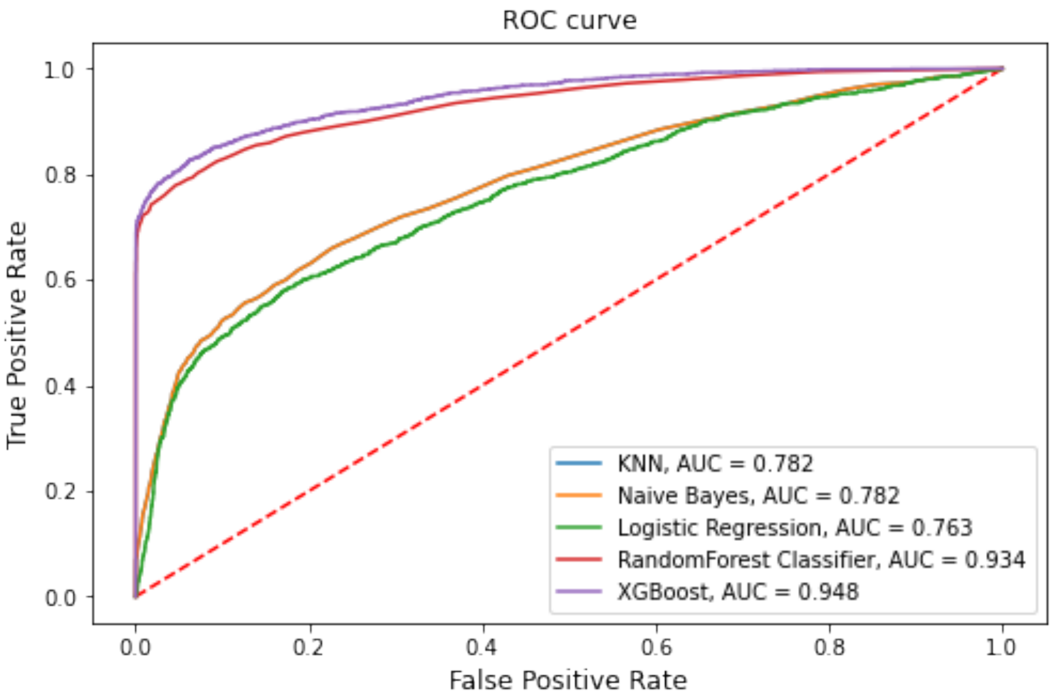
In [23]:
```
from tabulate import tabulate
```

In [25]:
```
head = ["Model", "Accuracy", "Precision", "Recall", "F1-score"]

final_metrics = [
    ["Naive Bayes", 0.83, 0.81, 0.83, 0.81],
    ["Logistic Regression", 0.80, 0.79, 0.80, 0.75],
    ["KNN Classifier", 0.83, 0.82, 0.83, 0.81],
    ["RandomForestClassifier", 0.93, 0.94, 0.93, 0.93],
    ["Xgboost", 0.94, 0.94, 0.94, 0.93]
]

print(tabulate(final_metrics, headers = head))
```

```
Model                     Accuracy    Precision    Recall    F1-sc
ore
----------------------  ----------  -----------  --------  -------
---
Naive Bayes                   0.83         0.81      0.83        0
.81
Logistic Regression           0.8          0.79      0.8         0
.75
KNN Classifier                0.83         0.82      0.83        0
.81
RandomForestClassifier        0.93         0.94      0.93        0
.93
Xgboost                       0.94         0.94      0.94        0
.93
```

In [31]:
```
### ROC AUC
fig = plt.figure(figsize=(8,5))
plt.plot([0, 1], [0, 1],'r--')

preds_proba_knn = knn.predict_proba(x_test)
probsknn = preds_proba_knn[:, 1]
fpr, tpr, thresh = metrics.roc_curve(y_test, probsknn)
aucknn = roc_auc_score(y_test, probsknn)
plt.plot(fpr, tpr, label=f'KNN, AUC = {str(round(aucknn,3))}')


preds_proba_N_bayes = N_bayes.predict_proba(x_test)
probsknn = preds_proba_knn[:, 1]
fpr, tpr, thresh = metrics.roc_curve(y_test, probsknn)
aucknn = roc_auc_score(y_test, probsknn)
plt.plot(fpr, tpr, label=f'Naive Bayes, AUC = {str(round(aucknn,3))

#Logistic Regression
```

```
#Logistic Regression
preds_proba_lg = lg.predict_proba(x_test)
probslg = preds_proba_lg[:, 1]
fpr, tpr, thresh = metrics.roc_curve(y_test, probslg)
auclg = roc_auc_score(y_test, probslg)
plt.plot(fpr, tpr, label=f'Logistic Regression, AUC = {str(round(au


#RandomForest Classifier
preds_proba_R_tree = R_tree.predict_proba(x_test)
probsR_tree = preds_proba_R_tree[:, 1]
fpr, tpr, thresh = metrics.roc_curve(y_test, probsR_tree)
auclg = roc_auc_score(y_test, probsR_tree)
plt.plot(fpr, tpr, label=f'RandomForest Classifier, AUC = {str(roun

#XGBoost
preds_proba_xgb = XGB_model.predict_proba(x_test)
probsxgb = preds_proba_xgb[:, 1]
fpr, tpr, thresh = metrics.roc_curve(y_test, probsxgb)
aucxgb = roc_auc_score(y_test, probsxgb)
plt.plot(fpr, tpr, label=f'XGBoost, AUC = {str(round(aucxgb,3))}')


plt.ylabel("True Positive Rate", fontsize=12)
plt.xlabel("False Positive Rate", fontsize=12)
plt.title("ROC curve")
plt.rcParams['axes.titlesize'] = 16
plt.legend()
plt.show()
```

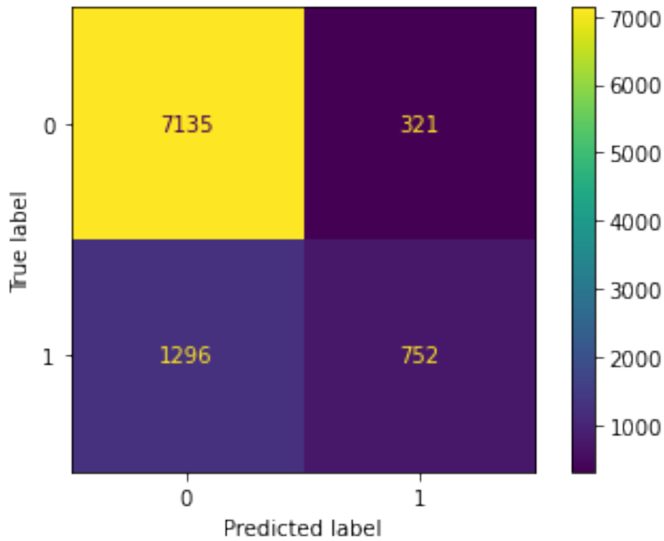


In [ ]:

In [44]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
```

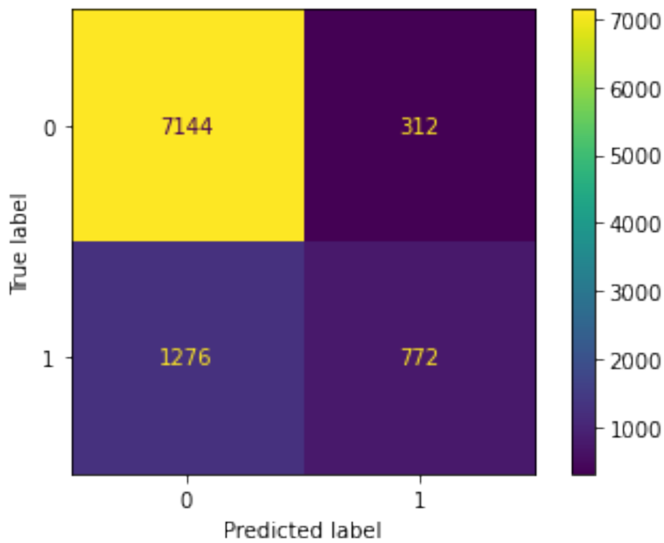
In [55]:
```
plot_confusion_matrix(N_bayes, x_test, y_test)
```

Out[55]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f89205950a0>



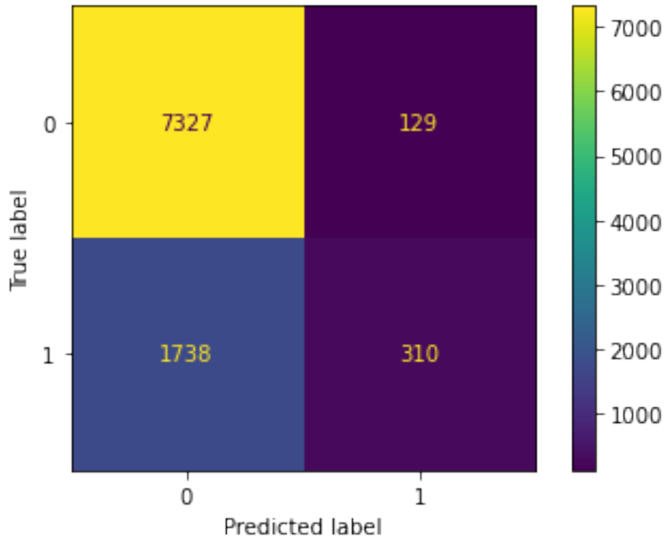
In [47]:
```
plot_confusion_matrix(knn, x_test, y_test)
```

Out[47]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f88f016c190>



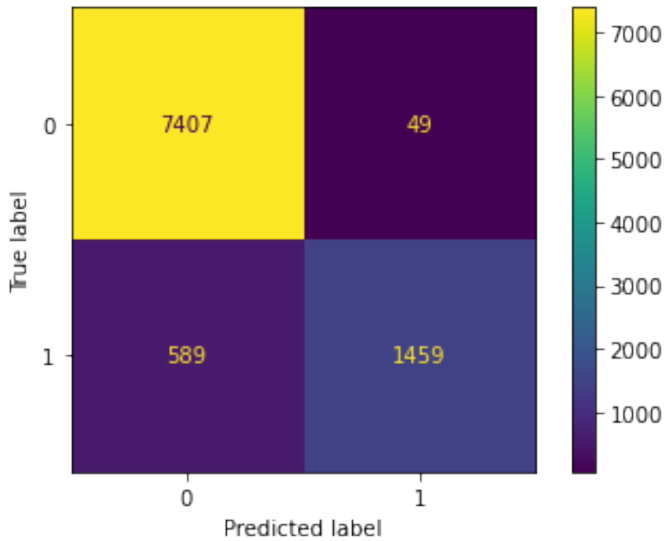
In [53]:
```
plot_confusion_matrix(lg, x_test, y_test)
```

Out[53]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f8930dd43d0>
```



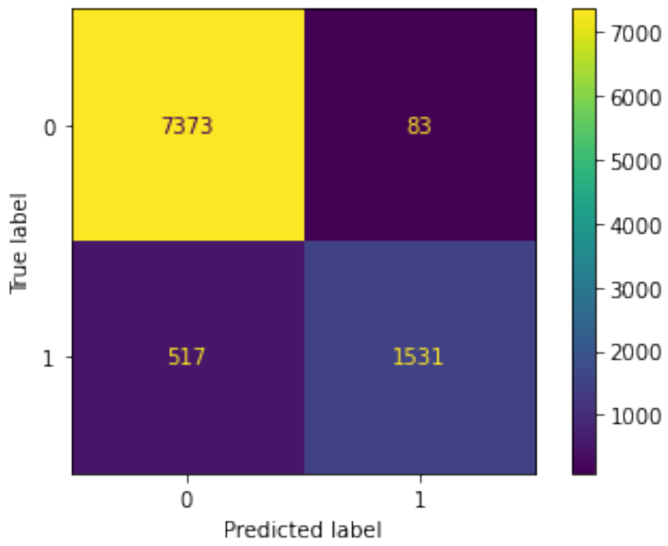
In [54]:
```
plot_confusion_matrix(R_tree, x_test, y_test)
```

Out[54]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f88e039a9a0>
```



In [52]:
```
plot_confusion_matrix(XGB_model, x_test, y_test)
```

Out[52]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f8931f3af40>
```



In [ ]:
```

```

In [ ]:
```
# KNeighbors Classifier
```

In [20]:
```
grid_params = { 'n_neighbors' : [5,7,9,11,13,15],
               'weights' : ['uniform','distance'],
               'metric' : ['minkowski','euclidean','manhattan']}
```

In [22]:
```
from sklearn.model_selection import GridSearchCV
gs = GridSearchCV(KNeighborsClassifier(), grid_params, verbose = 1,
```

In [24]:
```
gs.fit(x_train,y_train)
print(f"Best Score: {gs.best_score_}")
print("Standard Devaition:",gs.cv_results_['std_test_score'][gs.bes
print("Best parameters set:")
best_parameters = gs.best_estimator_.get_params()
for param_name in sorted(grid_params.keys()):
  print(f"\t{param_name}: {best_parameters[param_name]}")
```

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent
workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    2.5s

Best Score: 0.8443741381478156
Standard Devaition: 0.0026381041572729655
Best parameters set:
        metric: manhattan
        n_neighbors: 15
        weights: distance

[Parallel(n_jobs=-1)]: Done 108 out of 108 | elapsed:    4.1s fini
shed
```

In [25]:
```
g_res = gs.fit(x_train, y_train)
```

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent
workers.
[Parallel(n_jobs=-1)]: Done  52 tasks      | elapsed:    1.4s
[Parallel(n_jobs=-1)]: Done  93 out of 108 | elapsed:    2.4s rema
ining:    0.4s
[Parallel(n_jobs=-1)]: Done 108 out of 108 | elapsed:    2.6s fini
shed
```

In [26]:
```
g_res.best_score_
```

Out[26]:
```
0.8443741381478156
```

In [27]:
```
g_res.best_params_
```

Out[27]:
```
{'metric': 'manhattan', 'n_neighbors': 15, 'weights': 'distance'}
```

In [29]:
```
gs.cv_results_
```

```
5320168, 0.2270871 ,
        0.0407571 , 0.24188296, 0.05830042, 0.25471973, 0.06716299
,
        0.24916704, 0.07165273, 0.24579279, 0.0480605 , 0.23538852
,
        0.06678303, 0.302713  , 0.07904553, 0.32016063, 0.06641014
,
        0.28328514, 0.06828213, 0.26133839, 0.0660344 , 0.21840445
,
        0.05286956, 0.23570021, 0.05055467, 0.23927943, 0.05580799
,
        0.22160697, 0.07106972, 0.24949837, 0.06862577, 0.18994912
,
        0.06747842]),
 'std_score_time': array([0.00822901, 0.00514166, 0.01411297, 0.00
735241, 0.02825148,
        0.00309762, 0.03853331, 0.00211289, 0.03971449, 0.00536478
,
        0.01180724, 0.01513796, 0.02797528, 0.00166903, 0.04598741
```

In [30]:
```
result=pd.DataFrame(gs.cv_results_)
print(result)
```

```
    mean_fit_time  std_fit_time  mean_score_time  std_score_time p
aram_metric  \
0        0.019805      0.000917         0.240762        0.008229
minkowski
1        0.022732      0.002807         0.046491        0.005142
minkowski
2        0.018147      0.005314         0.225082        0.014113
minkowski
3        0.015069      0.000466         0.053202        0.007352
minkowski
4        0.014763      0.002351         0.227087        0.028251
minkowski
5        0.013540      0.002371         0.040757        0.003098
minkowski
6        0.017636      0.003851         0.241883        0.038533
minkowski
7        0.017406      0.003491         0.058300        0.002113
minkowski
8        0.015601      0.000580         0.254720        0.039714
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
# Logistic Regression
```

In [31]:
```
LR_param_grid = {
    'C': np.logspace(0, 4, num=10),
    'penalty': ['l2'],
    'solver': ['liblinear','saga','newton-cg','lbfgs']
}
```

In [32]:
```
logistic = LogisticRegression()

Logistic_model = model_selection.RandomizedSearchCV(
    estimator = logistic,
    param_distributions = LR_param_grid,
    n_iter = 20,
    scoring = "accuracy",
    verbose = 5,
    n_jobs = 1,
    cv = 5
)
```

In [33]:
```
Logistic_model.fit(x_train,y_train)
print(f"Best Score: {Logistic_model.best_score_}")
print("Standard Devaition:",Logistic_model.cv_results_['std_test_sc
print("Best parameters set:")
best_parameters = Logistic_model.best_estimator_.get_params()
for param_name in sorted(LR_param_grid.keys()):
print(f"\t{param_name}: {best_parameters[param_name]}")
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
[CV] solver=newton-cg, penalty=l2, C=21.544346900318832 .........
....

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concu
rrent workers.
/opt/anaconda3/lib/python3.8/site-packages/scipy/optimize/linesear
ch.py:477: LineSearchWarning: The line search algorithm did not co
nverge
  warn('The line search algorithm did not converge', LineSearchWar
ning)
/opt/anaconda3/lib/python3.8/site-packages/scipy/optimize/linesear
ch.py:327: LineSearchWarning: The line search algorithm did not co
nverge
  warn('The line search algorithm did not converge', LineSearchWar
ning)
/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/optimize.
py:211: ConvergenceWarning: newton-cg failed to converge. Increase
the number of iterations.
  warnings.warn("newton-cg failed to converge. Increase the "
```

In [34]:
```
Logistic_model.cv_results_
```

Out[34]:
```
{'mean_fit_time': array([0.60576491, 0.01789865, 0.20223923, 0.200
```

```
76528, 0.64459615,
        0.01796274, 0.63561707, 0.20858283, 0.20496206, 0.04289246
,
        0.04218559, 0.60432382, 0.65315065, 0.04150758, 0.01874418
,
        0.04258528, 0.01881323, 0.02018046, 0.20426989, 0.20889564
]),
 'std_fit_time': array([0.04925154, 0.00227371, 0.0021942 , 0.0020
789 , 0.01315948,
        0.00142151, 0.03296847, 0.00603425, 0.00143005, 0.00682262
,
        0.00641753, 0.00899923, 0.0311405 , 0.00569562, 0.00133522
,
        0.00694027, 0.00182796, 0.00201667, 0.00283887, 0.00290232
]),
 'mean_score_time': array([0.00159802, 0.00116835, 0.00187278, 0.0
0176234, 0.00157585,
        0.00117164, 0.00139194, 0.00192318, 0.00171061, 0.00137129
,
        0.00137019, 0.00149293, 0.00152178, 0.00137038, 0.00121737
,
        0.00137825, 0.00117159, 0.00139918, 0.00151815, 0.00189476
]),
 'std_score_time': array([1.09841134e-04, 2.02622691e-05, 2.515765
46e-04, 2.29276446e-04,
        2.49128802e-04, 9.44209096e-05, 7.00354160e-05, 2.87488978
e-04,
        2.51206414e-04, 8.13724532e-05, 8.94229239e-05, 1.02717343
e-04,
        2.56664545e-04, 6.84694658e-05, 2.94949897e-05, 1.35802218
e-04,
        7.68996573e-05, 2.30319534e-04, 1.24908911e-04, 2.25424604
e-04]),
 'param_solver': masked_array(data=['newton-cg', 'lbfgs', 'saga',
'saga', 'newton-cg',
                    'lbfgs', 'newton-cg', 'saga', 'saga', 'libline
ar',
                    'liblinear', 'newton-cg', 'newton-cg', 'liblin
ear',
                    'lbfgs', 'liblinear', 'lbfgs', 'lbfgs', 'saga'
, 'saga'],
              mask=[False, False, False, False, False, False, Fals
e, False,
                    False, False, False, False, False, False, Fals
e, False,
                    False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_penalty': masked_array(data=['l2', 'l2', 'l2', 'l2', 'l2',
'l2', 'l2', 'l2', 'l2',
                    'l2', 'l2', 'l2', 'l2', 'l2', 'l2', 'l2', 'l2'
, 'l2',
                    'l2', 'l2'],
```

```
              mask=[False, False, False, False, False, False, Fals
e, False,
                    False, False, False, False, False, False, Fals
e, False,
                    False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_C': masked_array(data=[21.544346900318832, 3593.8136638046
26,
                    21.544346900318832, 1291.5496650148827,
                    1291.5496650148827, 21.544346900318832,
                    464.15888336127773, 1.0, 166.81005372000593,
                    464.15888336127773, 1291.5496650148827,
                    59.94842503189409, 3593.813663804626,
                    21.544346900318832, 166.81005372000593,
                    2.7825594022071245, 464.15888336127773, 1.0,
                    7.742636826811269, 2.7825594022071245],
              mask=[False, False, False, False, False, False, Fals
e, False,
                    False, False, False, False, False, False, Fals
e, False,
                    False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'solver': 'newton-cg', 'penalty': 'l2', 'C': 21.54434
6900318832},
  {'solver': 'lbfgs', 'penalty': 'l2', 'C': 3593.813663804626},
  {'solver': 'saga', 'penalty': 'l2', 'C': 21.544346900318832},
  {'solver': 'saga', 'penalty': 'l2', 'C': 1291.5496650148827},
  {'solver': 'newton-cg', 'penalty': 'l2', 'C': 1291.5496650148827
},
  {'solver': 'lbfgs', 'penalty': 'l2', 'C': 21.544346900318832},
  {'solver': 'newton-cg', 'penalty': 'l2', 'C': 464.15888336127773
},
  {'solver': 'saga', 'penalty': 'l2', 'C': 1.0},
  {'solver': 'saga', 'penalty': 'l2', 'C': 166.81005372000593},
  {'solver': 'liblinear', 'penalty': 'l2', 'C': 464.15888336127773
},
  {'solver': 'liblinear', 'penalty': 'l2', 'C': 1291.5496650148827
},
  {'solver': 'newton-cg', 'penalty': 'l2', 'C': 59.94842503189409}
,
  {'solver': 'newton-cg', 'penalty': 'l2', 'C': 3593.813663804626}
,
  {'solver': 'liblinear', 'penalty': 'l2', 'C': 21.544346900318832
},
  {'solver': 'lbfgs', 'penalty': 'l2', 'C': 166.81005372000593},
  {'solver': 'liblinear', 'penalty': 'l2', 'C': 2.7825594022071245
},
  {'solver': 'lbfgs', 'penalty': 'l2', 'C': 464.15888336127773},
  {'solver': 'lbfgs', 'penalty': 'l2', 'C': 1.0},
  {'solver': 'saga', 'penalty': 'l2', 'C': 7.742636826811269},
  {'solver': 'saga', 'penalty': 'l2', 'C': 2.7825594022071245}],
```

```
 'split0_test_score': array([0.8410372 , 0.8065389 , 0.80631342, 0
.80631342, 0.84126268,
        0.8065389 , 0.84193912, 0.80631342, 0.80631342, 0.82322435
,
        0.82390079, 0.84239008, 0.84193912, 0.82322435, 0.8065389
,
        0.82322435, 0.8065389 , 0.8065389 , 0.80631342, 0.80631342
]),
 'split1_test_score': array([0.8475761 , 0.80766629, 0.80586246, 0
.80586246, 0.84735062,
        0.80766629, 0.84712514, 0.80586246, 0.80586246, 0.82998873
,
        0.82998873, 0.84870349, 0.84712514, 0.82998873, 0.80766629
,
        0.82998873, 0.80766629, 0.80766629, 0.80586246, 0.80586246
]),
 'split2_test_score': array([0.84870349, 0.80947012, 0.81014656, 0
.81014656, 0.84847802,
        0.80947012, 0.84847802, 0.81014656, 0.81014656, 0.82908681
,
        0.81894025, 0.84847802, 0.84870349, 0.81894025, 0.80947012
,
        0.82931229, 0.80947012, 0.80947012, 0.81014656, 0.81014656
]),
 'split3_test_score': array([0.8518602 , 0.81172492, 0.81059752, 0
.81059752, 0.8518602 ,
        0.81172492, 0.8518602 , 0.81059752, 0.81059752, 0.80473506
,
        0.80473506, 0.85163472, 0.8518602 , 0.80473506, 0.81172492
,
        0.80473506, 0.81172492, 0.81172492, 0.81059752, 0.81059752
]),
 'split4_test_score': array([0.83878241, 0.80360767, 0.80180383, 0
.80180383, 0.83720406,
        0.80360767, 0.8378805 , 0.80180383, 0.80180383, 0.81894025
,
        0.81894025, 0.83742954, 0.8378805 , 0.81894025, 0.80360767
,
        0.81894025, 0.80360767, 0.80360767, 0.80180383, 0.80180383
]),
 'mean_test_score': array([0.84559188, 0.80780158, 0.80694476, 0.8
0694476, 0.84523112,
        0.80780158, 0.8454566 , 0.80694476, 0.80694476, 0.82119504
,
        0.81930101, 0.84572717, 0.84550169, 0.81916573, 0.80780158
,
        0.82124014, 0.80780158, 0.80780158, 0.80694476, 0.80694476
]),
 'std_test_score': array([0.00489949, 0.00273416, 0.00321227, 0.00
321227, 0.00527563,
        0.00273416, 0.00495357, 0.00321227, 0.00321227, 0.00916234
,
        0.00834041, 0.00512468, 0.00498181, 0.00826989, 0.00273416
```

```
,
        0.00920154, 0.00273416, 0.00273416, 0.00321227, 0.00321227
]),
 'rank_test_score': array([ 2, 10, 15, 15,  5, 10,  4, 15, 15,  7,
8,  1,  3,  9, 10,  6, 10,
        10, 15, 15], dtype=int32)}
```

In [35]:
```
result=pd.DataFrame(Logistic_model.cv_results_)
print(result)
```

```
    mean_fit_time  std_fit_time  mean_score_time  std_score_time p
aram_solver  \
0        0.605765      0.049252         0.001598        0.000110
newton-cg
1        0.017899      0.002274         0.001168        0.000020
lbfgs
2        0.202239      0.002194         0.001873        0.000252
saga
3        0.200765      0.002079         0.001762        0.000229
saga
4        0.644596      0.013159         0.001576        0.000249
newton-cg
5        0.017963      0.001422         0.001172        0.000094
lbfgs
6        0.635617      0.032968         0.001392        0.000070
newton-cg
7        0.208583      0.006034         0.001923        0.000287
saga
8        0.204962      0.001430         0.001711        0.000251
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
# Naive Bayes
```

In [33]:
```
from sklearn.model_selection import RandomizedSearchCV
classifier2 = RandomizedSearchCV(R_tree,params,cv=5)
```

In [31]:
```
params = {
'var_smoothing': np.logspace(0,-9, num=100)
}
```

In [34]:
```
classifier4 = RandomizedSearchCV(N_bayes,params,cv=5)
```

In [35]:
```
classifier4.fit(x_train,y_train)
```

Out[35]:
```
RandomizedSearchCV(cv=5, estimator=GaussianNB(),
                   param_distributions={'var_smoothing': array([1.
00000000e+00, 8.11130831e-01, 6.57933225e-01, 5.33669923e-01,
       4.32876128e-01, 3.51119173e-01, 2.84803587e-01, 2.31012970e
-01,
       1.87381742e-01, 1.51991108e-01, 1.23284674e-01, 1.00000000e
-01,
       8.11130831e-02, 6.57933225e-02, 5.33669923e-02, 4.32876128e
-02,
       3.51119173e-02, 2.84...
       1.23284674e-07, 1.00000000e-07, 8.11130831e-08, 6.57933225e
-08,
       5.33669923e-08, 4.32876128e-08, 3.51119173e-08, 2.84803587e
-08,
       2.31012970e-08, 1.87381742e-08, 1.51991108e-08, 1.23284674e
-08,
       1.00000000e-08, 8.11130831e-09, 6.57933225e-09, 5.33669923e
-09,
       4.32876128e-09, 3.51119173e-09, 2.84803587e-09, 2.31012970e
-09,
       1.87381742e-09, 1.51991108e-09, 1.23284674e-09, 1.00000000e
-09])})
```

In [36]:
```
classifier4.cv_results_
```

Out[36]:
```
{'mean_fit_time': array([0.00831461, 0.00435195, 0.00388036, 0.003
46899, 0.0034637 ,
        0.00345373, 0.00345793, 0.00346546, 0.00343475, 0.00344124
]),
 'std_fit_time': array([3.59709244e-03, 3.73388572e-04, 7.00713065
e-04, 1.57052527e-05,
        2.10842920e-05, 1.00487138e-05, 1.91464222e-05, 3.87259299
e-05,
        9.38121819e-06, 1.49772418e-05]),
 'mean_score_time': array([0.00240235, 0.00149393, 0.00117865, 0.0
0113735, 0.00112734,
        0.00112901, 0.00114446, 0.00113096, 0.00111995, 0.00112214
]),
 'std_score_time': array([4.61878940e-04, 2.06275147e-04, 7.022349
60e-05, 7.44629737e-06,
        5.05222414e-06, 4.86186870e-06, 2.53878958e-05, 1.16553282
e-05,
        5.67135611e-06, 4.37704305e-06]),
 'param_var_smoothing': masked_array(data=[5.3366992312063123e-05,
0.12328467394420659,
                    0.002848035868435802, 3.511191734215127e-09,
                    6.579332246575682e-07, 2.848035868435799e-06,
                    8.111308307896872e-07, 1e-05, 0.00035111917342
15131,
                    1.5199110829529333e-06],
              mask=[False, False, False, False, False, False, Fals
```

```
e, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'params': [{'var_smoothing': 5.3366992312063123e-05},
  {'var_smoothing': 0.12328467394420659},
  {'var_smoothing': 0.0028480358684358O2},
  {'var_smoothing': 3.5111917342151273e-09},
  {'var_smoothing': 6.579332246575682e-07},
  {'var_smoothing': 2.848035868435799e-06},
  {'var_smoothing': 8.111308307896872e-07},
  {'var_smoothing': 1e-05},
  {'var_smoothing': 0.0003511191734215131},
  {'var_smoothing': 1.519911082952933e-06}],
 'split0_test_score': array([0.78624577, 0.78444194, 0.78692221, 0
.80202931, 0.78624577,
        0.78624577, 0.78624577, 0.78624577, 0.78737317, 0.78624577
]),
 'split1_test_score': array([0.78759865, 0.78444194, 0.78782413, 0
.80856821, 0.78782413,
        0.78759865, 0.78782413, 0.78759865, 0.78759865, 0.78782413
]),
 'split2_test_score': array([0.78872604, 0.78466742, 0.78692221, 0
.80225479, 0.789177  ,
        0.789177  , 0.789177  , 0.789177  , 0.78850056, 0.789177
]),
 'split3_test_score': array([0.78308906, 0.78466742, 0.78827508, 0
.80225479, 0.78354002,
        0.78286359, 0.78354002, 0.78286359, 0.78466742, 0.78308906
]),
 'split4_test_score': array([0.78714769, 0.78466742, 0.78624577, 0
.80315671, 0.78737317,
        0.78737317, 0.78737317, 0.78737317, 0.78759865, 0.78737317
]),
 'mean_test_score': array([0.78656144, 0.78457723, 0.78723788, 0.8
0365276, 0.78683202,
        0.78665163, 0.78683202, 0.78665163, 0.78714769, 0.78674183
]),
 'std_test_score': array([0.00191113, 0.00011046, 0.00072153, 0.00
248805, 0.0018951 ,
        0.0021123 , 0.0018951 , 0.0021123 , 0.0012992 , 0.00205372
]),
 'rank_test_score': array([ 9, 10,  2,  1,  4,  7,  4,  7,  3,  6]
, dtype=int32)}
```

In [37]:

```
best_parameters=classifier4.best_params_
print(best_parameters)
```

```
{'var_smoothing': 3.5111917342151273e-09}
```

In [38]:
```
highest_accuracy=classifier4.best_score_
print(highest_accuracy)
```

```
0.803652762119504
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: