In [1]:
```
# This Python 3 environment comes with many helpful analytics libraries install
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/dc
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns# data processing, CSV file I/O (e.g. pd.read_csv)


# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list

import os


# You can write up to 20GB to the current directory (/kaggle/working/) that get
# You can also write temporary files to /kaggle/temp/, but they won't be saved
```

In [2]:
```
df = pd.read_csv('WA_Fn-UseC_-Telco-Customer-Churn.csv')
```

Undertstanding the Data

In [3]:
```
df.head()
```

Out[3]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service
1	5575-GNVDE	Male	0	No	No	34	Yes	No
2	3668-QPYBK	Male	0	No	No	2	Yes	No
3	7795-CFOCW	Male	0	No	No	45	No	No phone service
4	9237-HQITU	Female	0	No	No	2	Yes	No

5 rows × 21 columns

In [4]:
```
df.columns
```

Out[4]:
```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

In [5]:
```
df.drop("customerID", axis = 1, inplace = True)
```

In [6]:
```
df.tenure.mean()
```

Out[6]:
```
32.37114865824223
```

In [7]:
```
df.Contract.value_counts()
```

Out[7]:
```
Month-to-month    3875
Two year          1695
One year          1473
Name: Contract, dtype: int64
```

In [8]:
```
churn_rate = df.Churn[df.Churn == "Yes"].count() / df.shape[0]
print(churn_rate)
```

```
0.2653698707936959
```

In [9]:
```
churned = df.loc[df.Churn == "Yes"]
```

In [10]:
```
churned["Contract"].value_counts() #long contracts leads to lesser churn rate
```

Out[10]:
```
Month-to-month    1655
One year           166
Two year            48
Name: Contract, dtype: int64
```

In [11]:
```
churned["tenure"].value_counts() #similarly over here too
```

Out[11]:
```
1     380
2     123
3      94
4      83
5      64
     ...
60      6
72      6
62      5
64      4
63      4
Name: tenure, Length: 72, dtype: int64
```

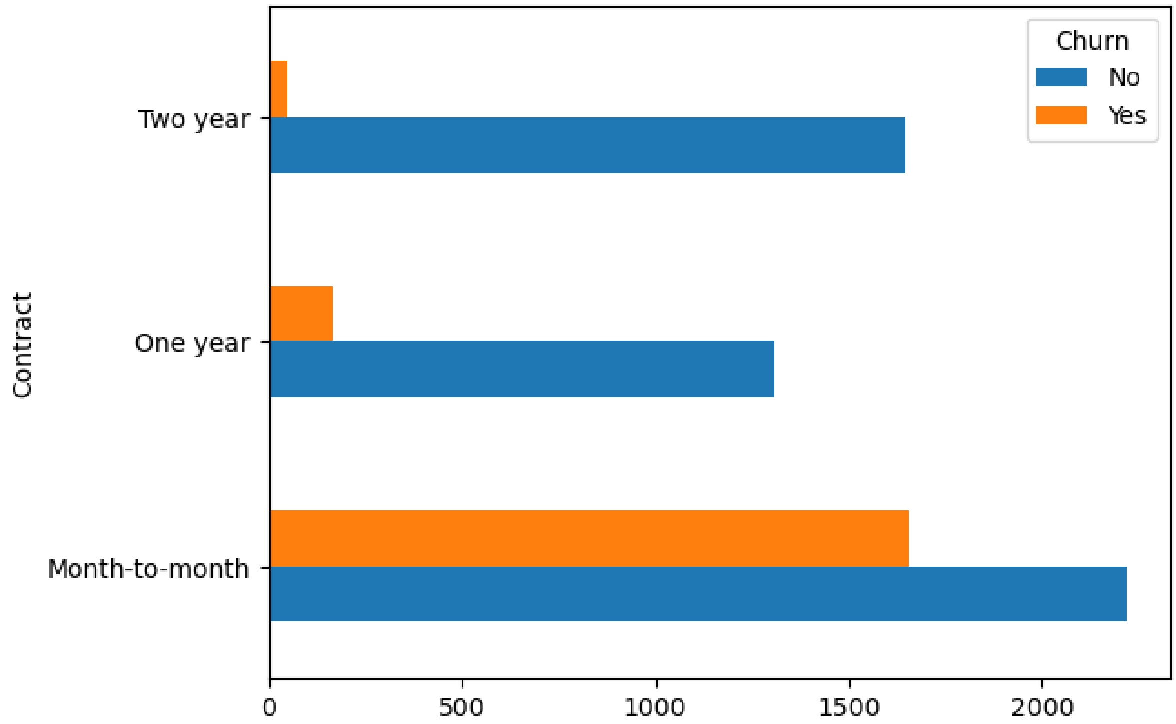
In [12]:
```
df.Churn.value_counts() #1 represents Churned
```

Out[12]:
```
No     5174
Yes    1869
Name: Churn, dtype: int64
```

In [13]:
```
CTR = pd.crosstab(index = df.Contract, columns = df.Churn)
```

In [14]:
```
CTR.plot.barh()
```

Out[14]: <Axes: ylabel='Contract'>



In [15]:
```
CTR
```

Out[15]:

Churn	No	Yes
Contract		
Month-to-month	2220	1655
One year	1307	166
Two year	1647	48

In [16]:
```
df.isnull().sum()
```

Out[16]:
```
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

In [17]:
```
df.info() #Total charges is object data type, must be some white space since no
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   gender            7043 non-null   object 
 1   SeniorCitizen     7043 non-null   int64  
 2   Partner           7043 non-null   object 
 3   Dependents        7043 non-null   object 
 4   tenure            7043 non-null   int64  
 5   PhoneService      7043 non-null   object 
 6   MultipleLines     7043 non-null   object 
 7   InternetService   7043 non-null   object 
 8   OnlineSecurity    7043 non-null   object 
 9   OnlineBackup      7043 non-null   object 
 10  DeviceProtection  7043 non-null   object 
 11  TechSupport       7043 non-null   object 
 12  StreamingTV       7043 non-null   object 
 13  StreamingMovies   7043 non-null   object 
 14  Contract          7043 non-null   object 
 15  PaperlessBilling  7043 non-null   object 
 16  PaymentMethod     7043 non-null   object 
 17  MonthlyCharges    7043 non-null   float64
 18  TotalCharges      7043 non-null   object 
 19  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(17)
memory usage: 1.1+ MB
```

In [18]:
```
df.TotalCharges = pd.to_numeric(df.TotalCharges, errors = "coerce")
```

In [19]:
```
df.TotalCharges #dtype is float now
```

Out[19]:
```
0         29.85
1       1889.50
2        108.15
3       1840.75
4        151.65
         ...
7038    1990.50
7039    7362.90
7040     346.45
7041     306.60
7042    6844.50
Name: TotalCharges, Length: 7043, dtype: float64
```

In [20]:
```
df.isnull().sum() #now we have 11 missing values from the same column
```

Out[20]:
```
gender               0
SeniorCitizen        0
Partner              0
Dependents           0
tenure               0
PhoneService         0
MultipleLines        0
InternetService      0
OnlineSecurity       0
OnlineBackup         0
DeviceProtection     0
TechSupport          0
StreamingTV          0
StreamingMovies      0
Contract             0
PaperlessBilling     0
PaymentMethod        0
MonthlyCharges       0
TotalCharges        11
Churn                0
dtype: int64
```

In [21]:
```
mask = df.TotalCharges.isna()
missing = df[mask] #subsetting the missing rows
```

In [22]:
```
missing
```

Out[22]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetSe
488	Female	0	Yes	Yes	0	No	No phone service	
753	Male	0	No	Yes	0	Yes	No	
936	Female	0	Yes	Yes	0	Yes	No	
1082	Male	0	Yes	Yes	0	Yes	Yes	
1340	Female	0	Yes	Yes	0	No	No phone service	
3331	Male	0	Yes	Yes	0	Yes	No	
3826	Male	0	Yes	Yes	0	Yes	Yes	
4380	Female	0	Yes	Yes	0	Yes	No	
5218	Male	0	Yes	Yes	0	Yes	No	
6670	Female	0	Yes	Yes	0	Yes	Yes	
6754	Male	0	No	Yes	0	Yes	Yes	

In [23]:
```
df.iloc[753,19] #inspecting the missing values individually, white space it is
```

Out[23]: 'No'

In [24]:
```
"""instead of using MICE or replacing it with median, logical replacement will
The tenure column is assumed to be number of months. Since the tenure is 0 for
values, thus 0 seems to be the most likely replacement"""
```

Out[24]:
```
'instead of using MICE or replacing it with median, logical replacement will
be better.\nThe tenure column is assumed to be number of months. Since the te
nure is 0 for all missing\nvalues, thus 0 seems to be the most likely replace
ment'
```

In [25]:
```
df.TotalCharges.fillna(0, inplace=True)
```

In [26]:
```
df.TotalCharges.dtype
```

Out[26]: dtype('float64')

In [27]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   gender            7043 non-null   object 
 1   SeniorCitizen     7043 non-null   int64  
 2   Partner           7043 non-null   object 
 3   Dependents        7043 non-null   object 
 4   tenure            7043 non-null   int64  
 5   PhoneService      7043 non-null   object 
 6   MultipleLines     7043 non-null   object 
 7   InternetService   7043 non-null   object 
 8   OnlineSecurity    7043 non-null   object 
 9   OnlineBackup      7043 non-null   object 
 10  DeviceProtection  7043 non-null   object 
 11  TechSupport       7043 non-null   object 
 12  StreamingTV       7043 non-null   object 
 13  StreamingMovies   7043 non-null   object 
 14  Contract          7043 non-null   object 
 15  PaperlessBilling  7043 non-null   object 
 16  PaymentMethod     7043 non-null   object 
 17  MonthlyCharges    7043 non-null   float64
 18  TotalCharges      7043 non-null   float64
 19  Churn             7043 non-null   object 
dtypes: float64(2), int64(2), object(16)
memory usage: 1.1+ MB
```

In [28]:
```
df.PaymentMethod.unique()
```

Out[28]:
```
array(['Electronic check', 'Mailed check', 'Bank transfer (automatic)',
       'Credit card (automatic)'], dtype=object)
```

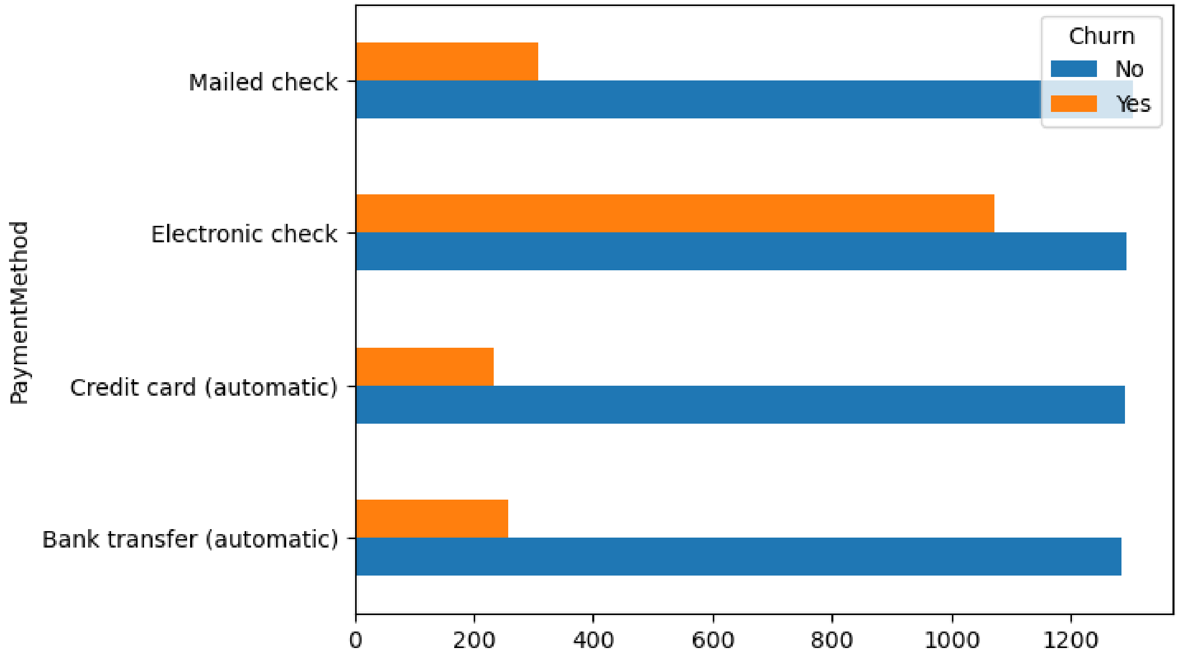
In [29]:
```
df.PaymentMethod.value_counts()
```

Out[29]:
```
Electronic check             2365
Mailed check                 1612
Bank transfer (automatic)    1544
Credit card (automatic)      1522
Name: PaymentMethod, dtype: int64
```

In [30]:
```
CPR = pd.crosstab(index = df.PaymentMethod, columns= df.Churn)
```

In [31]:
```
CPR.plot.barh()
```

Out[31]:
```
<Axes: ylabel='PaymentMethod'>
```



In [32]:
```
df.StreamingTV.value_counts()
```

Out[32]:
```
No                     2810
Yes                    2707
No internet service    1526
Name: StreamingTV, dtype: int64
```

In [33]:
```
numeric_cols = df.select_dtypes(["float64", "int64"])
```

In [34]:
```
numeric_cols
```

Out[34]:

	SeniorCitizen	tenure	MonthlyCharges	TotalCharges
0	0	1	29.85	29.85
1	0	34	56.95	1889.50
2	0	2	53.85	108.15
3	0	45	42.30	1840.75
4	0	2	70.70	151.65
...	...	...	...	...
7038	0	24	84.80	1990.50
7039	0	72	103.20	7362.90
7040	0	11	29.60	346.45
7041	1	4	74.40	306.60
7042	0	66	105.65	6844.50

7043 rows × 4 columns

Preprocessing and Model Selection

In [35]:
```
from sklearn import preprocessing
from sklearn.impute import SimpleImputer
from sklearn.linear_model import SGDClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score
from sklearn.metrics import precision_recall_curve
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn import svm
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.compose import ColumnTransformer
from sklearn.metrics import PrecisionRecallDisplay
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.model_selection import RandomizedSearchCV
```

In [36]:
```
unique_counts = df.select_dtypes("O").nunique()
binary_columns = unique_counts[unique_counts == 2].index.drop("Churn").tolist(
categorical_columns = unique_counts[unique_counts > 2].index.tolist()
target_column = "Churn"
```

In [37]:
```
X = df.drop(target_column, axis=1)
y = df[target_column]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, stratify=y, random_state=42
)
X_train.head(5)
```

Out[37]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetSe
5557	Female	0	No	No	5	Yes	No	Fibe
2270	Female	1	No	No	3	Yes	No	Fibe
6930	Female	0	Yes	No	3	Yes	Yes	Fibe
2257	Female	0	No	No	60	Yes	Yes	
898	Female	0	No	No	12	Yes	No	Fibe

In [38]:
```
transformer = ColumnTransformer(
    [
        ("scaler", StandardScaler(), ["MonthlyCharges", "TotalCharges", "tenure
        ("binary_encoder", preprocessing.OrdinalEncoder(), binary_columns),
        ("ohe", preprocessing.OneHotEncoder(drop="first"), categorical_columns)
    ],
    remainder="passthrough",
)

transformer.fit(X_train)
columns = transformer.get_feature_names_out()
columns = list(map(lambda x: str(x).split("__")[-1], columns))

X_train = pd.DataFrame(transformer.transform(X_train), columns=columns)
X_test = pd.DataFrame(transformer.transform(X_test), columns=columns)
```

In [39]:
```
label_encoder = preprocessing.LabelEncoder()
label_encoder.fit(y_train)
y_train = label_encoder.transform(y_train)
y_test = label_encoder.transform(y_test)
```

Support Vector Classifier

In [40]:
```
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}
```

In [41]:
```
SVC_grid = GridSearchCV(svm.SVC(), param_grid, refit = True, verbose = 6, cv =
SVC_grid.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 25 candidates, totalling 75 fits
[CV 1/3] END ........C=0.1, gamma=1, kernel=rbf;, score=0.754 total time=  1
0.9s
[CV 2/3] END ........C=0.1, gamma=1, kernel=rbf;, score=0.753 total time=  1
0.8s
[CV 3/3] END ........C=0.1, gamma=1, kernel=rbf;, score=0.744 total time=  1
0.6s
[CV 1/3] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.806 total time=
6.2s
[CV 2/3] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.796 total time=
5.6s
[CV 3/3] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.780 total time=
6.2s
[CV 1/3] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.735 total time=
6.1s
[CV 2/3] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.738 total time=
6.7s
[CV 3/3] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.768 total time=
6.5s
[CV 1/3] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.735 total time=
4.9s
[CV 2/3] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.735 total time=
5.5s
[CV 3/3] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.735 total time=
5.5s
[CV 1/3] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.735 total time=
5.5s
[CV 2/3] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.735 total time=
4.8s
[CV 3/3] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.735 total time=
5.1s
[CV 1/3] END ..........C=1, gamma=1, kernel=rbf;, score=0.790 total time=
9.3s
[CV 2/3] END ..........C=1, gamma=1, kernel=rbf;, score=0.787 total time=  1
1.7s
[CV 3/3] END ..........C=1, gamma=1, kernel=rbf;, score=0.772 total time=
9.3s
[CV 1/3] END ........C=1, gamma=0.1, kernel=rbf;, score=0.819 total time=
5.6s
[CV 2/3] END ........C=1, gamma=0.1, kernel=rbf;, score=0.804 total time=
5.4s
[CV 3/3] END ........C=1, gamma=0.1, kernel=rbf;, score=0.786 total time=
5.7s
[CV 1/3] END .......C=1, gamma=0.01, kernel=rbf;, score=0.814 total time=
6.5s
[CV 2/3] END .......C=1, gamma=0.01, kernel=rbf;, score=0.802 total time=
6.0s
[CV 3/3] END .......C=1, gamma=0.01, kernel=rbf;, score=0.783 total time=
6.5s
[CV 1/3] END ......C=1, gamma=0.001, kernel=rbf;, score=0.735 total time=
6.4s
[CV 2/3] END ......C=1, gamma=0.001, kernel=rbf;, score=0.739 total time=
5.6s
[CV 3/3] END ......C=1, gamma=0.001, kernel=rbf;, score=0.772 total time=
5.8s
[CV 1/3] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.735 total time=
5.7s
```

```
[CV 2/3] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.735 total time=
5.6s
[CV 3/3] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.735 total time=
5.8s
[CV 1/3] END .........C=10, gamma=1, kernel=rbf;, score=0.768 total time=
7.4s
[CV 2/3] END .........C=10, gamma=1, kernel=rbf;, score=0.771 total time=
9.0s
[CV 3/3] END .........C=10, gamma=1, kernel=rbf;, score=0.761 total time=  1
0.0s
[CV 1/3] END .......C=10, gamma=0.1, kernel=rbf;, score=0.804 total time=
7.4s
[CV 2/3] END .......C=10, gamma=0.1, kernel=rbf;, score=0.796 total time=
6.0s
[CV 3/3] END .......C=10, gamma=0.1, kernel=rbf;, score=0.780 total time=
6.7s
[CV 1/3] END ......C=10, gamma=0.01, kernel=rbf;, score=0.815 total time=
5.5s
[CV 2/3] END ......C=10, gamma=0.01, kernel=rbf;, score=0.795 total time=
5.0s
[CV 3/3] END ......C=10, gamma=0.01, kernel=rbf;, score=0.781 total time=
5.3s
[CV 1/3] END .....C=10, gamma=0.001, kernel=rbf;, score=0.815 total time=
5.8s
[CV 2/3] END .....C=10, gamma=0.001, kernel=rbf;, score=0.805 total time=
6.6s
[CV 3/3] END .....C=10, gamma=0.001, kernel=rbf;, score=0.785 total time=
6.4s
[CV 1/3] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.735 total time=
6.1s
[CV 2/3] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.739 total time=
8.5s
[CV 3/3] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.772 total time=
7.3s
[CV 1/3] END ........C=100, gamma=1, kernel=rbf;, score=0.765 total time=  1
6.6s
[CV 2/3] END ........C=100, gamma=1, kernel=rbf;, score=0.767 total time=  1
3.5s
[CV 3/3] END ........C=100, gamma=1, kernel=rbf;, score=0.758 total time=  1
2.8s
[CV 1/3] END ......C=100, gamma=0.1, kernel=rbf;, score=0.760 total time=  1
4.2s
[CV 2/3] END ......C=100, gamma=0.1, kernel=rbf;, score=0.743 total time=  1
6.2s
[CV 3/3] END ......C=100, gamma=0.1, kernel=rbf;, score=0.744 total time=  1
5.8s
[CV 1/3] END .....C=100, gamma=0.01, kernel=rbf;, score=0.823 total time=  1
0.9s
[CV 2/3] END .....C=100, gamma=0.01, kernel=rbf;, score=0.806 total time=
8.9s
[CV 3/3] END .....C=100, gamma=0.01, kernel=rbf;, score=0.783 total time=
8.9s
[CV 1/3] END ....C=100, gamma=0.001, kernel=rbf;, score=0.819 total time=
9.4s
[CV 2/3] END ....C=100, gamma=0.001, kernel=rbf;, score=0.800 total time=
5.4s
[CV 3/3] END ....C=100, gamma=0.001, kernel=rbf;, score=0.786 total time=
```

```
5.4s
[CV 1/3] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.816 total time=
5.9s
[CV 2/3] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.804 total time=
6.4s
[CV 3/3] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.783 total time=
7.4s
[CV 1/3] END .......C=1000, gamma=1, kernel=rbf;, score=0.753 total time=  2
0.6s
[CV 2/3] END .......C=1000, gamma=1, kernel=rbf;, score=0.758 total time=  1
7.1s
[CV 3/3] END .......C=1000, gamma=1, kernel=rbf;, score=0.757 total time=  1
7.9s
[CV 1/3] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.731 total time=  2
3.7s
[CV 2/3] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.724 total time=  2
4.5s
[CV 3/3] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.715 total time=  1
9.9s
[CV 1/3] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.813 total time=  2
1.3s
[CV 2/3] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.797 total time=  1
3.5s
[CV 3/3] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.772 total time=  1
3.3s
[CV 1/3] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.818 total time=
6.2s
[CV 2/3] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.795 total time=
5.4s
[CV 3/3] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.782 total time=
6.2s
[CV 1/3] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.816 total time=
5.4s
[CV 2/3] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.804 total time=
5.3s
[CV 3/3] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.781 total time=
4.8s
```

Out[41]:

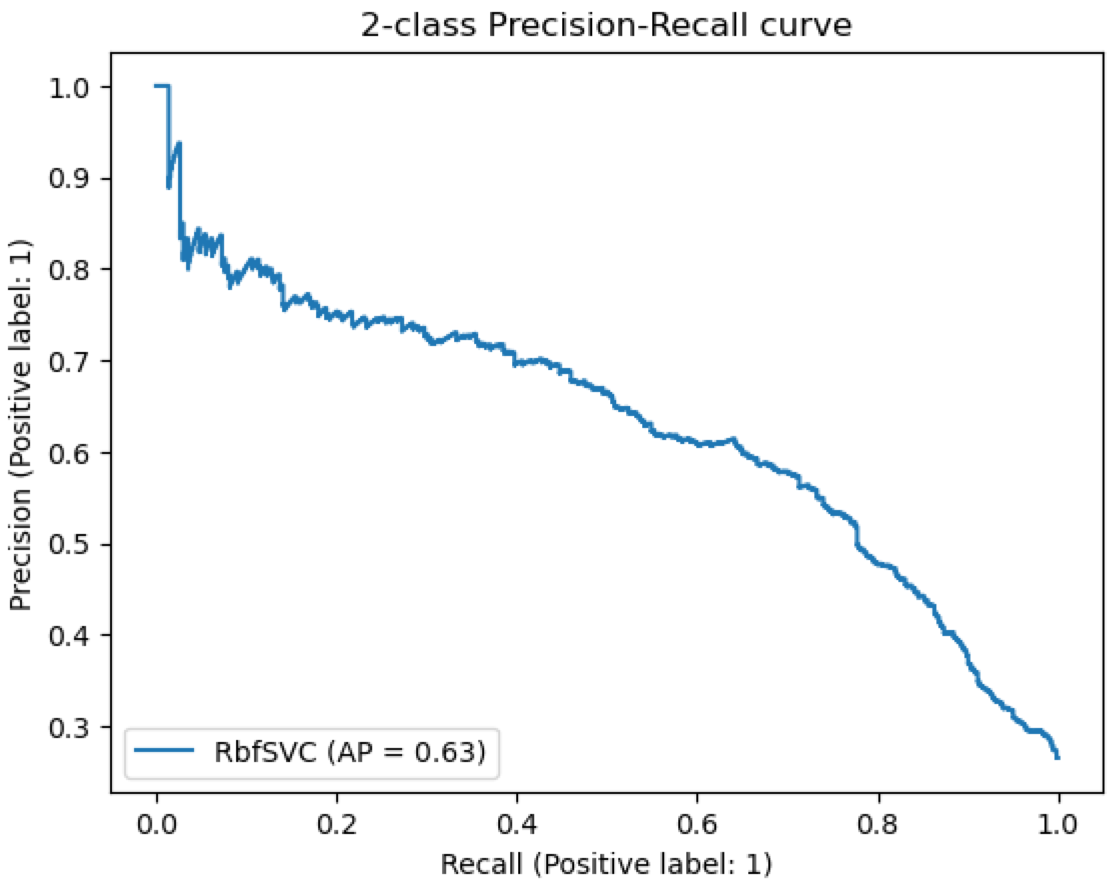
```
▸ GridSearchCV
▸ estimator: SVC
    ▸ SVC
```

In [42]:
```
y_pred_SVC = SVC_grid.predict(X_test)
```

In [43]:
```
print(accuracy_score(y_test, y_pred_SVC)) #rbf kernel gives the best accuracy c
```

```
0.7998106956933271
```

In [44]:
```
display = PrecisionRecallDisplay.from_estimator(
    SVC_grid, X_test, y_test, name="RbfSVC"
)
_ = display.ax_.set_title("2-class Precision-Recall curve") #Average precision
```



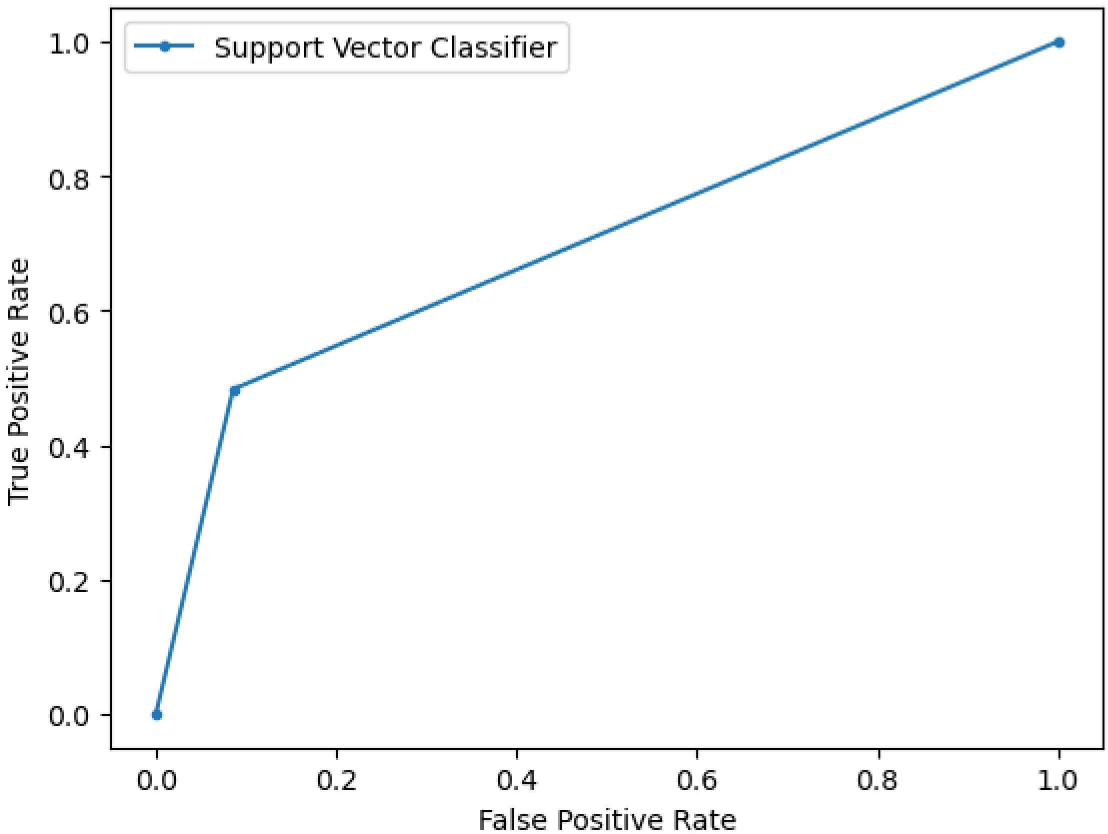
In [45]:
```
print(classification_report(y_test, y_pred_SVC)) #model precision and recall nc
```

```
              precision    recall  f1-score   support

           0       0.83      0.91      0.87      1552
           1       0.67      0.48      0.56       561

    accuracy                           0.80      2113
   macro avg       0.75      0.70      0.72      2113
weighted avg       0.79      0.80      0.79      2113
```

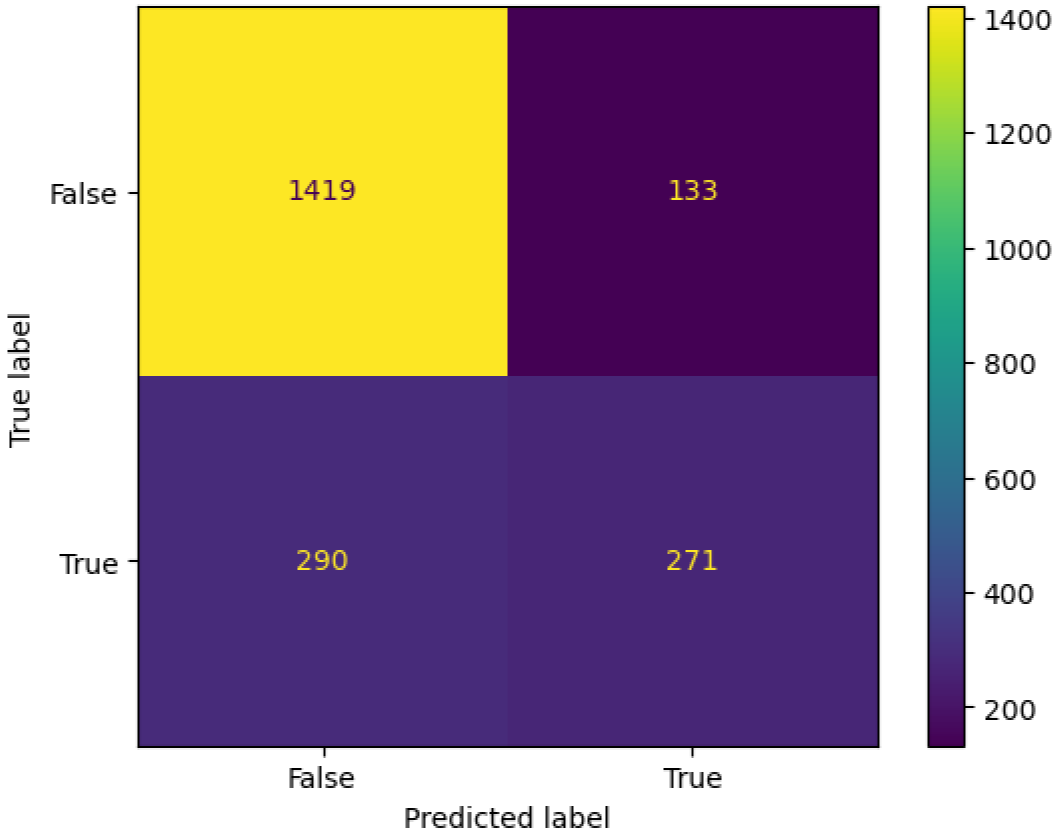
In [46]:
```
roc_auc_score(y_test, y_pred_SVC)
```

Out[46]:
```
0.6986850386827647
```

In [47]:
```
lr_fpr_SVC, lr_tpr_SVC, _ = roc_curve(y_test, y_pred_SVC)
# plot the roc curve for the model
plt.plot(lr_fpr_SVC, lr_tpr_SVC, marker='.', label='Support Vector Classifier')
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# show the legend
plt.legend()
# show the plot
plt.show()
```



In [48]:
```
confusion_matrix_SVC = confusion_matrix(y_test, y_pred_SVC)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matri

cm_display.plot()
plt.show()
```



Decision Tree Classifier

In [49]:
```
params = {'max_leaf_nodes': list(range(2, 100)), 'min_samples_split': [2, 3, 4]
grid_search_cv = GridSearchCV(DecisionTreeClassifier(random_state=42), params,

grid_search_cv.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 294 candidates, totalling 882 fits
```

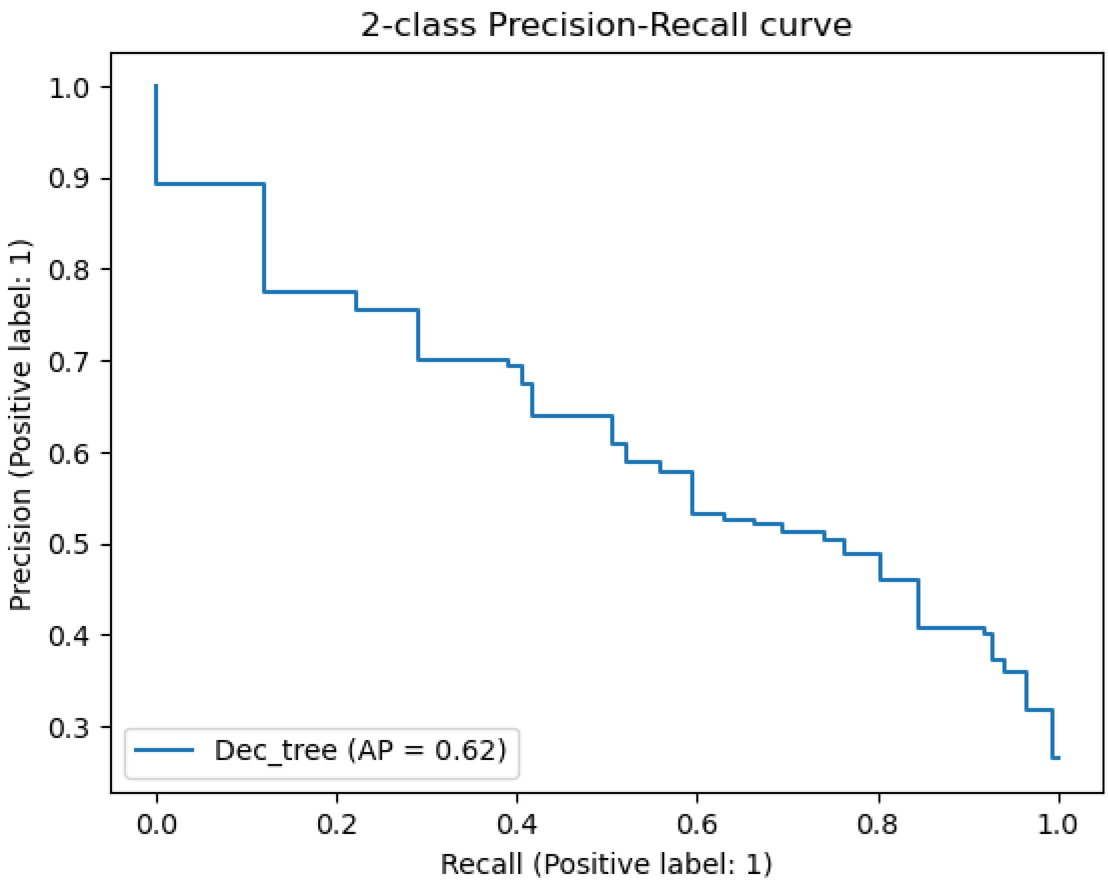
Out[49]:
```
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

In [50]:
```
y_pred_dec = grid_search_cv.predict(X_test)
```

In [51]:
```
print(accuracy_score(y_test, y_pred_dec))
```

```
0.7931850449597728
```

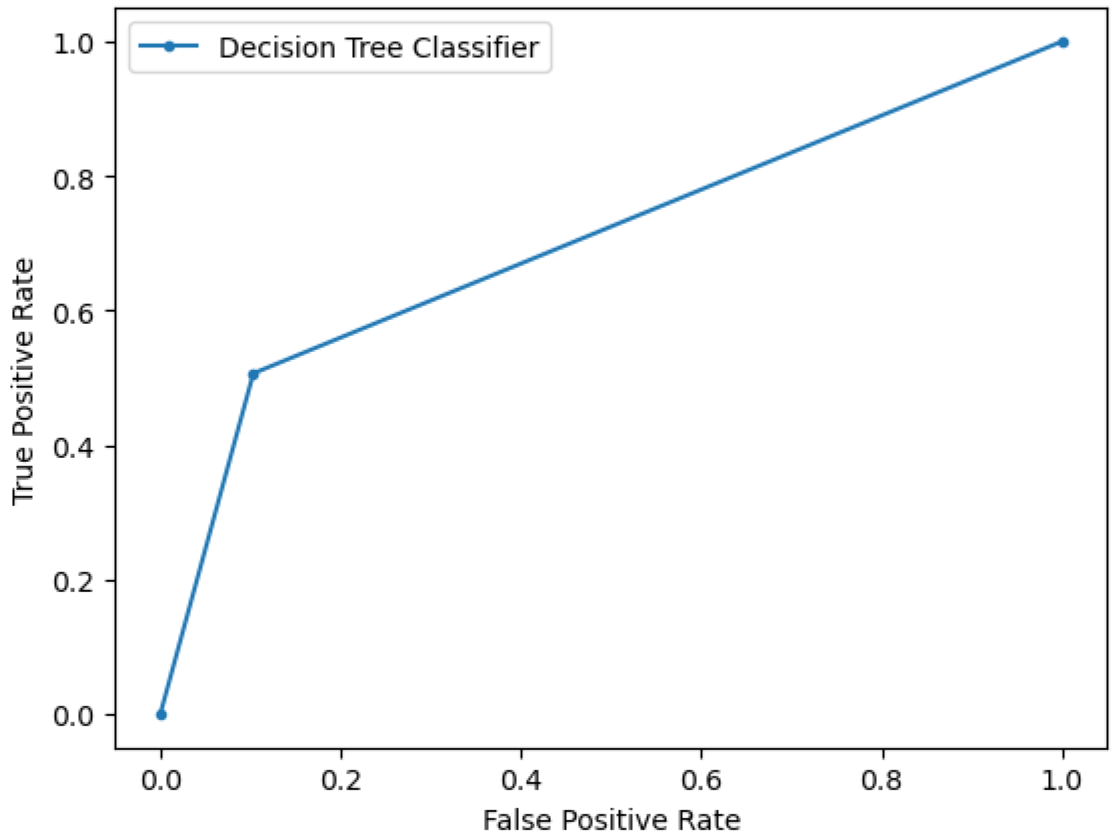
In [52]:
```
display = PrecisionRecallDisplay.from_estimator(
    grid_search_cv, X_test, y_test, name="Dec_tree"
)
_ = display.ax_.set_title("2-class Precision-Recall curve")
```



In [53]:
```
roc_auc_score(y_test, y_pred_dec)
```

Out[53]:
```
0.7015730378374406
```

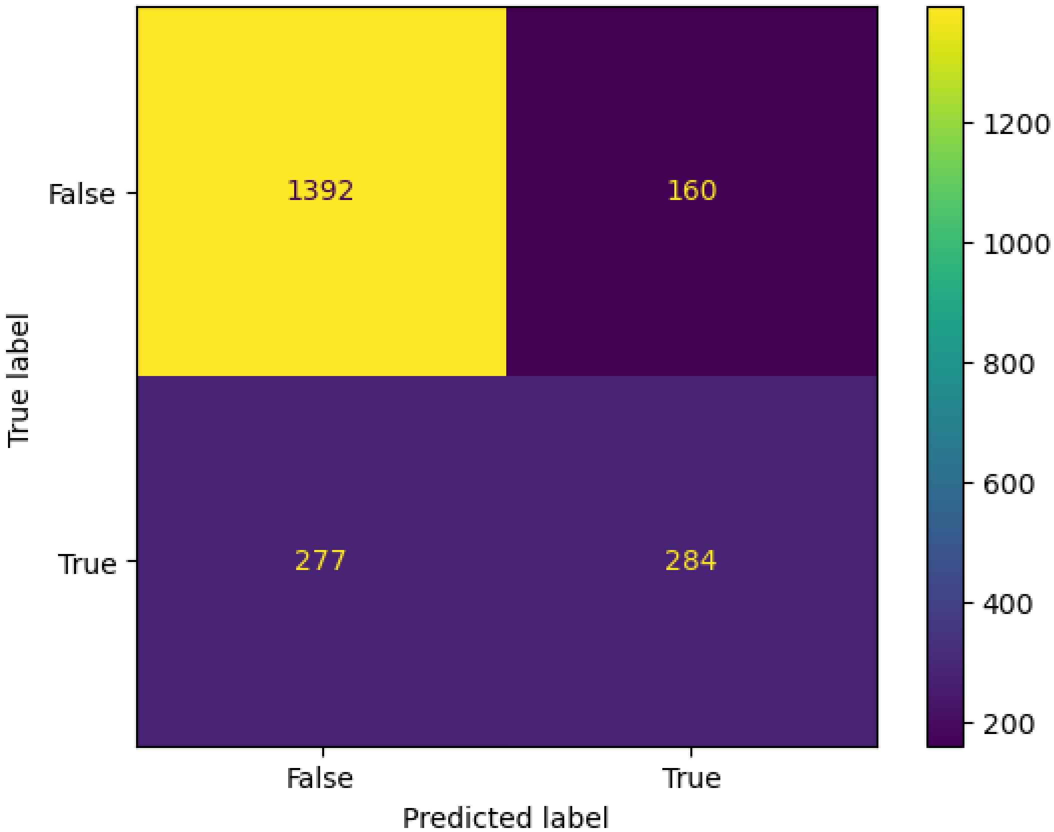
In [54]:
```
lr_fpr_dec, lr_tpr_dec, _ = roc_curve(y_test, y_pred_dec)
# plot the roc curve for the model
plt.plot(lr_fpr_dec, lr_tpr_dec, marker='.', label='Decision Tree Classifier')
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# show the legend
plt.legend()
# show the plot
plt.show()
```



In [55]:
```
print(classification_report(y_test, y_pred_dec))
```

```
              precision    recall  f1-score   support

           0       0.83      0.90      0.86      1552
           1       0.64      0.51      0.57       561

    accuracy                           0.79      2113
   macro avg       0.74      0.70      0.71      2113
weighted avg       0.78      0.79      0.78      2113
```

In [56]:
```
confusion_matrix_dec = confusion_matrix(y_test, y_pred_dec)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matri

cm_display.plot()
plt.show()
```



Naive Bayes Classifier

In [57]:
```
nb_classifier = GaussianNB()

params_NB = {'var_smoothing': np.logspace(0,-9, num=100)}
gs_NB = GridSearchCV(estimator=nb_classifier,
                 param_grid=params_NB,
                 cv= 3,
                 verbose=1,
                 scoring='accuracy')
gs_NB.fit(X_train, y_train)

gs_NB.best_params_
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

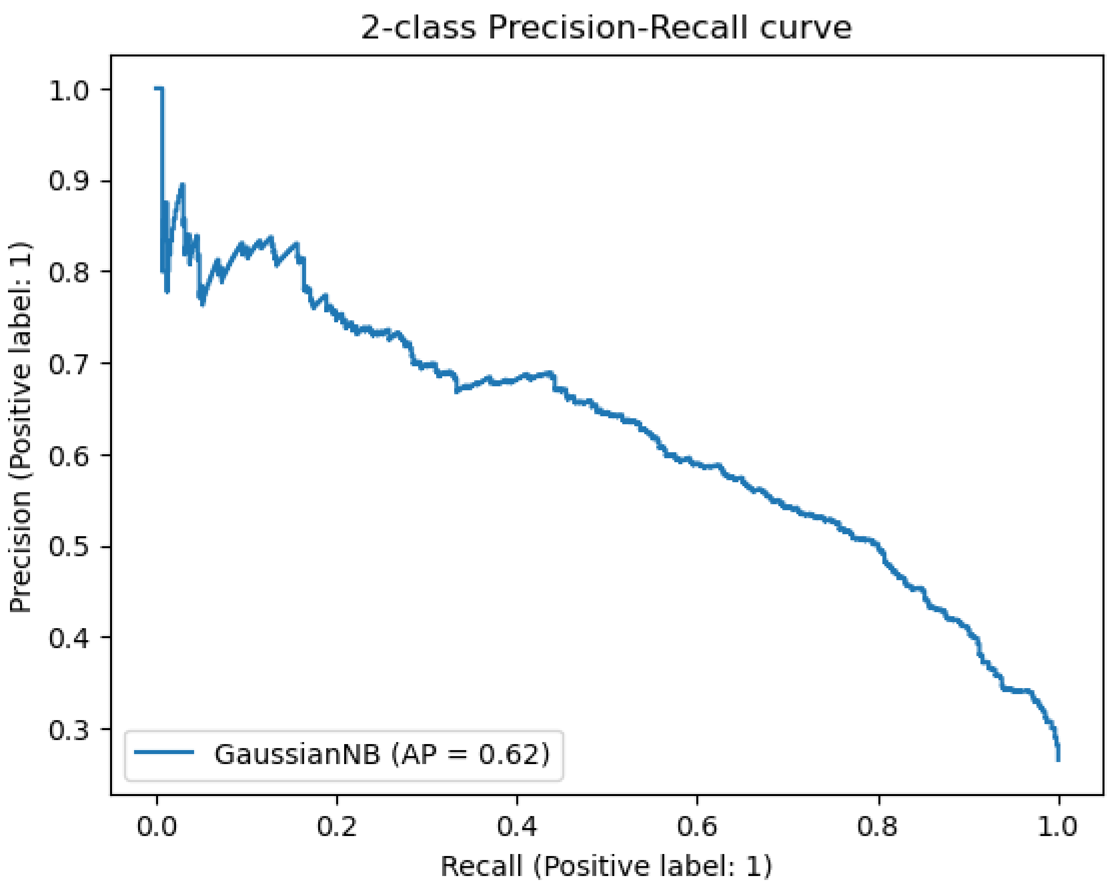
Out[57]:
```
{'var_smoothing': 1.0}
```

In [58]:
```
y_pred_NB = gs_NB.predict(X_test)
```

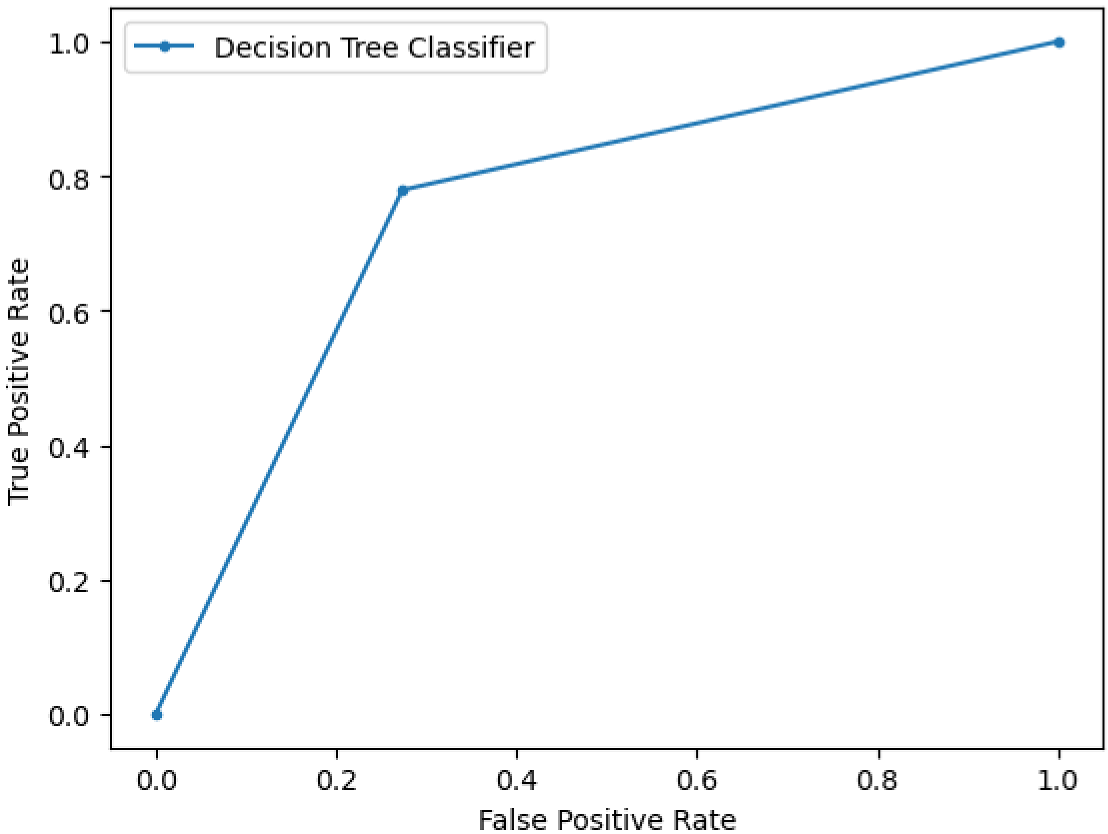
In [59]:
```
print(accuracy_score(y_test, y_pred_NB))
```

```
0.7406530998580217
```

In [60]:
```
display = PrecisionRecallDisplay.from_estimator(
    gs_NB, X_test, y_test, name="GaussianNB"
)
_ = display.ax_.set_title("2-class Precision-Recall curve")
```



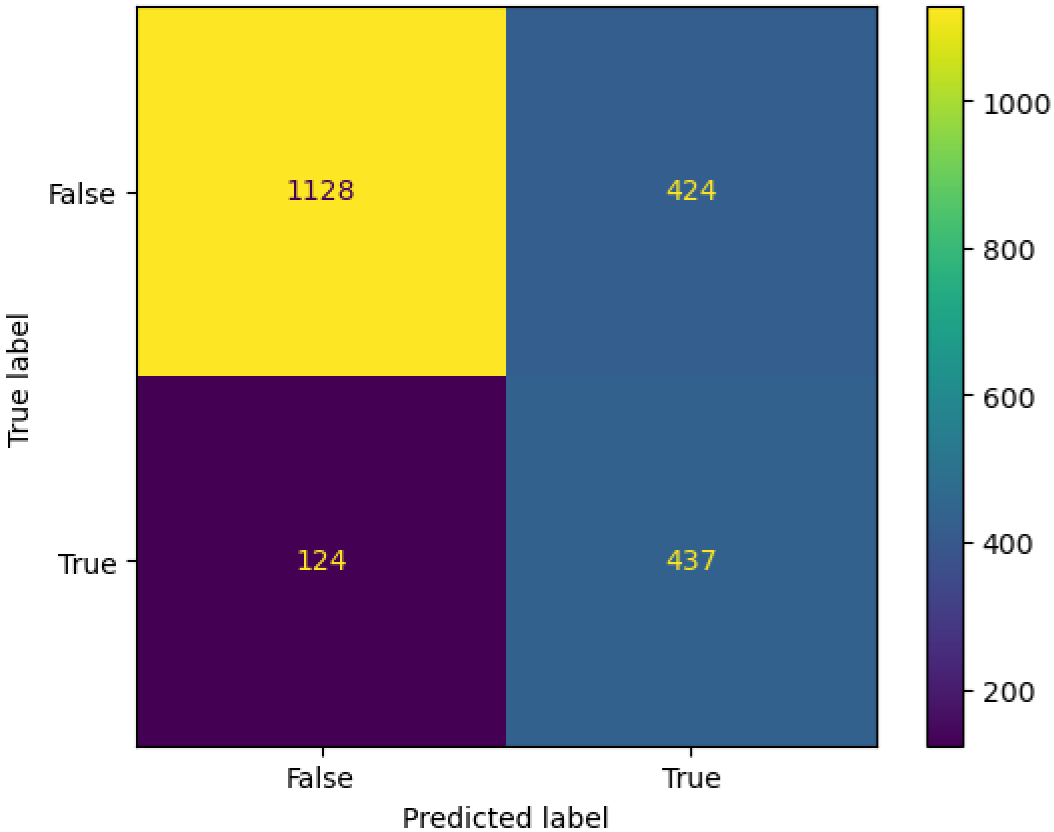
In [61]:
```
lr_fpr_NB, lr_tpr_NB, _ = roc_curve(y_test, y_pred_NB)
# plot the roc curve for the model
plt.plot(lr_fpr_NB, lr_tpr_NB, marker='.', label='Decision Tree Classifier')
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# show the legend
plt.legend()
# show the plot
plt.show()
```



In [62]:
```
print(classification_report(y_test, y_pred_NB))
```

```
              precision    recall  f1-score   support

           0       0.90      0.73      0.80      1552
           1       0.51      0.78      0.61       561

    accuracy                           0.74      2113
   macro avg       0.70      0.75      0.71      2113
weighted avg       0.80      0.74      0.75      2113
```

In [63]:
```
confusion_matrix_NB = confusion_matrix(y_test, y_pred_NB)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matri

cm_display.plot()
plt.show()
```



Random Forest

In [64]:
```
param_grid_rfc = {
    'n_estimators': [50, 150],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}
```

In [65]:
```
rfc=RandomForestClassifier(random_state=123)
```

In [66]:
```
CV_rfc = GridSearchCV(estimator=rfc, param_grid = param_grid_rfc, cv= 3)
CV_rfc.fit(X_train, y_train)
```

```
E:\Anaconda Nav\lib\site-packages\sklearn\ensemble\_forest.py:424: FutureWa
rning: `max_features='auto'` has been deprecated in 1.1 and will be removed
in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or
remove this parameter as it is also the default value for RandomForestClass
ifiers and ExtraTreesClassifiers.
  warn(
E:\Anaconda Nav\lib\site-packages\sklearn\ensemble\_forest.py:424: FutureWa
rning: `max_features='auto'` has been deprecated in 1.1 and will be removed
in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or
remove this parameter as it is also the default value for RandomForestClass
ifiers and ExtraTreesClassifiers.
  warn(
E:\Anaconda Nav\lib\site-packages\sklearn\ensemble\_forest.py:424: FutureWa
rning: `max_features='auto'` has been deprecated in 1.1 and will be removed
in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or
remove this parameter as it is also the default value for RandomForestClass
ifiers and ExtraTreesClassifiers.
  warn(
E:\Anaconda Nav\lib\site-packages\sklearn\ensemble\_forest.py:424: FutureWa
```

In [67]:
```
CV_rfc.best_params_
```

Out[67]:
```
{'criterion': 'entropy',
 'max_depth': 8,
 'max_features': 'auto',
 'n_estimators': 150}
```

In [68]:
```
rfc1=RandomForestClassifier(random_state=42, max_features='auto', n_estimators=
```

In [69]:
```
rfc1.fit(X_train, y_train)
```

```
E:\Anaconda Nav\lib\site-packages\sklearn\ensemble\_forest.py:424: FutureWarn
ing: `max_features='auto'` has been deprecated in 1.1 and will be removed in
1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remo
ve this parameter as it is also the default value for RandomForestClassifiers
and ExtraTreesClassifiers.
  warn(
```

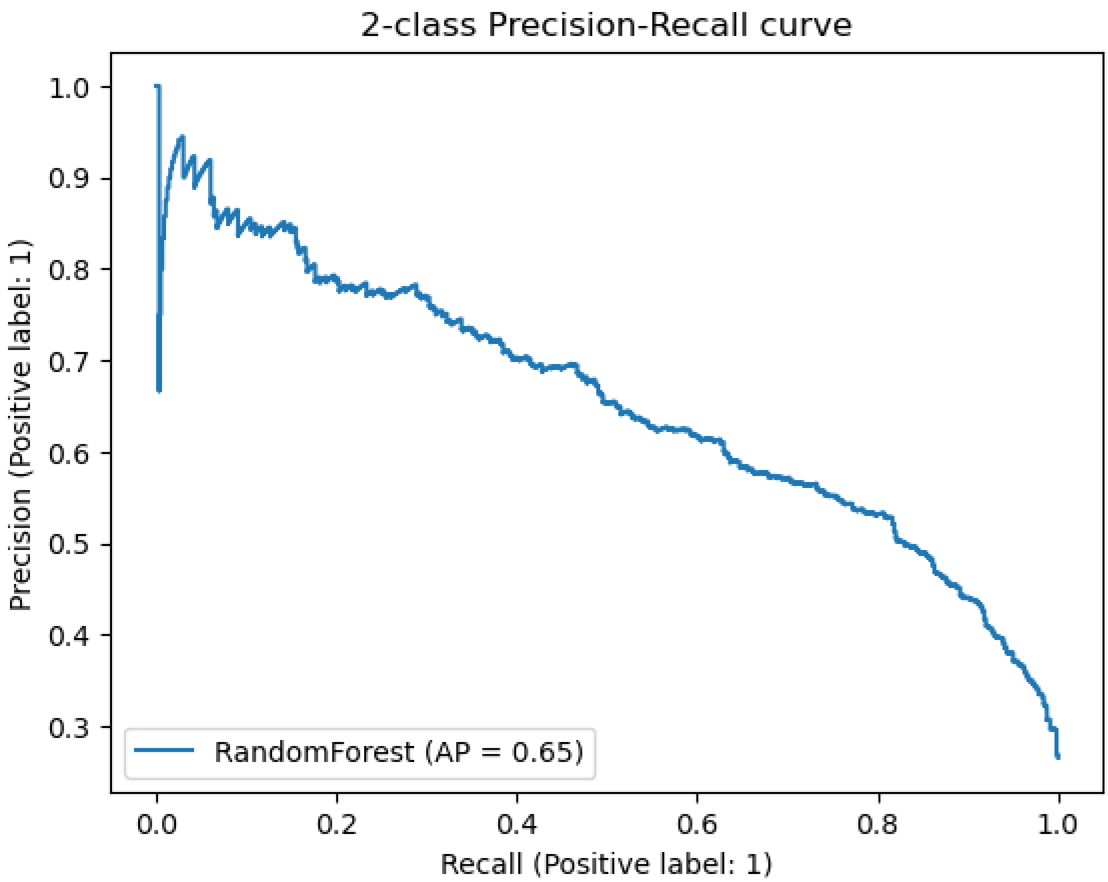
Out[69]:
```
RandomForestClassifier

RandomForestClassifier(criterion='entropy', max_depth=8, max_features='aut
o',
                       n_estimators=200, random_state=42)
```

In [70]:
```
y_pred_rfc = rfc1.predict(X_test)
```

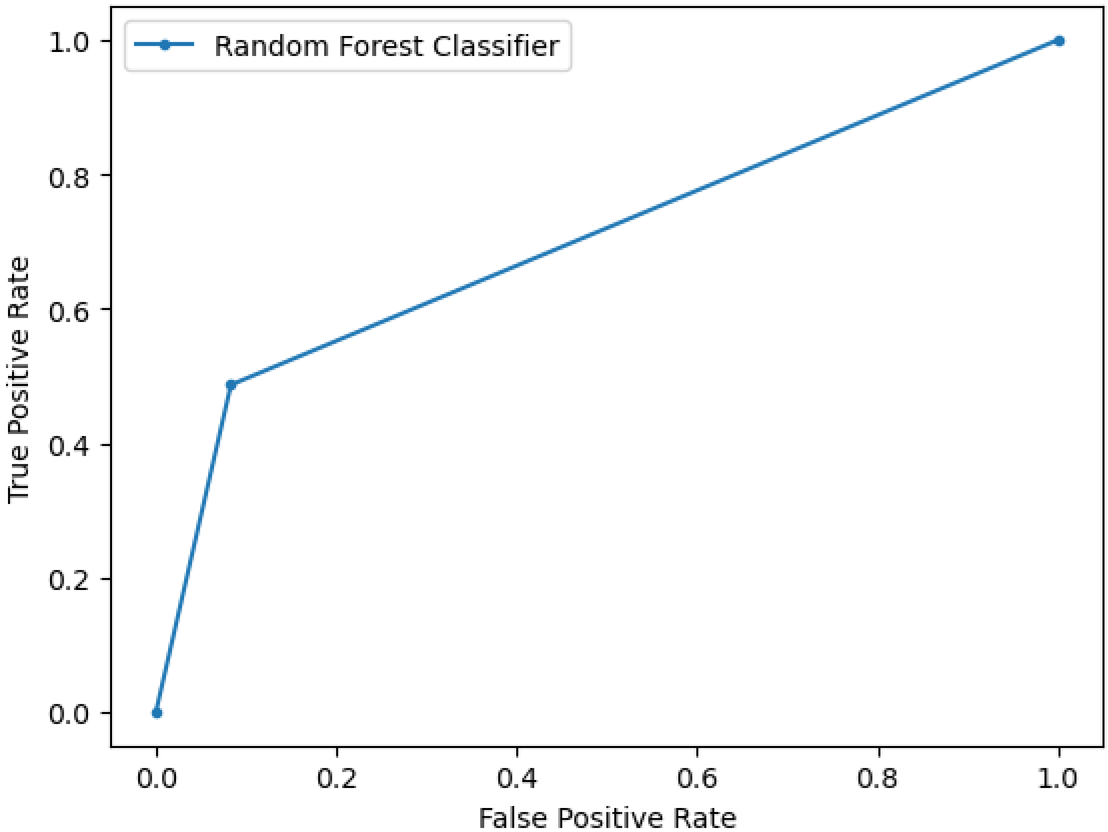
In [71]:
```
print(accuracy_score(y_test, y_pred_rfc))
```

```
0.8026502602934217
```

In [72]:
```
display = PrecisionRecallDisplay.from_estimator(
    rfc1, X_test, y_test, name="RandomForest"
)
_ = display.ax_.set_title("2-class Precision-Recall curve")
```



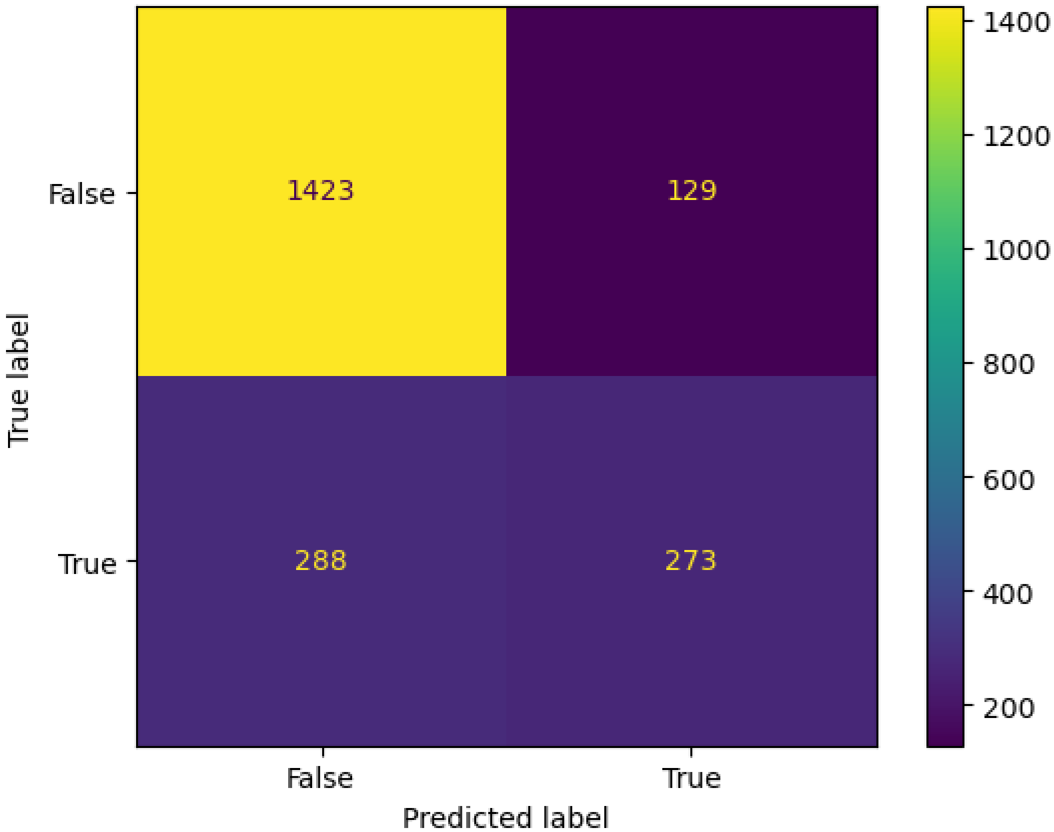
In [73]:
```
lr_fpr_rfc, lr_tpr_rfc, _ = roc_curve(y_test, y_pred_rfc)
# plot the roc curve for the model
plt.plot(lr_fpr_rfc, lr_tpr_rfc, marker='.', label='Random Forest Classifier')
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# show the legend
plt.legend()
# show the plot
plt.show()
```



In [74]:
```
print(classification_report(y_test, y_pred_rfc))
```

```
              precision    recall  f1-score   support

           0       0.83      0.92      0.87      1552
           1       0.68      0.49      0.57       561

    accuracy                           0.80      2113
   macro avg       0.76      0.70      0.72      2113
weighted avg       0.79      0.80      0.79      2113
```

In [75]:
```
confusion_matrix_rfc = confusion_matrix(y_test, y_pred_rfc)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matri

cm_display.plot()
plt.show()
```



Stochastic Gradient Classifier

In [76]:
```
param_dist_SGD = {
    'loss': ['hinge', 'log', 'perceptron'],
    'alpha': [0.0001, 0.001, 0.01, 0.1],
    'penalty': ['l2', 'l1', 'elasticnet']
}

model = SGDClassifier()
randomized_search = RandomizedSearchCV(model, param_dist_SGD, n_iter=10, cv=5)
randomized_search.fit(X_train, y_train)
```

```
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
```

```
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
```

Out[76]:

```
▸ RandomizedSearchCV
▸ estimator: SGDClassifier
    ▸ SGDClassifier
```

In [77]:
```
randomized_search.best_params_
```

Out[77]:
```
{'penalty': 'l1', 'loss': 'log', 'alpha': 0.001}
```

In [78]:
```
SGD1 = SGDClassifier(random_state = 42, penalty = "l2", loss = "log", alpha = (
SGD1.fit(X_train, y_train)
```

```
E:\Anaconda Nav\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be remov
ed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
```

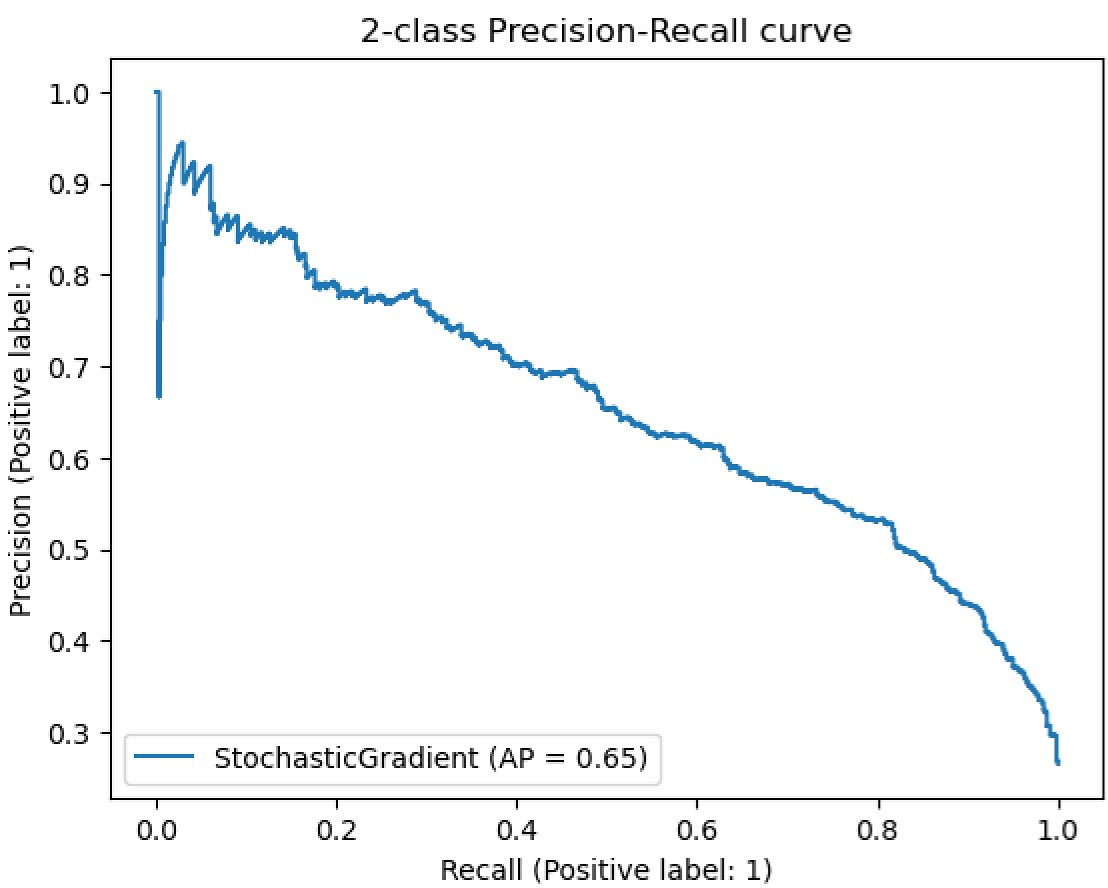
Out[78]:
```
SGDClassifier
SGDClassifier(alpha=0.001, loss='log', random_state=42)
```

In [79]:
```
y_pred_SGD = SGD1.predict(X_test)
```

In [80]:
```
print(accuracy_score(y_test, y_pred_SGD))
```

```
0.7927117841930904
```

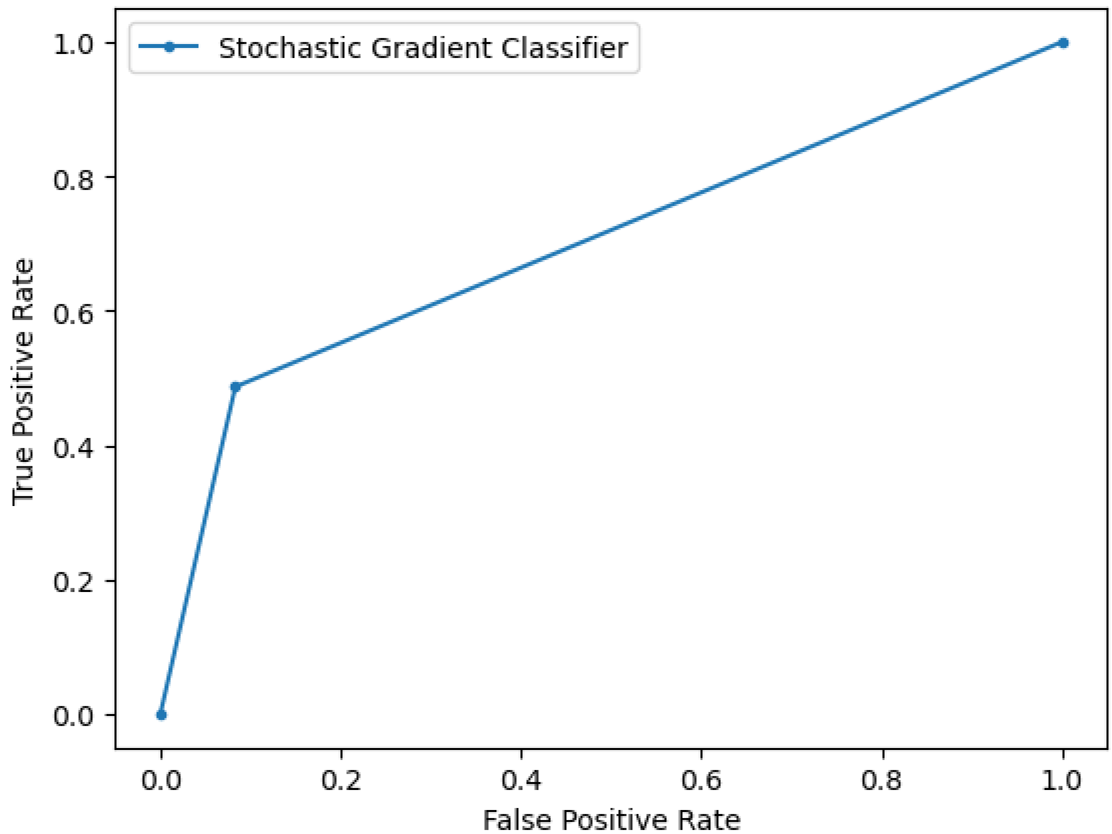
In [81]:
```
display = PrecisionRecallDisplay.from_estimator(
    rfc1, X_test, y_test, name="StochasticGradient"
)
_ = display.ax_.set_title("2-class Precision-Recall curve")
```



In [82]:
```
roc_auc_score(y_test, y_pred_SGD)
```

Out[82]:
```
0.7393806163515076
```

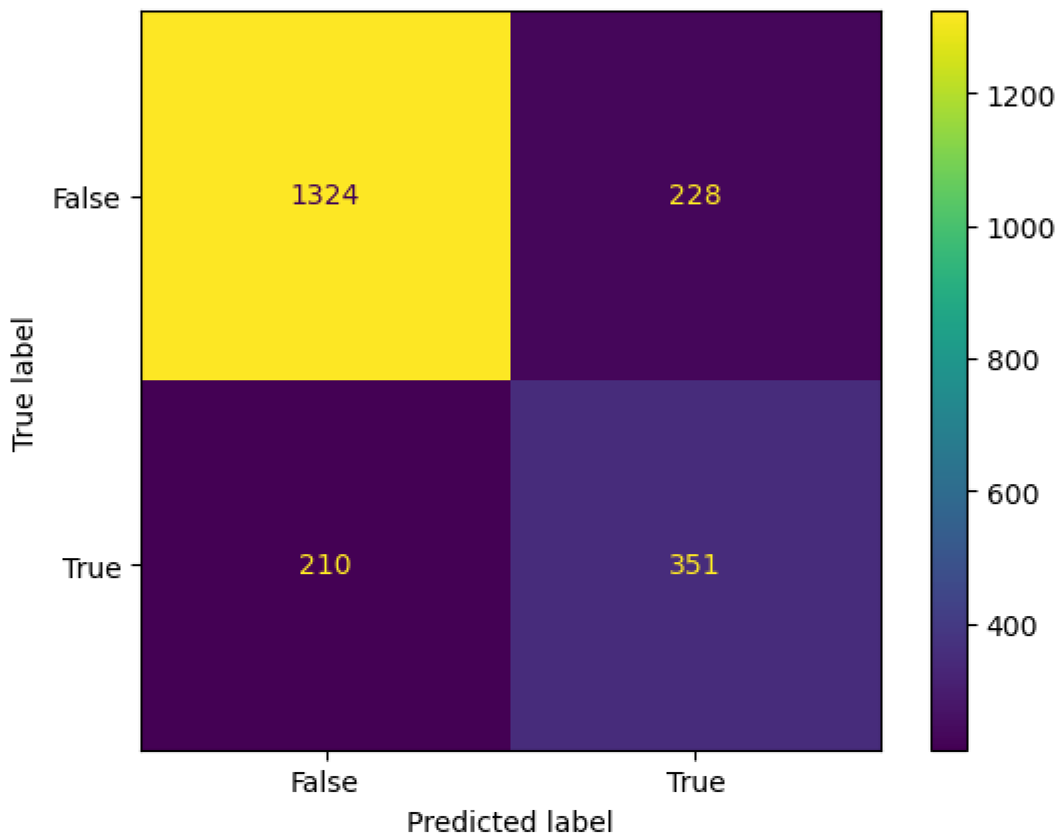
In [83]:
```
lr_fpr_SGD, lr_tpr_SGD, _ = roc_curve(y_test, y_pred_rfc)
# plot the roc curve for the model
plt.plot(lr_fpr_SGD, lr_tpr_SGD, marker='.', label='Stochastic Gradient Classi
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# show the legend
plt.legend()
# show the plot
plt.show()
```



In [84]:
```
print(classification_report(y_test, y_pred_SGD))
```

```
              precision    recall  f1-score   support

           0       0.86      0.85      0.86      1552
           1       0.61      0.63      0.62       561

    accuracy                           0.79      2113
   macro avg       0.73      0.74      0.74      2113
weighted avg       0.79      0.79      0.79      2113
```

In [85]:
```
confusion_matrix_SGD = confusion_matrix(y_test, y_pred_SGD)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matri

cm_display.plot()
plt.show()
```



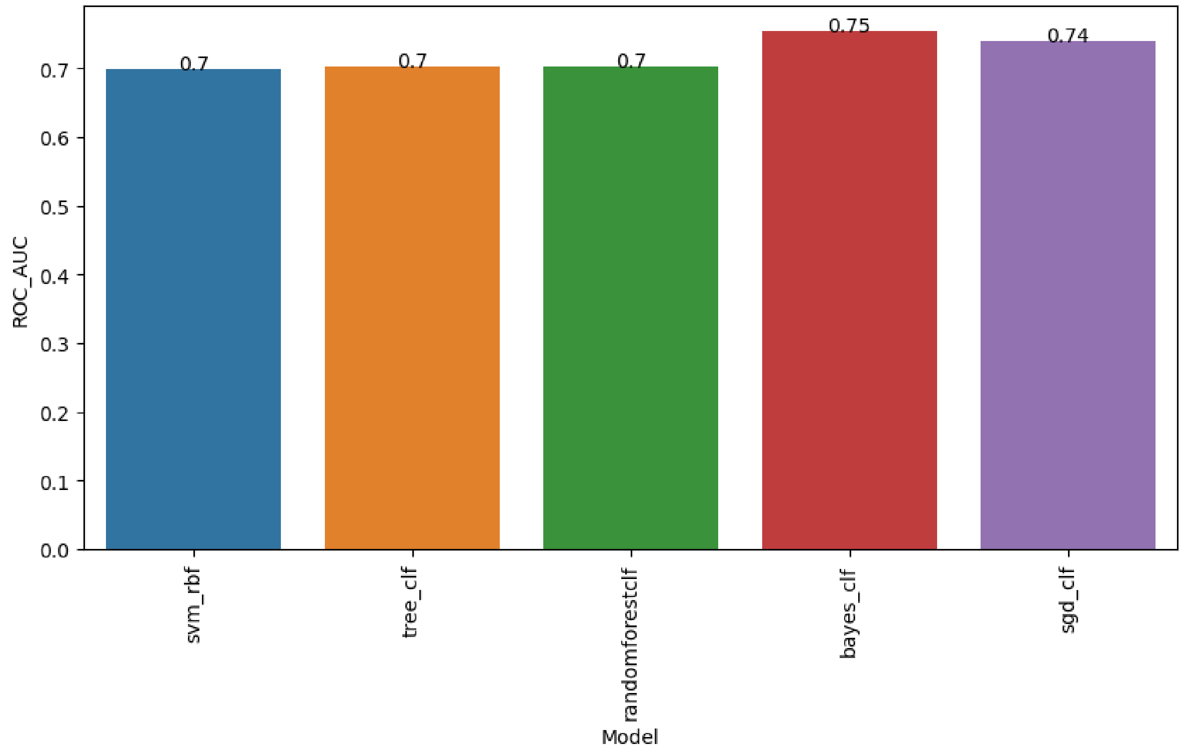
Summary

In [87]:
```
#calculate roc_auc score of each and save to a dataframe
roc_auc_scores = pd.DataFrame(columns=['Model','ROC_AUC'])
roc_auc_scores.loc[0] = ['svm_rbf',roc_auc_score(y_test, SVC_grid.predict(X_te
roc_auc_scores.loc[1] = ['tree_clf',roc_auc_score(y_test,grid_search_cv.predic
roc_auc_scores.loc[2] = ['randomforestclf',roc_auc_score(y_test,rfc1.predict(X_
roc_auc_scores.loc[3] = ['bayes_clf',roc_auc_score(y_test,gs_NB.predict(X_test
roc_auc_scores.loc[4] = ['sgd_clf',roc_auc_score(y_test,SGD1.predict(X_test))]
roc_auc_scores
```

Out[87]:

	Model	ROC_AUC
0	svm_rbf	0.698685
1	tree_clf	0.701573
2	randomforestclf	0.701756
3	bayes_clf	0.752885
4	sgd_clf	0.739381

In [88]:
```
#make a bar chart to show the highest values of roc_auc with values also printe
plt.figure(figsize=(10,5))
sns.barplot(x='Model',y='ROC_AUC',data=roc_auc_scores)
plt.xticks(rotation=90)
for i in range(len(roc_auc_scores)):
    plt.text(i,roc_auc_scores['ROC_AUC'][i],round(roc_auc_scores['ROC_AUC'][i],
plt.show()
```



Thus bayes_clf is the best estimator for Churn Modelling