Final Project

November 29, 2022

Team Member Names: Madeline Witters

Project Title: Predicting Customer Churn and Identifying Attributes of At-Risk Customers

Exploratory Data Analysis

[1]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LassoCV
from sklearn.preprocessing import StandardScaler
from statsmodels.graphics.mosaicplot import mosaic
from sklearn.linear_model import LogisticRegressionCV
from scipy import stats
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc
from sklearn.metrics import roc_curve
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import PrecisionRecallDisplay
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

[2]:
```
data = pd.read_csv("Bank Customer Churn Prediction.csv")
data = data.drop("customer_id", axis=1)
data.head()
```

[2]:
```
   credit_score country  gender  age  tenure    balance  products_number  \
0           619  France  Female   42       2       0.00                1
1           608   Spain  Female   41       1   83807.86                1
2           502  France  Female   42       8  159660.80                3
```

```
3          699  France  Female   39       1       0.00                2
4          850   Spain  Female   43       2  125510.82                1

   credit_card  active_member  estimated_salary  churn
0            1              1         101348.88      1
1            0              1         112542.58      0
2            1              0         113931.57      1
3            0              0          93826.63      0
4            1              1          79084.10      0
```

[3]:
```
data.dtypes
```

[3]:
```
credit_score          int64
country              object
gender               object
age                   int64
tenure                int64
balance             float64
products_number       int64
credit_card           int64
active_member         int64
estimated_salary    float64
churn                 int64
dtype: object
```

[4]:
```
#info on why it's best to not do too much one-hot encoding for trees:
#https://towardsdatascience.com/
 ↪one-hot-encoding-is-making-your-tree-based-ensembles-worse-heres-why-d64b282b5769

#outlier removal (do before standardization):
#https://medium.com/geekculture/
 ↪essential-guide-to-handle-outliers-for-your-logistic-regression-model-63c97690a84d
```

[5]:
```
null_check = data.isnull().any() #no missing data in the dataframe
null_check
```

[5]:
```
credit_score        False
country             False
gender              False
age                 False
tenure              False
balance             False
products_number     False
credit_card         False
active_member       False
estimated_salary    False
churn               False
```

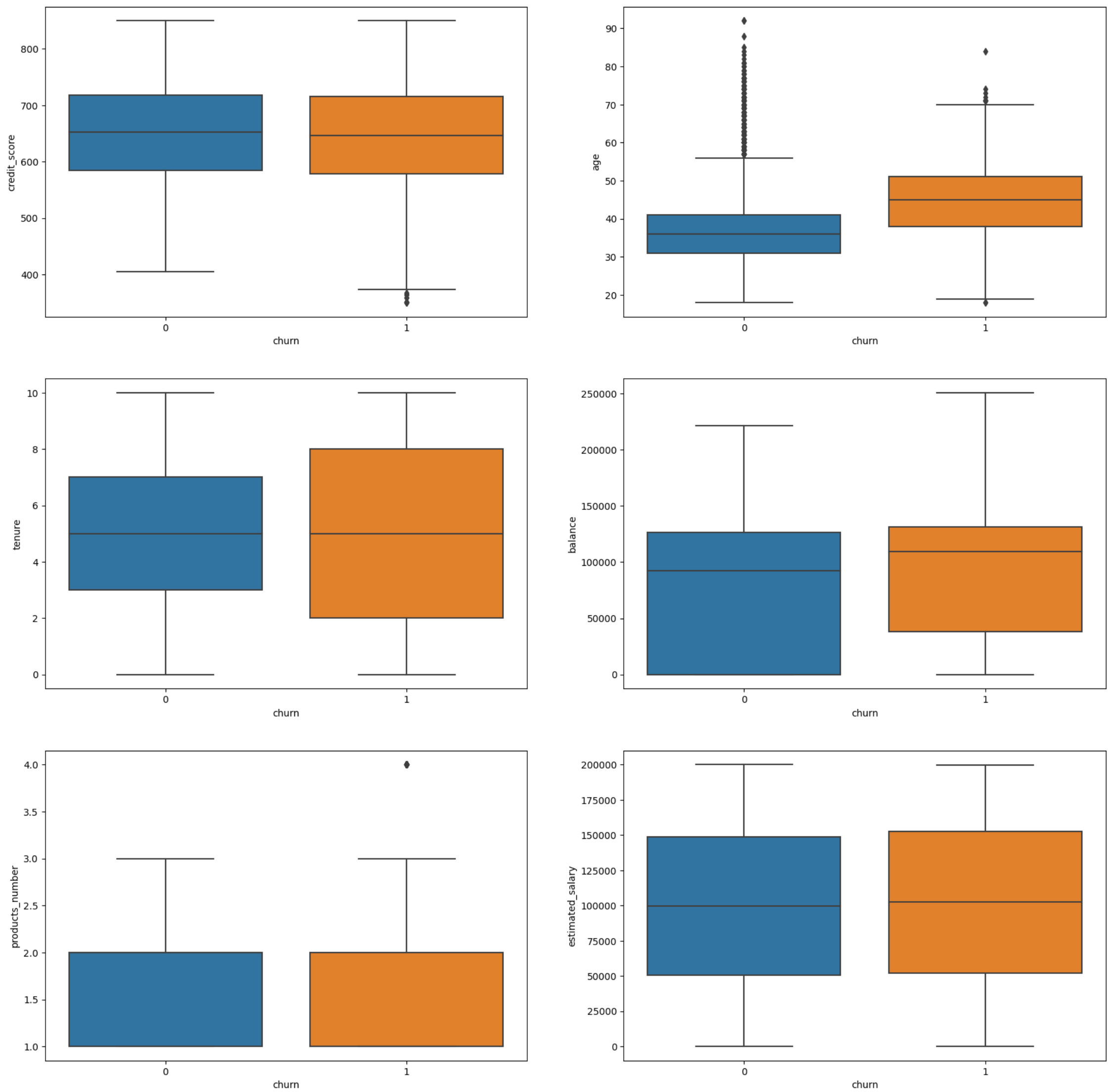
```
dtype: bool
```

[6]:
```
#data.min()
```

[7]:
```
#data.max()
```

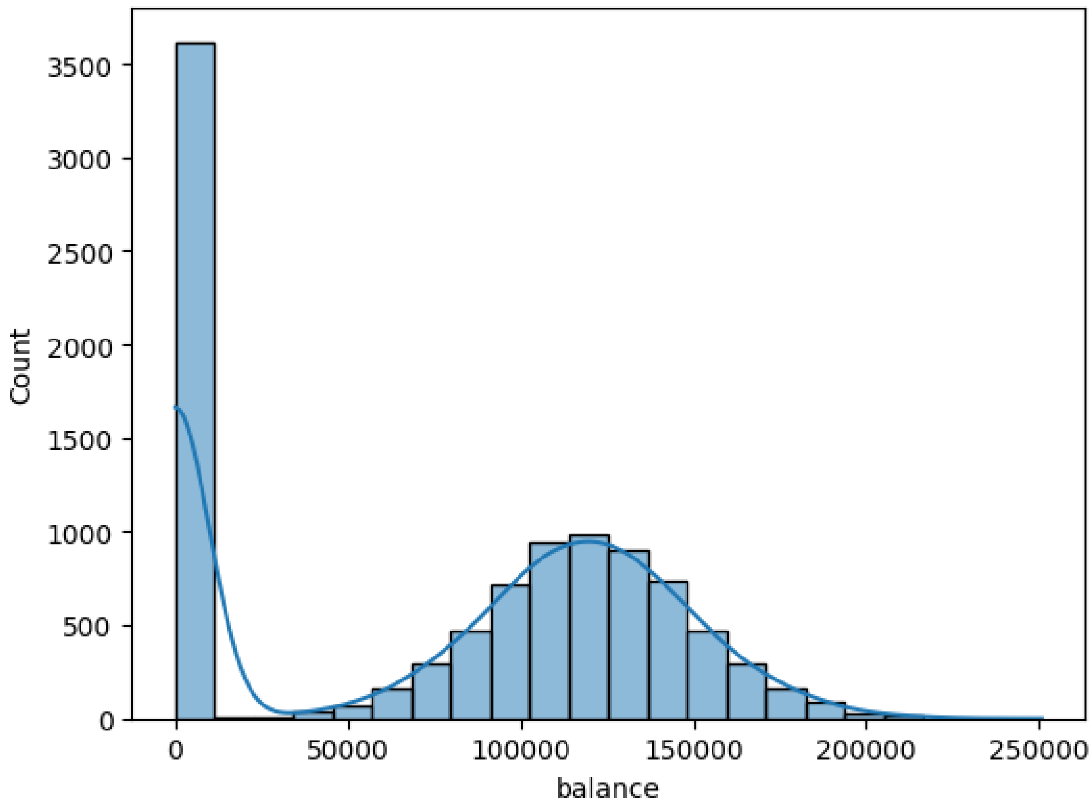
[8]:
```
plt.figure(figsize=(20,20))
#plt.title("Boxplots of Numeric Dependent Variables") try to add in tile later ?

plt.subplot(3,2,1)
sns.boxplot(x='churn', y='credit_score', data=data)

plt.subplot(3,2,2)
sns.boxplot(x='churn', y='age', data=data)

plt.subplot(3,2,3)
sns.boxplot(x='churn', y='tenure', data=data)

plt.subplot(3,2,4)
sns.boxplot(x='churn', y='balance', data=data)

plt.subplot(3,2,5)
sns.boxplot(x='churn', y='products_number', data=data)

plt.subplot(3,2,6)
sns.boxplot(x='churn', y='estimated_salary', data=data)
```

[8]:
```
<AxesSubplot:xlabel='churn', ylabel='estimated_salary'>
```



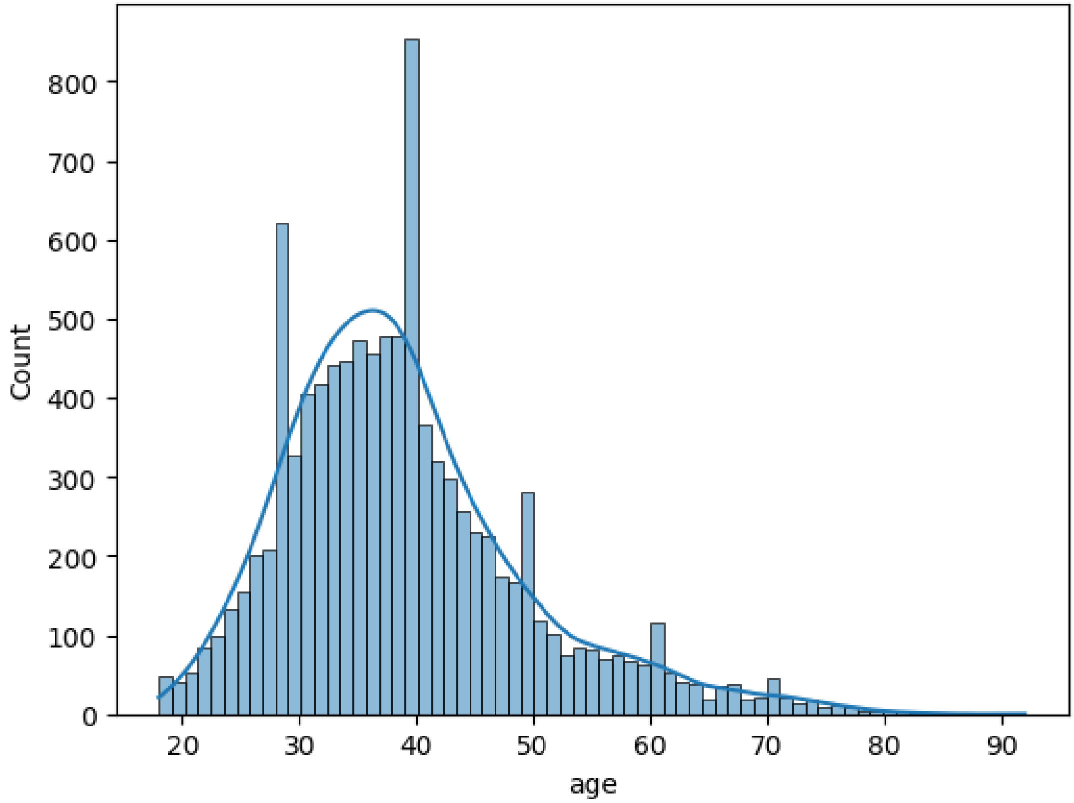

```
[9]: sns.histplot(data=data, x="balance", kde=True)
```

```
[9]: <AxesSubplot:xlabel='balance', ylabel='Count'>
```



[10]:
```
sns.histplot(data=data, x="age", kde=True)
```

[10]: <AxesSubplot:xlabel='age', ylabel='Count'>



[11]:
```
credit_score_zs = stats.zscore(data['credit_score'])
```

[12]:
```
age_z = stats.zscore(data['age'])
```

[13]:
```
ten_z = stats.zscore(data['tenure'])
print(ten_z)
```

```
0      -1.041760
1      -1.387538
2       1.032908
3      -1.387538
4      -1.041760
          …
9995   -0.004426
9996    1.724464
9997    0.687130
9998   -0.695982
9999   -0.350204
Name: tenure, Length: 10000, dtype: float64
```

[14]:
```python
bal_z = stats.zscore(data['balance'])
print(bal_z)
```

```
0      -1.225848
1       0.117350
2       1.333053
3      -1.225848
4       0.785728
          …
9995   -1.225848
9996   -0.306379
9997   -1.225848
9998   -0.022608
9999    0.859965
Name: balance, Length: 10000, dtype: float64
```

[15]:
```python
prod_z = stats.zscore(data['products_number'])
print(prod_z)
```

```
0      -0.911583
1      -0.911583
2       2.527057
3       0.807737
4      -0.911583
          …
9995    0.807737
9996   -0.911583
9997   -0.911583
9998    0.807737
9999   -0.911583
Name: products_number, Length: 10000, dtype: float64
```

[16]:
```python
sal_z = stats.zscore(data['estimated_salary'])
print(sal_z)
```

```
0       0.021886
1       0.216534
2       0.240687
3      -0.108918
4      -0.365276
          …
9995   -0.066419
9996    0.027988
9997   -1.008643
9998   -0.125231
9999   -1.076370
Name: estimated_salary, Length: 10000, dtype: float64
```

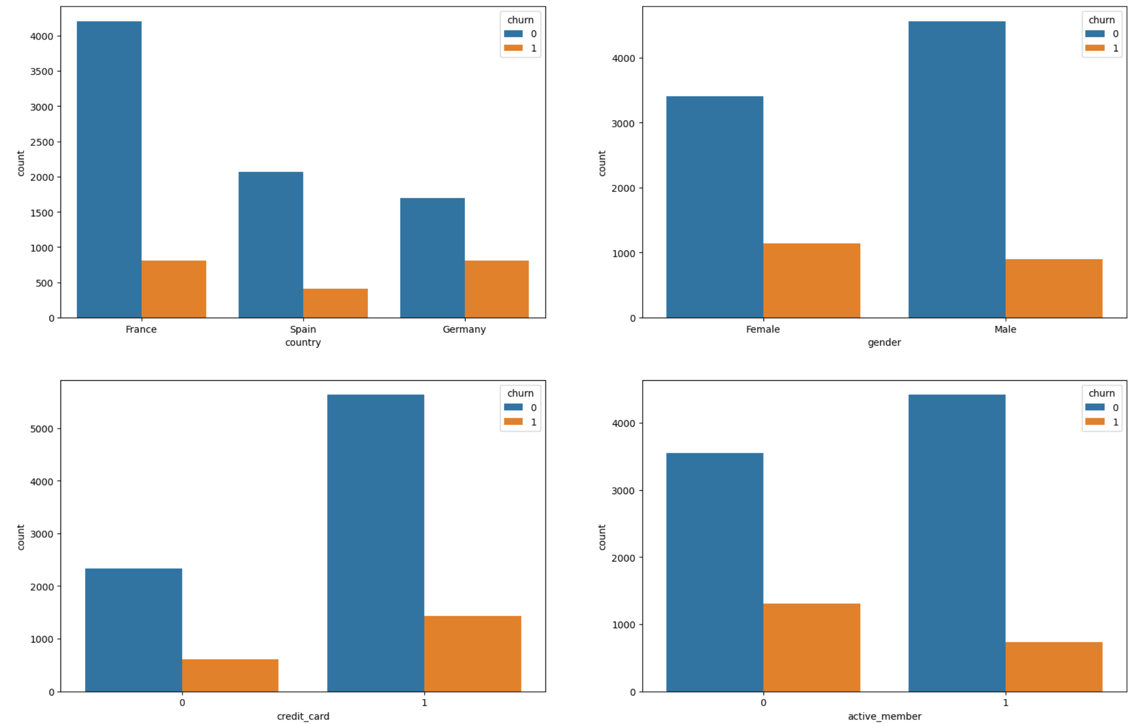
[17]:
```python
threshold = 3
outlier = [] #write in report about how I conducted outlier analysis, and why I
 ↪decided not to exclude any points
for z in age_z: #further address how this could be modeled in the future
 ↪(segment customers by age, products number) + build more models
    if z > threshold: #this could be addressed in conclusion or EDA section
        outlier.append(z)
#print('outlier in dataset is', outlier)
#print(len(outlier))
```

[18]:
```python
data['churn'].value_counts()
```

[18]:
```
0    7963
1    2037
Name: churn, dtype: int64
```

[19]:
```python
plt.figure(figsize=(20,20))
#plt.title("Boxplots of Numeric Dependent Variables") try to add in tile later ?

plt.subplot(3,2,1)
sns.countplot(data=data, x="country", hue="churn")

plt.subplot(3,2,2)
sns.countplot(data=data, x="gender", hue="churn")

plt.subplot(3,2,3)
sns.countplot(data=data, x="credit_card", hue="churn")

plt.subplot(3,2,4)
sns.countplot(data=data, x="active_member", hue="churn")
```

[19]:
```
<AxesSubplot:xlabel='active_member', ylabel='count'>
```



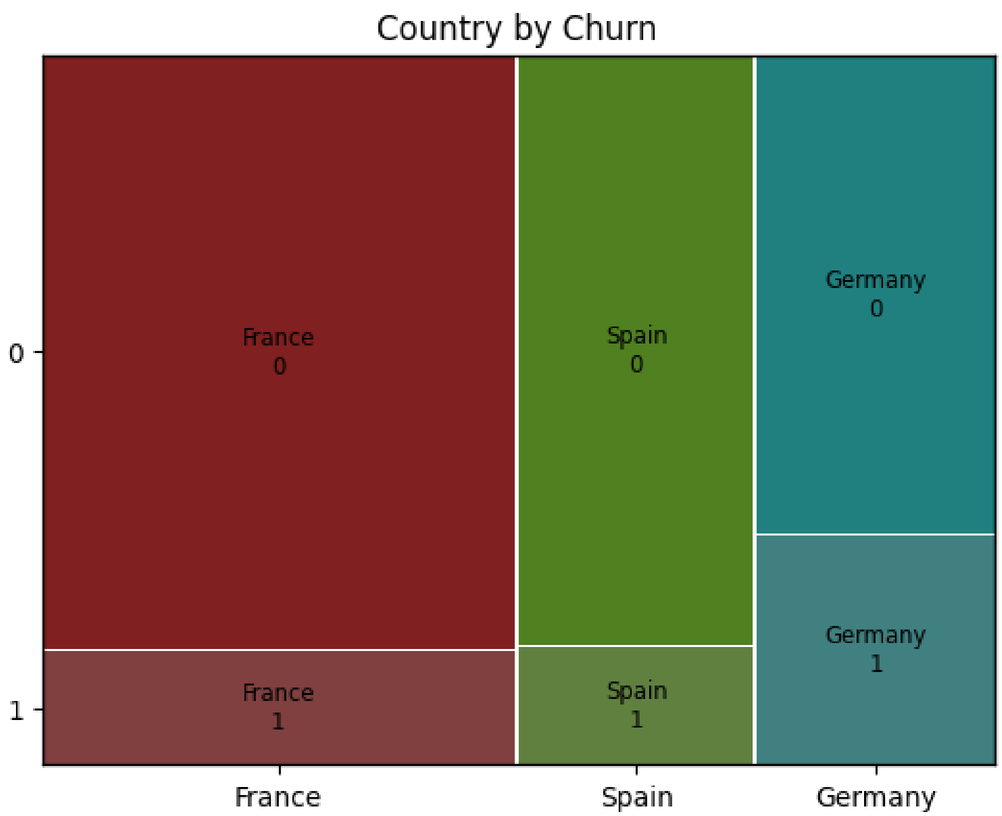
[20]:
```python
crosstable = pd.crosstab(data['churn'], data['gender'])
crosstable
```

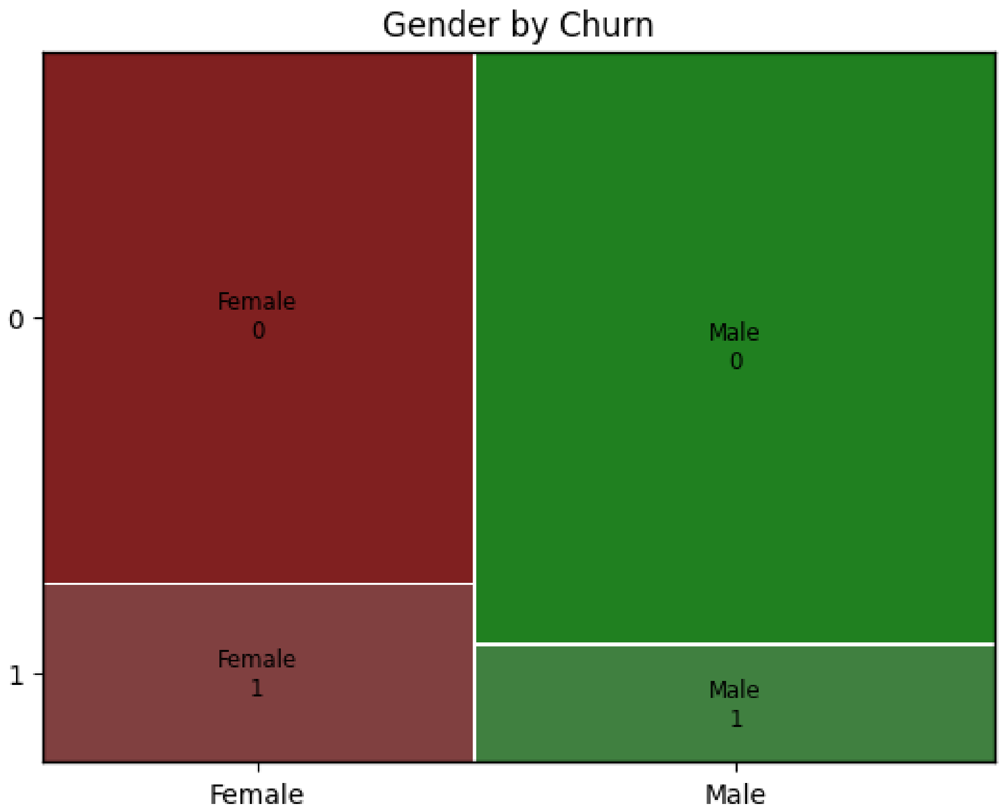
[20]:
```
gender   Female  Male
churn
0          3404  4559
1          1139   898
```

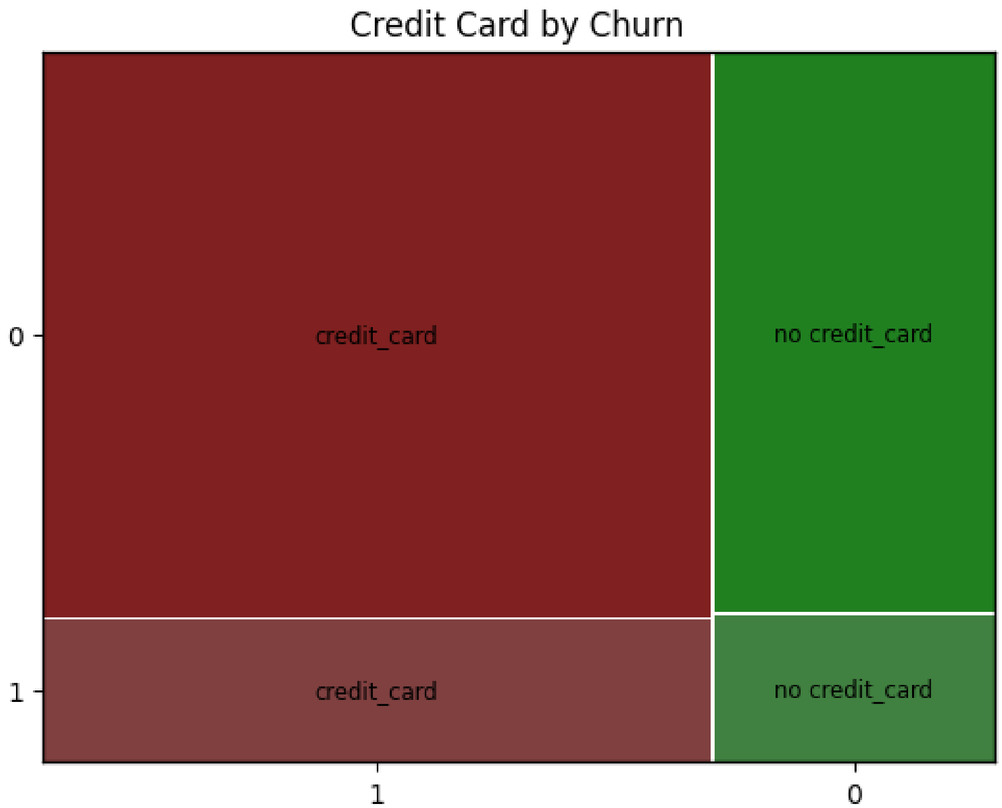
[21]:
```python
mosaic(data, ['country', 'churn'], title="Country by Churn")
mosaic(data, ['gender', 'churn'], title="Gender by Churn")
labelizer = lambda k: {('0','0'): 'no credit_card', ('0','1'): 'no␣
  ↪credit_card', ('1','0'): 'credit_card',('1','1'): 'credit_card'}[k]
mosaic(data, ['credit_card', 'churn'], labelizer =labelizer, title="Credit Card␣
  ↪by Churn")
labels = lambda k: {('0','0'): 'non-active member', ('0','1'): 'non-active␣
  ↪member', ('1','0'): 'active_member',('1','1'): 'active_member'}[k]
mosaic(data, ['active_member', 'churn'], labelizer=labels, title="Active Member␣
  ↪by Churn")
```

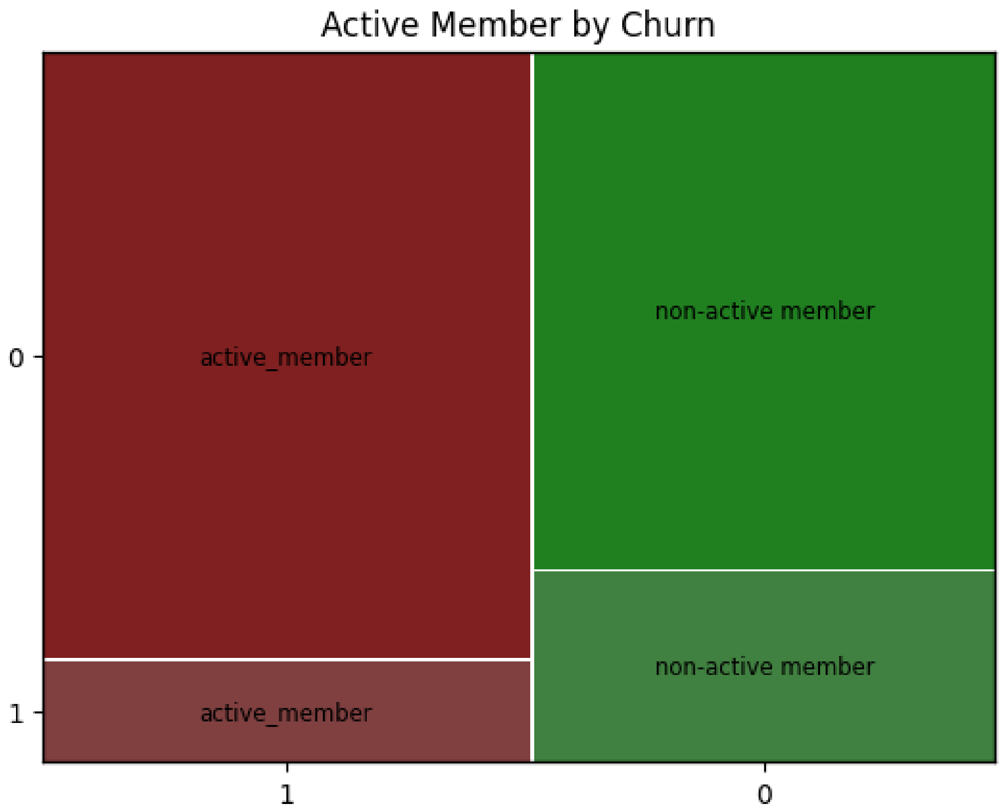
[21]:
```
(<Figure size 640x480 with 3 Axes>,
 {('1', '1'): (0.0, 0.0, 0.5125373134328359, 0.14221668404870583),
  ('1', '0'): (0.0,
   0.14553894318491845,
   0.5125373134328359,
```

```
 0.8544610568150816),
('0', '1'): (0.5175124378109454,
 0.0,
 0.48248756218905475,
 0.2676169145400394),
('0', '0'): (0.5175124378109454,
 0.27093917367625203,
 0.48248756218905475,
 0.7290608263237479)})
```




Gender by Churn
Female
0
Male
0
Female
1
Male
1
0
1
Female
Male


Credit Card by Churn
credit_card
no credit_card
credit_card
no credit_card
0
1
1
0



[22]:
```
g_one_hot = pd.get_dummies(data['gender'])
g_one_hot.head()
```

[22]:

	Female	Male
0	1	0
1	1	0
2	1	0
3	1	0
4	1	0

[23]:
```
data = data.drop('gender',axis = 1)
# Join the encoded df
data = data.join(g_one_hot)
data.head()
```

[23]:

	credit_score	country	age	tenure	balance	products_number	credit_card \
0	619	France	42	2	0.00	1	1
1	608	Spain	41	1	83807.86	1	0
2	502	France	42	8	159660.80	3	1
3	699	France	39	1	0.00	2	0
4	850	Spain	43	2	125510.82	1	1

```
   active_member  estimated_salary  churn  Female  Male
0              1         101348.88      1       1     0
1              1         112542.58      0       1     0
2              0         113931.57      1       1     0
3              0          93826.63      0       1     0
4              1          79084.10      0       1     0
```

[24]:
```python
c_one_hot = pd.get_dummies(data['country'])
#c_one_hot.head()
```

[25]:
```python
data2 = data.drop('country',axis = 1)
data2 = data2.join(c_one_hot)
#data2.head()
```

[26]:
```python
data2['zero_balance'] = np.where(data2['balance'] == 0.0, 1, 0) #leave this for␣
↪later on!
```

[27]:
```python
#data2.head()
```

[28]:
```python
data2 = data2.drop('balance',axis = 1)
```

[29]:
```python
first_column = data2.pop('churn')
data2.insert(0, 'churn', first_column)
```

[30]:
```python
data2.head()
```

[30]:
```
   churn  credit_score  age  tenure  products_number  credit_card  \
0      1           619   42       2                1            1
1      0           608   41       1                1            0
2      1           502   42       8                3            1
3      0           699   39       1                2            0
4      0           850   43       2                1            1

   active_member  estimated_salary  Female  Male  France  Germany  Spain  \
0              1         101348.88       1     0       1        0      0
1              1         112542.58       1     0       0        0      1
2              0         113931.57       1     0       1        0      0
3              0          93826.63       1     0       1        0      0
4              1          79084.10       1     0       0        0      1

   zero_balance
0             1
1             0
2             0
3             1
4             0
```

[31]:
```
data2.dtypes
```

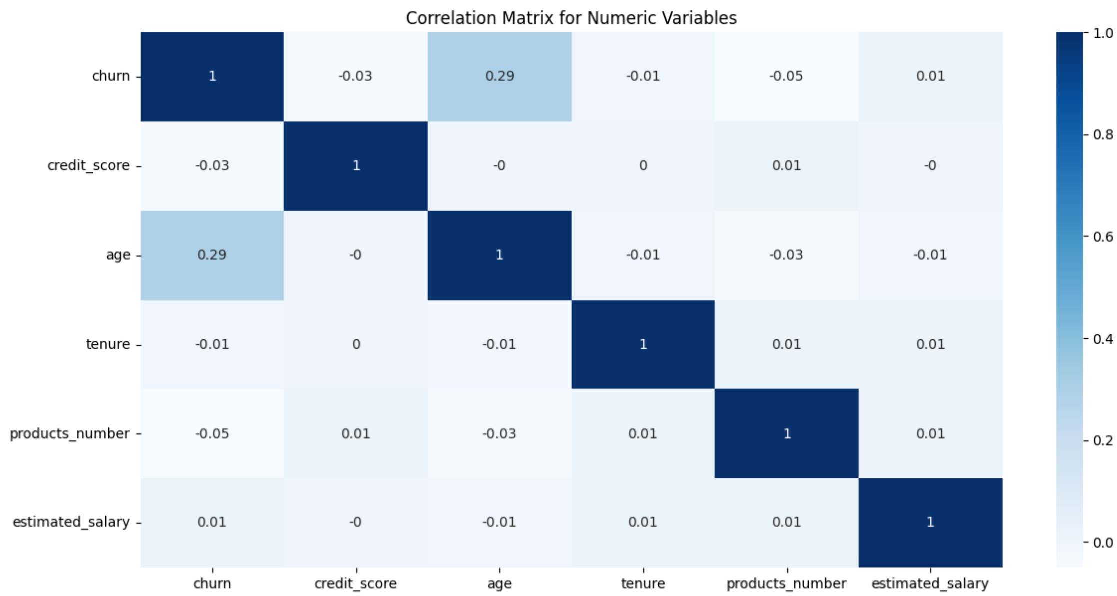
[31]:
```
churn                  int64
credit_score           int64
age                    int64
tenure                 int64
products_number        int64
credit_card            int64
active_member          int64
estimated_salary     float64
Female                 uint8
Male                   uint8
France                 uint8
Germany                uint8
Spain                  uint8
zero_balance           int64
dtype: object
```

[32]:
```
numeric_vars = data2[['churn', 'credit_score', 'age',␣
 ↪'tenure','products_number', 'estimated_salary']]
```

[33]:
```
corr_matrix = numeric_vars.corr().round(2)
print(corr_matrix)
```

```
                  churn  credit_score   age  tenure  products_number  \
churn              1.00         -0.03  0.29   -0.01            -0.05
credit_score      -0.03          1.00 -0.00    0.00             0.01
age                0.29         -0.00  1.00   -0.01            -0.03
tenure            -0.01          0.00 -0.01    1.00             0.01
products_number   -0.05          0.01 -0.03    0.01             1.00
estimated_salary   0.01         -0.00 -0.01    0.01             0.01

                  estimated_salary
churn                         0.01
credit_score                 -0.00
age                          -0.01
tenure                        0.01
products_number               0.01
estimated_salary              1.00
```

[34]:
```
plt.figure(figsize = (14,7))
sns.heatmap(corr_matrix, annot=True, cmap='Blues')
plt.title(label="Correlation Matrix for Numeric Variables")
plt.show()
```



Variable Selection

[35]:
```
data2.head()
```

[35]:
```
   churn  credit_score  age  tenure  products_number  credit_card  \
0      1           619   42       2                1            1
1      0           608   41       1                1            0
2      1           502   42       8                3            1
3      0           699   39       1                2            0
4      0           850   43       2                1            1

   active_member  estimated_salary  Female  Male  France  Germany  Spain  \
0              1         101348.88       1     0       1        0      0
1              1         112542.58       1     0       0        0      1
2              0         113931.57       1     0       1        0      0
3              0          93826.63       1     0       1        0      0
4              1          79084.10       1     0       0        0      1

   zero_balance
0             1
1             0
2             0
3             1
4             0
```

[36]:
```
features = data2.columns[1:14]
target = data2.columns[0]
X = data2[features].values
```

```
y = data2[target].values
```

[37]:
```
print(X)
```

```
[[619.  42.   2. …   0.   0.   1.]
 [608.  41.   1. …   0.   1.   0.]
 [502.  42.   8. …   0.   0.   0.]
 …
 [709.  36.   7. …   0.   0.   1.]
 [772.  42.   3. …   1.   0.   0.]
 [792.  28.   4. …   0.   0.   0.]]
```

[38]:
```
print(y)
```

```
[1 0 1 … 1 1 0]
```

[39]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=4)
```

[40]:
```
np.unique(y_train, return_counts=True)
```

[40]:
```
(array([0, 1]), array([6359, 1641]))
```

[41]:
```
np.unique(y_test, return_counts=True)
```

[41]:
```
(array([0, 1]), array([1604,  396]))
```

[42]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[43]:
```
lasso_glm = LogisticRegressionCV(Cs = [0.001, 0.005, 0.0075, 0.01, .05, .075, .
 ↪1, .5, .75, 1], cv=10, penalty='l1', solver="liblinear", random_state=2).
 ↪fit(X_train_scaled, y_train)
```

[44]:
```
lasso_glm.C_
```

[44]:
```
array([0.05])
```

[45]:
```
cs = lasso_glm.Cs_
print(cs)
```

```
[0.001  0.005  0.0075 0.01   0.05   0.075  0.1    0.5    0.75   1.    ]
```

[46]:
```
scs = lasso_glm.scores_[1]
scs
```

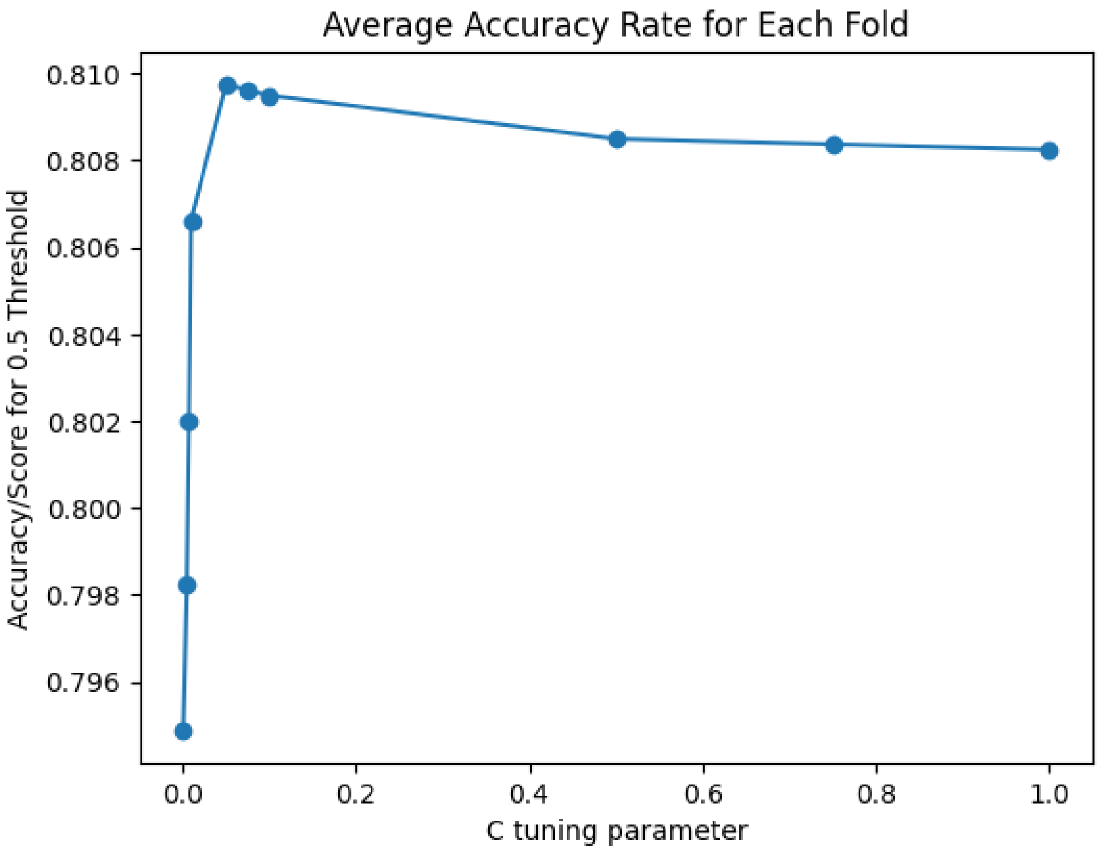
[46]: array([[0.795   , 0.79375, 0.79625, 0.80125, 0.80625, 0.80625, 0.80625,
         0.80375, 0.80375, 0.80375],
        [0.795   , 0.80125, 0.80125, 0.81125, 0.81   , 0.81   , 0.81   ,
         0.8125 , 0.81125, 0.81125],
        [0.795   , 0.80625, 0.805   , 0.80875, 0.82   , 0.82125, 0.82   ,
         0.8175 , 0.8175 , 0.8175],
        [0.795   , 0.8025 , 0.8075 , 0.8075 , 0.80625, 0.805   , 0.80625,
         0.8025 , 0.8025 , 0.80125],
        [0.795   , 0.80125, 0.80375, 0.81   , 0.815   , 0.815   , 0.81625,
         0.815 , 0.815 , 0.815],
        [0.795   , 0.79625, 0.8025 , 0.80625, 0.81125, 0.81   , 0.81   ,
         0.80875, 0.80875, 0.80875],
        [0.795   , 0.79125, 0.795   , 0.80125, 0.80625, 0.80375, 0.80375,
         0.80375, 0.80375, 0.80375],
        [0.795   , 0.8   , 0.805   , 0.81   , 0.8075 , 0.80875, 0.8075 ,
         0.81   , 0.81   , 0.81   ],
        [0.795   , 0.795   , 0.79875, 0.80375, 0.80375, 0.80375, 0.8025 ,
         0.80125, 0.80125, 0.80125],
        [0.79375, 0.795   , 0.805   , 0.80625, 0.81125, 0.8125 , 0.8125 ,
         0.81   , 0.81   , 0.81   ]])

[47]:
```
scores = np.mean(scs, axis=0)
scores
```

[47]: array([0.794875, 0.79825 , 0.802   , 0.806625, 0.80975 , 0.809625,
       0.8095 , 0.8085 , 0.808375, 0.80825])

[48]:
```
plt.plot(cs, scores, "-o") #Inverse of regularization strength; must be a␣
 ↪positive float. Smaller values specify stronger regularization.
plt.title("Average Accuracy Rate for Each Fold")
plt.xlabel("C tuning parameter")
plt.ylabel('Accuracy/Score for 0.5 Threshold')
```

[48]: Text(0, 0.5, 'Accuracy/Score for 0.5 Threshold')

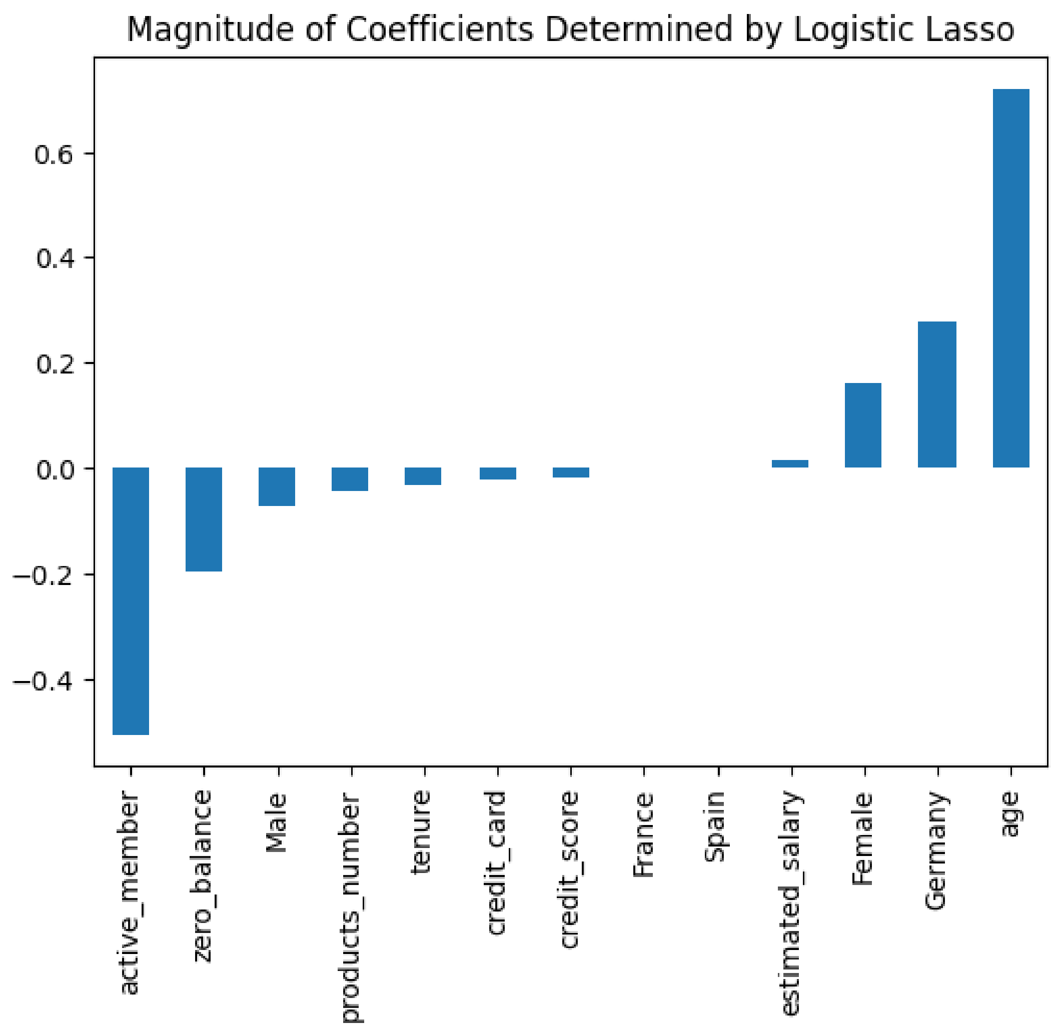


[49]:
```
lasso_glm.coef_
```

[49]:
```
array([[-0.01814378,  0.72043614, -0.03486288, -0.04349273, -0.0227542 ,
        -0.50621947,  0.01562966,  0.15976542, -0.07163175, -0.00107001,
         0.27854284,  0.        , -0.19793599]])
```

[50]:
```
pd.Series(lasso_glm.coef_[0], features).sort_values(ascending = True).plot(kind␣
 ↪= "bar")
plt.title("Magnitude of Coefficients Determined by Logistic Lasso")
```

[50]:
```
Text(0.5, 1.0, 'Magnitude of Coefficients Determined by Logistic Lasso')
```



Modeling

Logistic Regression

[51]:
```
#13 features: remove : 9, 10, 12 (one-indexed, correct for zero based)
```

[52]:
```
print(X_train)
X_train.shape
```

```
[[543.  30.   4. …   0.   0.   0.]
 [668.  46.   0. …   0.   0.   1.]
 [767.  35.   6. …   1.   0.   0.]
 …
 [686.  34.   3. …   0.   0.   0.]
 [637.  41.   2. …   0.   0.   1.]
 [614.  30.   3. …   1.   0.   0.]]
```

[52]: (8000, 13)

[53]:
```
X_train_mod = np.delete(X_train, [8,9,11], axis=1)
```

[54]:
```
X_train_mod.shape
```

[54]: (8000, 10)

[55]:
```
print(X_train_mod)
```

```
[[543.  30.   4. …   0.   0.   0.]
 [668.  46.   0. …   0.   0.   1.]
 [767.  35.   6. …   1.   1.   0.]
 …
 [686.  34.   3. …   1.   0.   0.]
 [637.  41.   2. …   0.   0.   1.]
 [614.  30.   3. …   1.   1.   0.]]
```

[56]:
```
X_test_mod = np.delete(X_test, [8,9,11], axis=1)
```

[57]:
```
X_test_mod.shape #dropped irrelevant vars, but did not scale
```

[57]: (2000, 10)

[58]:
```
log = LogisticRegression(random_state=0).fit(X_train_mod, y_train)
```

[59]:
```
log.coef_
```

[59]:
```
array([[-4.56200491e-03,  4.35590335e-02, -1.56321220e-03,
        -7.11305216e-04, -1.99925624e-04, -1.37854543e-03,
        -1.24710672e-06,  8.20365203e-04,  1.17576203e-03,
        -1.12528654e-03]])
```

[60]:
```
log.score(X_train_mod, y_train)
```

[60]: 0.786375

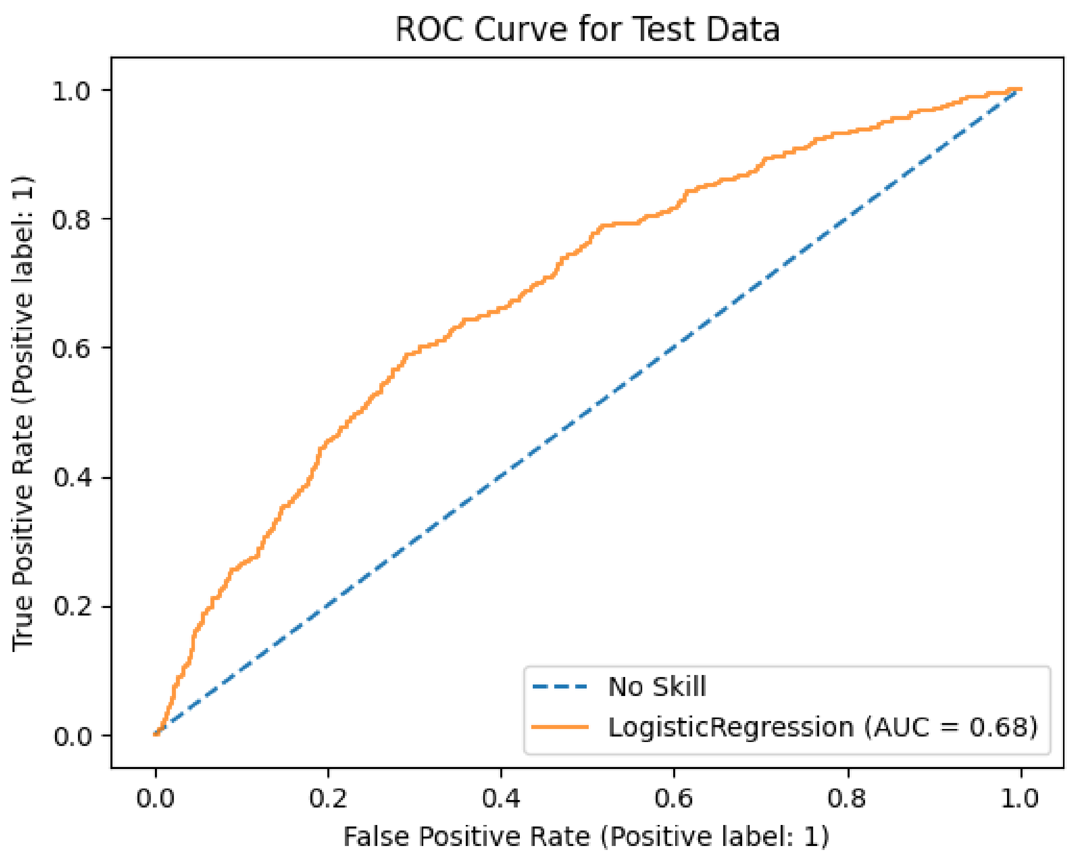
[61]:
```
log.score(X_test_mod, y_test) #accuracy for test data
```

[61]: 0.7965

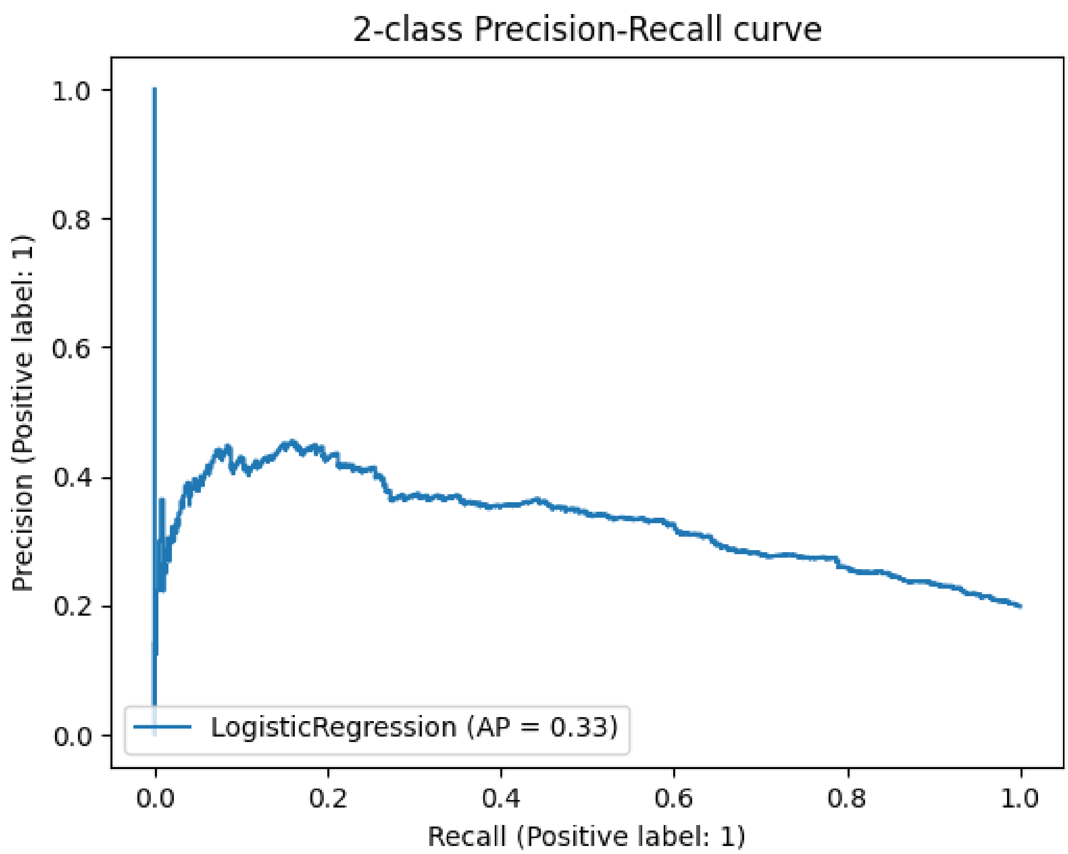
[62]:
```
#Citation: https://stackoverflow.com/questions/28716241/
↪controlling-the-threshold-in-logistic-regression-in-scikit-learn
```

[63]:
```
ax = plt.gca()
plt.plot([0, 1], [0, 1], linestyle="--", label='No Skill')
```

```
log_disp = RocCurveDisplay.from_estimator(log, X_test_mod, y_test, ax=ax,␣
  ↪alpha=0.8)
plt.title("ROC Curve for Test Data")
plt.show()
```



[64]:
```
probs_y=log.predict_proba(X_test_mod)

display = PrecisionRecallDisplay.from_predictions(y_test, probs_y[:,1],␣
  ↪name="LogisticRegression")
_ = display.ax_.set_title("2-class Precision-Recall curve")
```



[65]:
```
#https://machinelearningmastery.com/
↪roc-curves-and-precision-recall-curves-for-classification-in-python/

#info on interpretation of PRCs
```

[66]:
```
pred_proba = log.predict_proba(X_test_mod)[:,1]
#print(pred_proba)
threshold_list = [0.05,0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45,0.5,0.55,0.6,0.65,.
↪7,.75,.8,.85,.9,.95]
for i in threshold_list:
    print ('\n******** For i = {} ******'.format(i))
    y_test_pred = np.where(pred_proba > i, 1, 0)
    test_accuracy = accuracy_score(y_test, y_test_pred)
    print('Accuracy/Score is {}'.format(test_accuracy))
    print(confusion_matrix(y_test, y_test_pred))
    print(classification_report(y_test, y_test_pred, zero_division=1))
```

```
******** For i = 0.05 ******
Accuracy/Score is 0.1995
```

```
[[   3 1601]
 [   0  396]]
              precision    recall  f1-score   support

           0       1.00      0.00      0.00      1604
           1       0.20      1.00      0.33       396

    accuracy                           0.20      2000
   macro avg       0.60      0.50      0.17      2000
weighted avg       0.84      0.20      0.07      2000


******** For i = 0.1 ******
Accuracy/Score is 0.291
[[ 200 1404]
 [  14  382]]
              precision    recall  f1-score   support

           0       0.93      0.12      0.22      1604
           1       0.21      0.96      0.35       396

    accuracy                           0.29      2000
   macro avg       0.57      0.54      0.29      2000
weighted avg       0.79      0.29      0.25      2000


******** For i = 0.15 ******
Accuracy/Score is 0.446
[[ 552 1052]
 [  56  340]]
              precision    recall  f1-score   support

           0       0.91      0.34      0.50      1604
           1       0.24      0.86      0.38       396

    accuracy                           0.45      2000
   macro avg       0.58      0.60      0.44      2000
weighted avg       0.78      0.45      0.48      2000


******** For i = 0.2 ******
Accuracy/Score is 0.598
[[930 674]
 [130 266]]
              precision    recall  f1-score   support

           0       0.88      0.58      0.70      1604
           1       0.28      0.67      0.40       396
```

```
    accuracy                           0.60      2000
   macro avg       0.58      0.63      0.55      2000
weighted avg       0.76      0.60      0.64      2000


******** For i = 0.25 ******
Accuracy/Score is 0.7045
[[1203  401]
 [ 190  206]]
              precision    recall  f1-score   support

           0       0.86      0.75      0.80      1604
           1       0.34      0.52      0.41       396

    accuracy                           0.70      2000
   macro avg       0.60      0.64      0.61      2000
weighted avg       0.76      0.70      0.73      2000


******** For i = 0.3 ******
Accuracy/Score is 0.7515
[[1363  241]
 [ 256  140]]
              precision    recall  f1-score   support

           0       0.84      0.85      0.85      1604
           1       0.37      0.35      0.36       396

    accuracy                           0.75      2000
   macro avg       0.60      0.60      0.60      2000
weighted avg       0.75      0.75      0.75      2000


******** For i = 0.35 ******
Accuracy/Score is 0.784
[[1480  124]
 [ 308   88]]
              precision    recall  f1-score   support

           0       0.83      0.92      0.87      1604
           1       0.42      0.22      0.29       396

    accuracy                           0.78      2000
   macro avg       0.62      0.57      0.58      2000
weighted avg       0.75      0.78      0.76      2000
```

```
******** For i = 0.4 ******
Accuracy/Score is 0.793
[[1536   68]
 [ 346   50]]
              precision    recall  f1-score   support

           0       0.82      0.96      0.88      1604
           1       0.42      0.13      0.19       396

    accuracy                           0.79      2000
   macro avg       0.62      0.54      0.54      2000
weighted avg       0.74      0.79      0.75      2000


******** For i = 0.45 ******
Accuracy/Score is 0.7965
[[1562   42]
 [ 365   31]]
              precision    recall  f1-score   support

           0       0.81      0.97      0.88      1604
           1       0.42      0.08      0.13       396

    accuracy                           0.80      2000
   macro avg       0.62      0.53      0.51      2000
weighted avg       0.73      0.80      0.74      2000


******** For i = 0.5 ******
Accuracy/Score is 0.7965
[[1577   27]
 [ 380   16]]
              precision    recall  f1-score   support

           0       0.81      0.98      0.89      1604
           1       0.37      0.04      0.07       396

    accuracy                           0.80      2000
   macro avg       0.59      0.51      0.48      2000
weighted avg       0.72      0.80      0.72      2000


******** For i = 0.55 ******
Accuracy/Score is 0.797
[[1587   17]
 [ 389    7]]
              precision    recall  f1-score   support
```

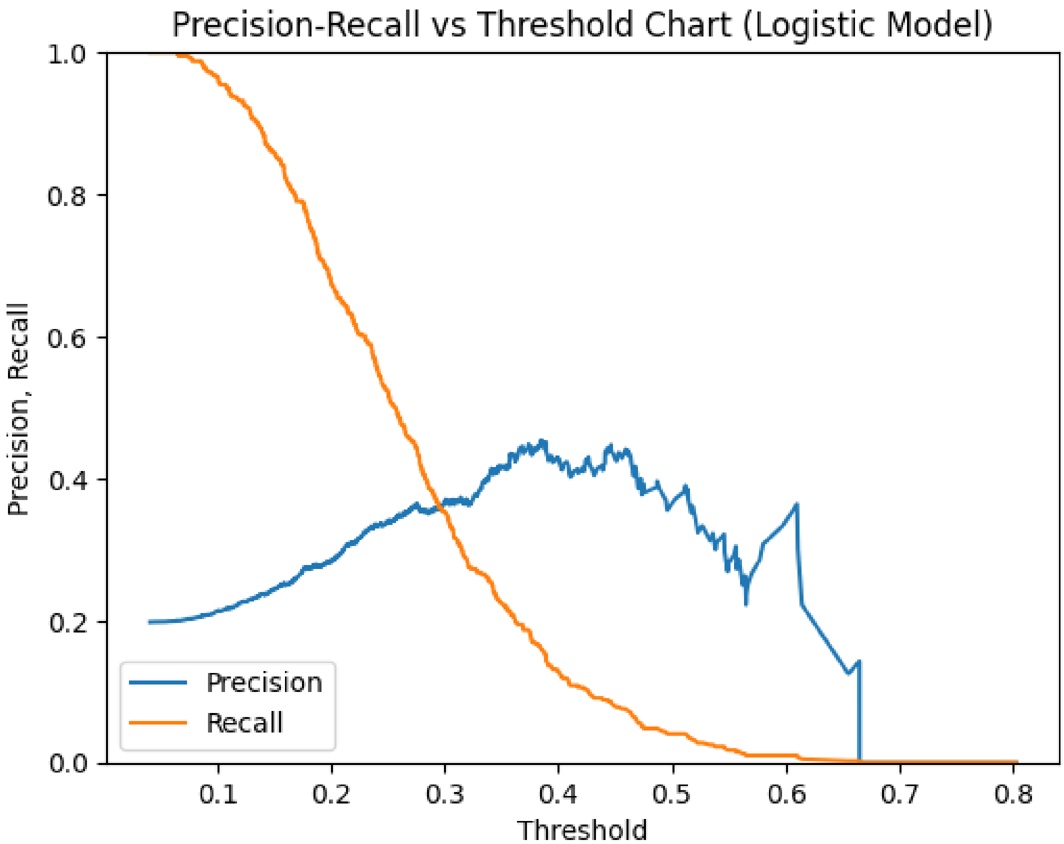
```
           0       0.80      0.99      0.89      1604
           1       0.29      0.02      0.03       396

    accuracy                           0.80      2000
   macro avg       0.55      0.50      0.46      2000
weighted avg       0.70      0.80      0.72      2000


******** For i = 0.6 ******
Accuracy/Score is 0.8005
[[1597    7]
 [ 392    4]]
              precision    recall  f1-score   support

           0       0.80      1.00      0.89      1604
           1       0.36      0.01      0.02       396

    accuracy                           0.80      2000
   macro avg       0.58      0.50      0.45      2000
weighted avg       0.72      0.80      0.72      2000


******** For i = 0.65 ******
Accuracy/Score is 0.799
[[1597    7]
 [ 395    1]]
              precision    recall  f1-score   support

           0       0.80      1.00      0.89      1604
           1       0.12      0.00      0.00       396

    accuracy                           0.80      2000
   macro avg       0.46      0.50      0.45      2000
weighted avg       0.67      0.80      0.71      2000


******** For i = 0.7 ******
Accuracy/Score is 0.801
[[1602    2]
 [ 396    0]]
              precision    recall  f1-score   support

           0       0.80      1.00      0.89      1604
           1       0.00      0.00      0.00       396

    accuracy                           0.80      2000
   macro avg       0.40      0.50      0.44      2000
weighted avg       0.64      0.80      0.71      2000
```

```
******** For i = 0.75 ******
Accuracy/Score is 0.8015
[[1603    1]
 [ 396    0]]
              precision    recall  f1-score   support

           0       0.80      1.00      0.89      1604
           1       0.00      0.00      0.00       396

    accuracy                           0.80      2000
   macro avg       0.40      0.50      0.44      2000
weighted avg       0.64      0.80      0.71      2000


******** For i = 0.8 ******
Accuracy/Score is 0.8015
[[1603    1]
 [ 396    0]]
              precision    recall  f1-score   support

           0       0.80      1.00      0.89      1604
           1       0.00      0.00      0.00       396

    accuracy                           0.80      2000
   macro avg       0.40      0.50      0.44      2000
weighted avg       0.64      0.80      0.71      2000


******** For i = 0.85 ******
Accuracy/Score is 0.802
[[1604    0]
 [ 396    0]]
              precision    recall  f1-score   support

           0       0.80      1.00      0.89      1604
           1       1.00      0.00      0.00       396

    accuracy                           0.80      2000
   macro avg       0.90      0.50      0.45      2000
weighted avg       0.84      0.80      0.71      2000


******** For i = 0.9 ******
Accuracy/Score is 0.802
[[1604    0]
 [ 396    0]]
```

```
              precision    recall  f1-score   support

           0       0.80      1.00      0.89      1604
           1       1.00      0.00      0.00       396

    accuracy                           0.80      2000
   macro avg       0.90      0.50      0.45      2000
weighted avg       0.84      0.80      0.71      2000


******** For i = 0.95 ******
Accuracy/Score is 0.802
[[1604    0]
 [ 396    0]]
              precision    recall  f1-score   support

           0       0.80      1.00      0.89      1604
           1       1.00      0.00      0.00       396

    accuracy                           0.80      2000
   macro avg       0.90      0.50      0.45      2000
weighted avg       0.84      0.80      0.71      2000
```

[67]:
```
#pred_y=log.predict(X_test_mod)

probs_y=log.predict_proba(X_test_mod)
  # probs_y is a 2-D array of probability of being labeled as 0 (first
  #column of
  #array) vs 1 (2nd column in array)

precision, recall, thresholds = precision_recall_curve(y_test, probs_y[:,
1])
   #retrieve probability of being 1(in second column of probs_y)
pr_auc = auc(recall, precision)

plt.title("Precision-Recall vs Threshold Chart (Logistic Model)")
plt.plot(thresholds, precision[: -1], label="Precision")
plt.plot(thresholds, recall[: -1], label="Recall")
plt.ylabel("Precision, Recall")
plt.xlabel("Threshold")
plt.legend(loc="lower left")
plt.ylim([0,1])
```

[67]: (0.0, 1.0)



[68]:
```
#based off plot above, a threshold around 0.3 may produce the best prediction␣
↪performance
```

Random Forest

[69]:
```
# error_rates = [] #varying number of trees and number of features did not␣
↪appear to significantly impact RF performance

# num_trees = range(50,120)
# for i in num_trees:
#     rf = RandomForestClassifier(n_estimators = i, random_state=0)
#     rf = rf.fit(X_train_mod, y_train)
#     rf_pred = rf.predict(X_test_mod)
#     rf_accuracy = accuracy_score(y_test, rf_pred)
#     error_rates.append(rf_accuracy)

# print(error_rates)
```

[70]:
```
# plt.plot(num_trees, error_rates, '-o', label="RF Error Rates")
# plt.title("Accuracy Against Number of Trees")
```

```python
# plt.xlabel("RF Number of Trees")
# plt.ylabel('Accuracy')
# plt.grid()
# plt.legend(loc="upper right")
```

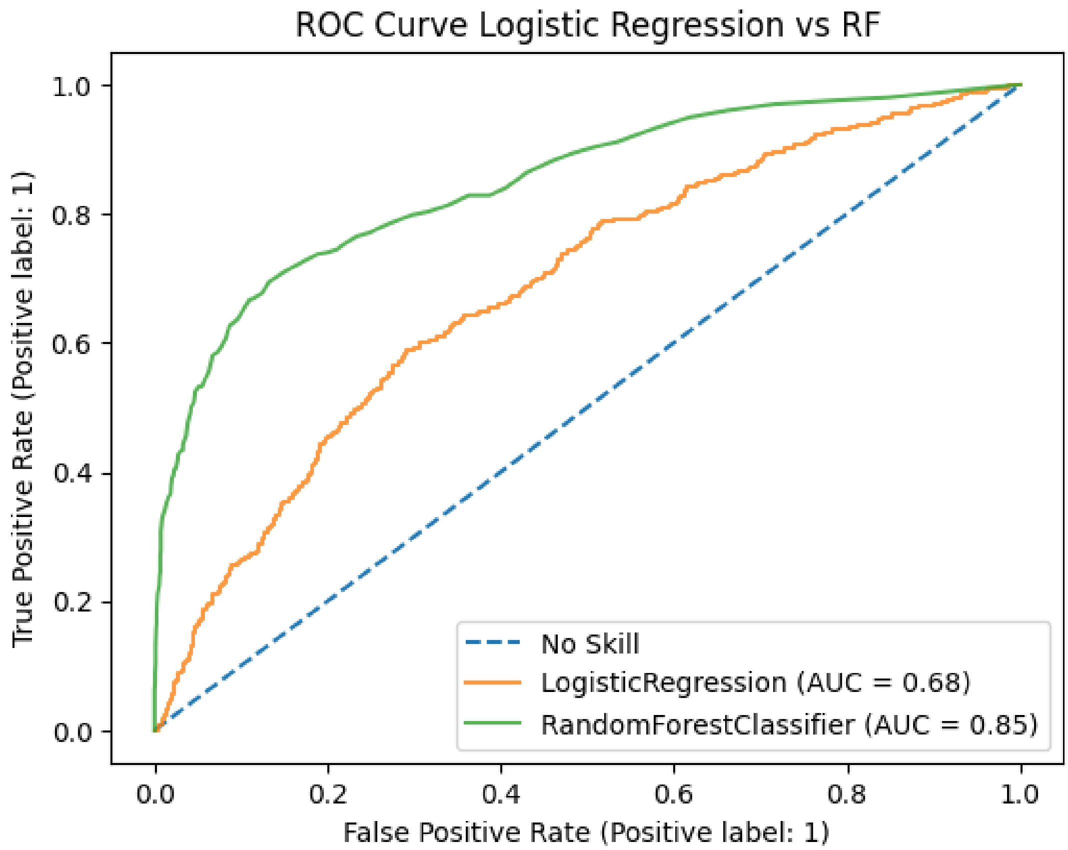
[71]:
```python
rf = RandomForestClassifier(random_state=0).fit(X_train_mod, y_train)
```

[72]:
```python
rf_preds = rf.predict(X_test_mod)
```

[73]:
```python
rf_accuracy = accuracy_score(y_test, rf_preds)
print('Accuracy/Score is {}'.format(rf_accuracy))
print(confusion_matrix(y_test, rf_preds))
print(classification_report(y_test, rf_preds, zero_division=1))
```

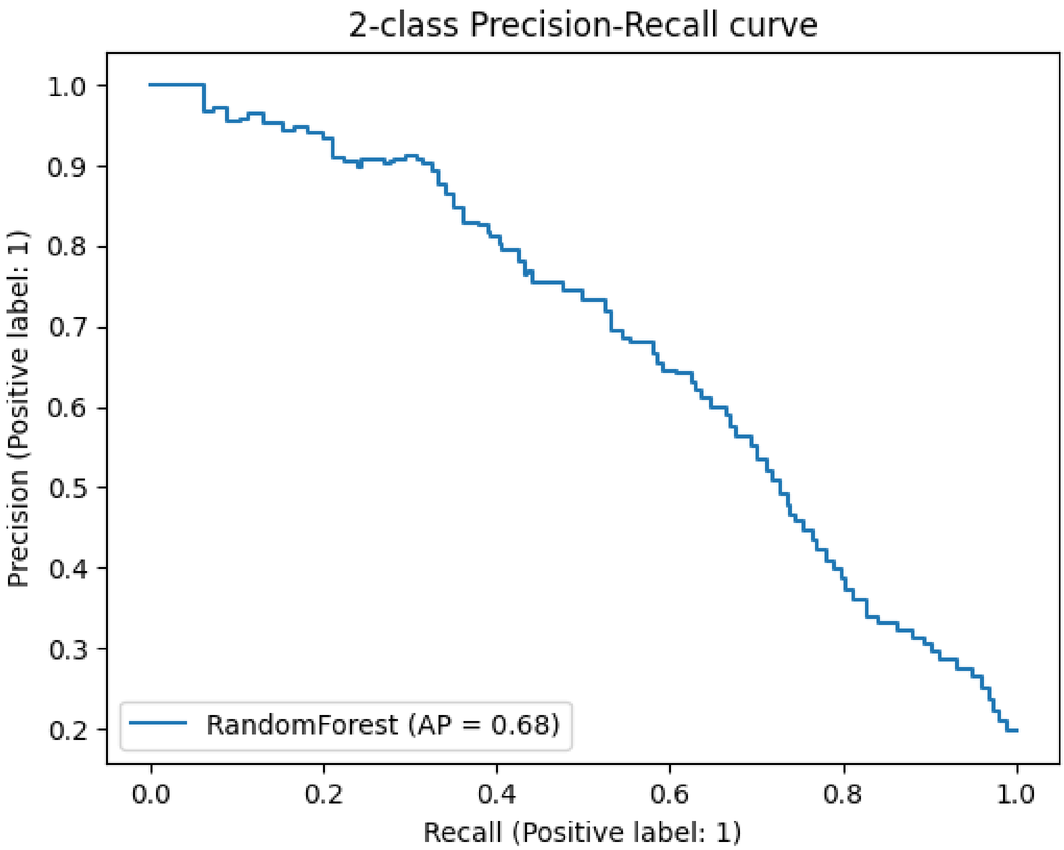
```
Accuracy/Score is 0.866
[[1537   67]
 [ 201  195]]
              precision    recall  f1-score   support

           0       0.88      0.96      0.92      1604
           1       0.74      0.49      0.59       396

    accuracy                           0.87      2000
   macro avg       0.81      0.73      0.76      2000
weighted avg       0.86      0.87      0.86      2000
```

[74]:
```python
ax = plt.gca()
plt.plot([0, 1], [0, 1], linestyle="--", label='No Skill')
log_disp.plot(ax=ax, alpha=0.8)
rfc_disp = RocCurveDisplay.from_estimator(rf, X_test_mod, y_test, ax=ax,␣
 ↪alpha=0.8)
plt.title("ROC Curve Logistic Regression vs RF")
plt.show()
```



[75]:

```
display = PrecisionRecallDisplay.from_estimator(
    rf, X_test_mod, y_test, name="RandomForest"
)
_ = display.ax_.set_title("2-class Precision-Recall curve")
```



[76]:
```
rf.feature_importances_
```

[76]:
```
array([0.18330255, 0.26450183, 0.09911398, 0.13126349, 0.02131865,
       0.04313341, 0.19025361, 0.02088831, 0.02481393, 0.02141024])
```

[77]:
```
#data2.columns.values.tolist()
```

[78]:
```
feature_names = [
 'credit_score',
 'age',
 'tenure',
 'products_number',
 'credit_card',
 'active_member',
 'estimated_salary',
 'Female',
 'Germany',
 'zero_balance']

feature_names = np.array(feature_names)
```

```
print(feature_names)
```

```
['credit_score' 'age' 'tenure' 'products_number' 'credit_card'
 'active_member' 'estimated_salary' 'Female' 'Germany' 'zero_balance']
```

[79]:
```
importances = rf.feature_importances_
print(importances)
important_names = feature_names[importances > np.mean(importances)]
print(important_names)
```

```
[0.18330255 0.26450183 0.09911398 0.13126349 0.02131865 0.04313341
 0.19025361 0.02088831 0.02481393 0.02141024]
['credit_score' 'age' 'products_number' 'estimated_salary']
```

[80]:
```
#https://towardsdatascience.com/
↪feature-selection-using-random-forest-26d7b747597f#:~:
↪text=The%20more%20a%20feature%20decreases,final%20importance%20of%20the%20variable.
↪
```

Refitting of Models

[81]:
```
X_train_mod2 = np.delete(X_train_mod, [2,4,5,7,8,9], axis=1) #delete 2, 4, 5,␣
↪7,8,9 (0 indexed)
```

[82]:
```
#print(X_train_mod2)
```

[83]:
```
X_test_mod2 = np.delete(X_test_mod, [2,4,5,7,8,9], axis=1)
```

[84]:
```
#print(X_test_mod2)
```

[85]:
```
rf2 = RandomForestClassifier(random_state=0).fit(X_train_mod2, y_train)
```

[86]:
```
rf2_preds = rf2.predict(X_test_mod2) #refitting RF and Logistic Reg to reduced␣
↪data did not improve performance
```

[87]:
```
rf_accuracy = accuracy_score(y_test, rf2_preds)
print('Accuracy/Score is {}'.format(rf_accuracy))
print(confusion_matrix(y_test, rf2_preds))
print(classification_report(y_test, rf2_preds, zero_division=1))
```

```
Accuracy/Score is 0.8305
[[1491  113]
 [ 226  170]]
              precision    recall  f1-score   support

           0       0.87      0.93      0.90      1604
           1       0.60      0.43      0.50       396
```

```
    accuracy                           0.83      2000
   macro avg       0.73      0.68      0.70      2000
weighted avg       0.82      0.83      0.82      2000
```

[88]:
```
#log2 = LogisticRegression(random_state=0).fit(X_train_mod2, y_train)
```

[89]:
```
#log2.score(X_test_mod2, y_test)
```

[90]:
```
rf3 = RandomForestClassifier(random_state=0).fit(X_train, y_train)
```

[91]:
```
rf3_preds = rf3.predict(X_test)
```

[92]:
```
rf_accuracy = accuracy_score(y_test, rf3_preds) #RF fit to vars chosen by Lasso still performs the best
print('Accuracy/Score is {}'.format(rf_accuracy))
print(confusion_matrix(y_test, rf3_preds))
print(classification_report(y_test, rf3_preds, zero_division=1))
```

```
Accuracy/Score is 0.8615
[[1542   62]
 [ 215  181]]
              precision    recall  f1-score   support

           0       0.88      0.96      0.92      1604
           1       0.74      0.46      0.57       396

    accuracy                           0.86      2000
   macro avg       0.81      0.71      0.74      2000
weighted avg       0.85      0.86      0.85      2000
```

KNN Classifier

[93]:
```
#distance-based algorithm, so requires scaling
```

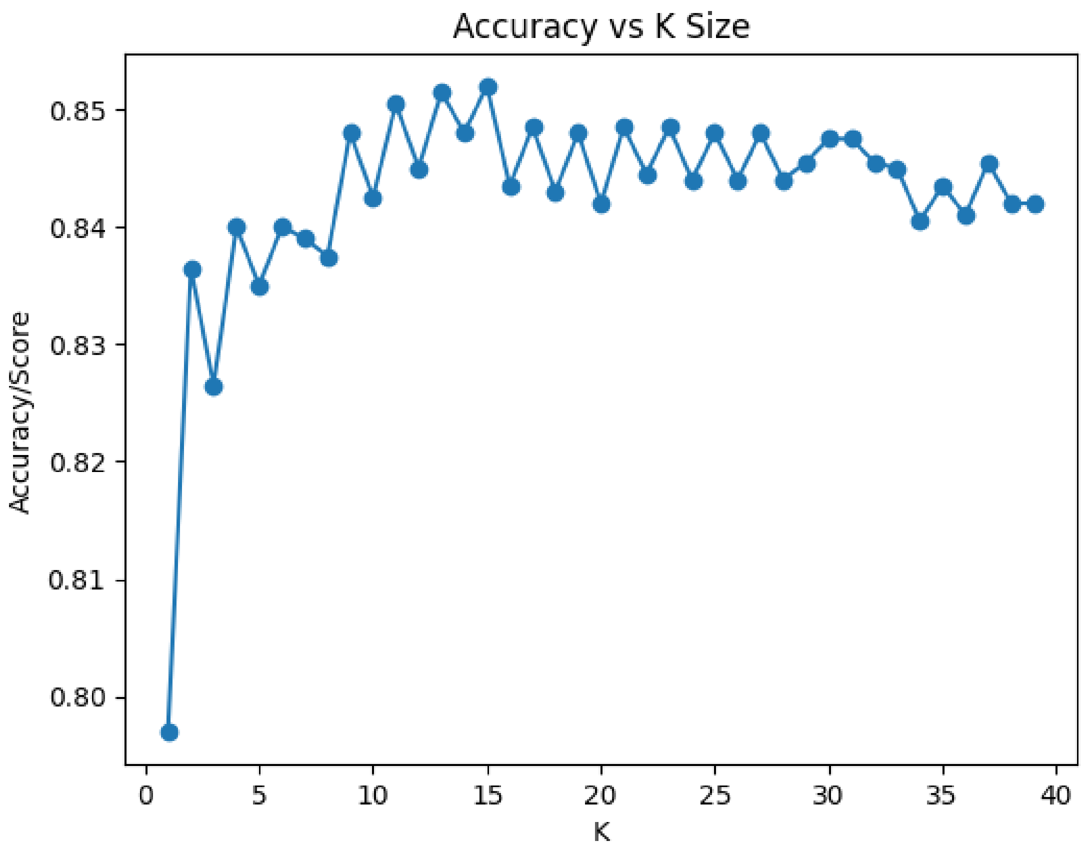
[94]:
```
scaler = StandardScaler()
X_train_mod_sc= scaler.fit_transform(X_train_mod)
X_test_mod_sc = scaler.transform(X_test_mod)
```

[95]:
```
ks = range(1, 40)
scores = []
recalls = []

for k in ks:
    knn = KNeighborsClassifier(n_neighbors=k).fit(X_train_mod_sc, y_train)
    preds = knn.predict(X_test_mod_sc)
    accuracy = knn.score(X_test_mod_sc, y_test)
```

```
    recall = classification_report(y_test, preds, zero_division=1,␣
↪output_dict=True)['1']['recall']
    scores.append(accuracy)
    recalls.append(recall)
```

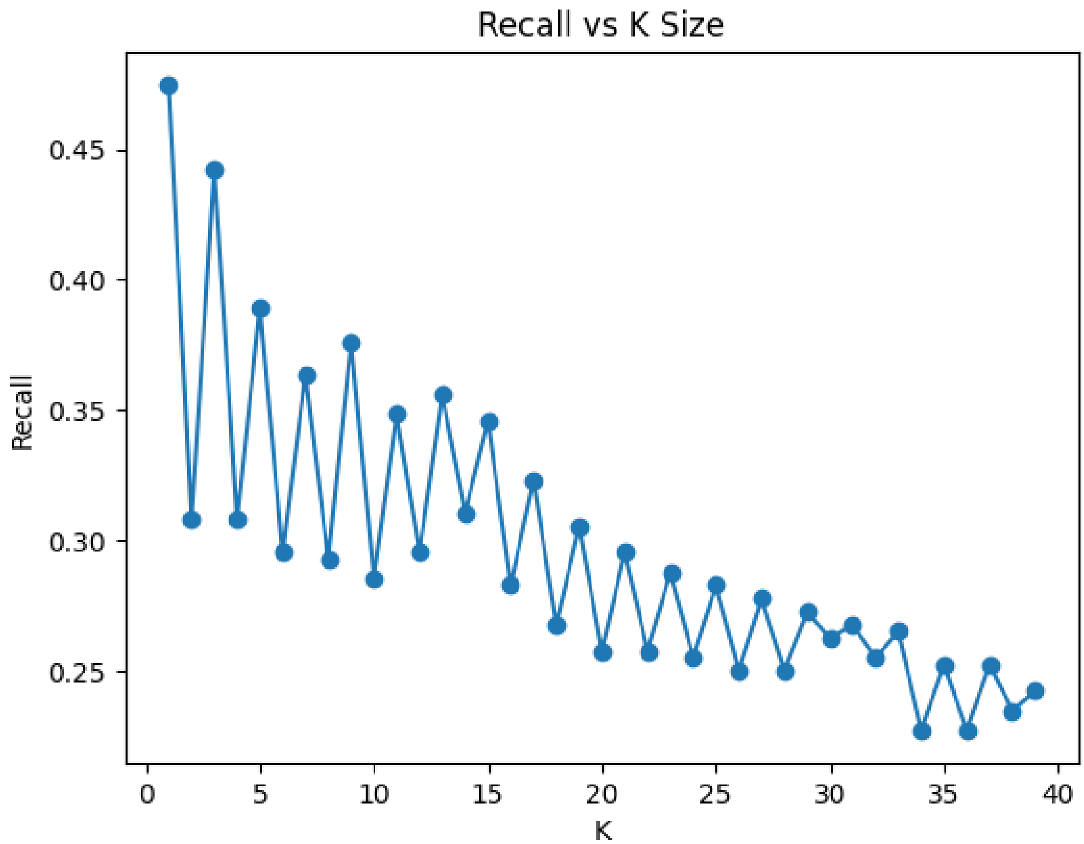
[96]:
```
plt.plot(ks, scores, "-o")
plt.title("Accuracy vs K Size")
plt.xlabel("K")
plt.ylabel('Accuracy/Score')
```

[96]:
```
Text(0, 0.5, 'Accuracy/Score')
```



[97]:
```
plt.plot(ks, recalls, "-o") #Inverse of regularization strength; must be a␣
↪positive float. Smaller values specify stronger regularization.
plt.title("Recall vs K Size")
plt.xlabel("K")
plt.ylabel('Recall')
```

[97]:
```
Text(0, 0.5, 'Recall')
```



[98]:
```
knn = KNeighborsClassifier(n_neighbors=15).fit(X_train_mod_sc, y_train) #best
 ↪accuracy at k=15; best recall for k=1
```

[99]:
```
knn.score(X_test_mod_sc, y_test)
```

[99]: 0.852

[100]:
```
knn_preds = knn.predict(X_test_mod_sc)
```

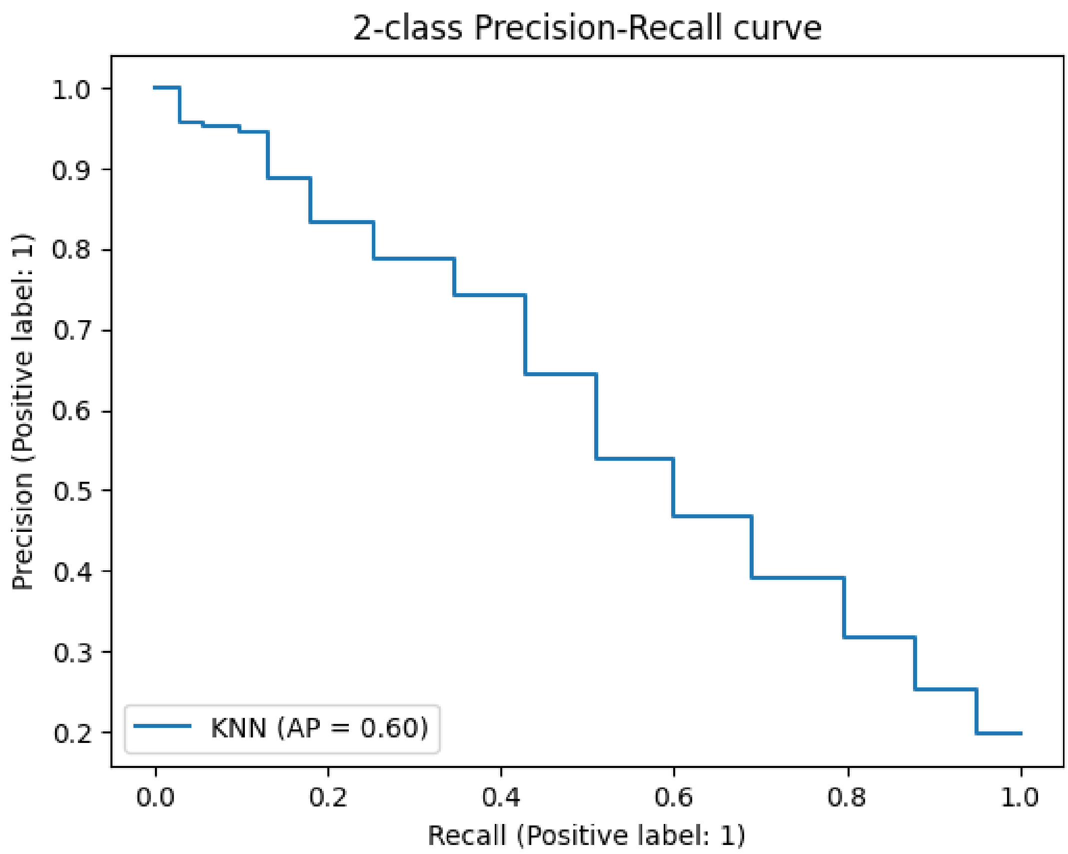
[101]:
```
knn_accuracy = accuracy_score(y_test, knn_preds) #RF fit to vars chosen by
 ↪Lasso still performs the best
print('Accuracy/Score is {}'.format(knn_accuracy))
print(confusion_matrix(y_test, knn_preds))
print(classification_report(y_test, knn_preds, zero_division=1))
```

```
Accuracy/Score is 0.852
[[1567   37]
 [ 259  137]]
              precision    recall  f1-score   support
```

```
           0       0.86      0.98      0.91      1604
           1       0.79      0.35      0.48       396

    accuracy                           0.85      2000
   macro avg       0.82      0.66      0.70      2000
weighted avg       0.84      0.85      0.83      2000
```

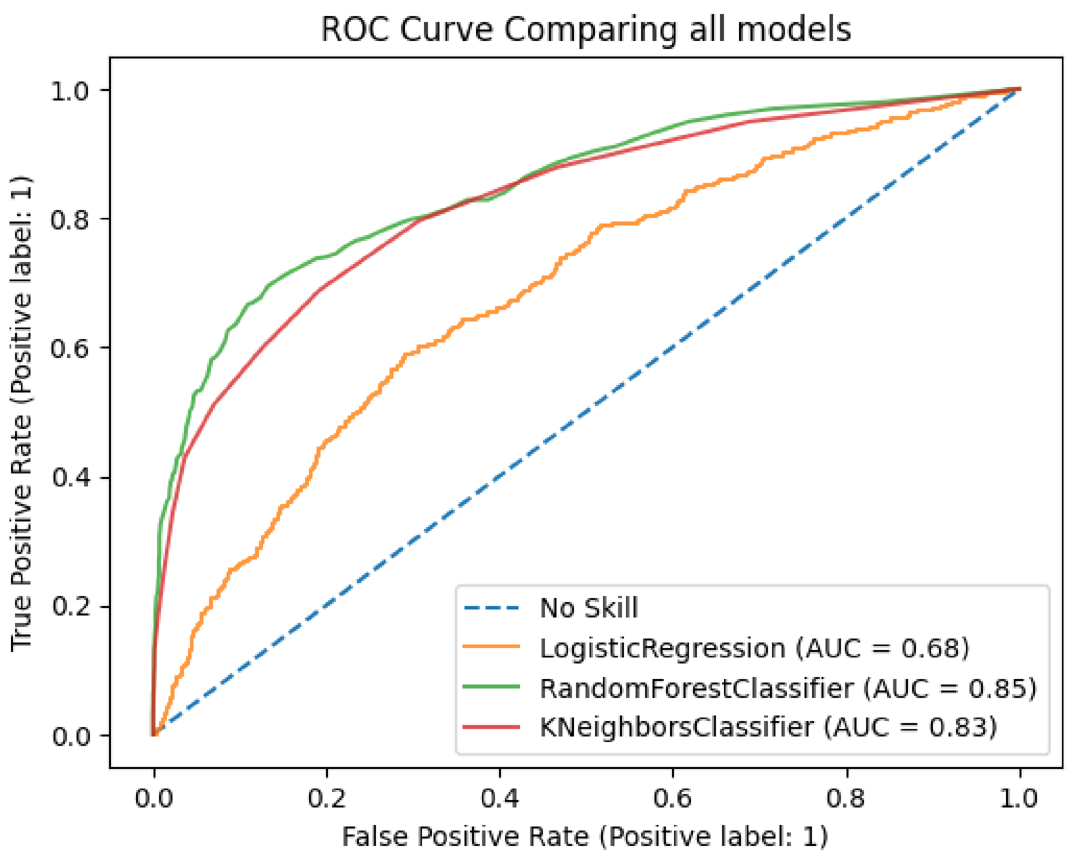
[102]:
```
#classification_report(y_test, knn_preds, zero_division=1,␣
 ↪output_dict=True)['1']['recall']
```

[103]:
```
display = PrecisionRecallDisplay.from_estimator(
    knn, X_test_mod_sc, y_test, name="KNN"
)
_ = display.ax_.set_title("2-class Precision-Recall curve")
```



[104]:
```
ax = plt.gca()
plt.plot([0, 1], [0, 1], linestyle="--", label='No Skill')
log_disp.plot(ax=ax, alpha=0.8)
rfc_disp.plot(ax=ax, alpha=0.8)
```

```
knn_disp = RocCurveDisplay.from_estimator(knn, X_test_mod_sc, y_test, ax=ax,␣
 ↪alpha=0.8)
plt.title("ROC Curve Comparing all models")
plt.show()
```



Support Vector Machine

[105]:
```
svm = SVC(kernel='rbf', random_state=0).fit(X_train_mod_sc, y_train)
```

[106]:
```
svm.score(X_test_mod_sc, y_test)
```

[106]: 0.869

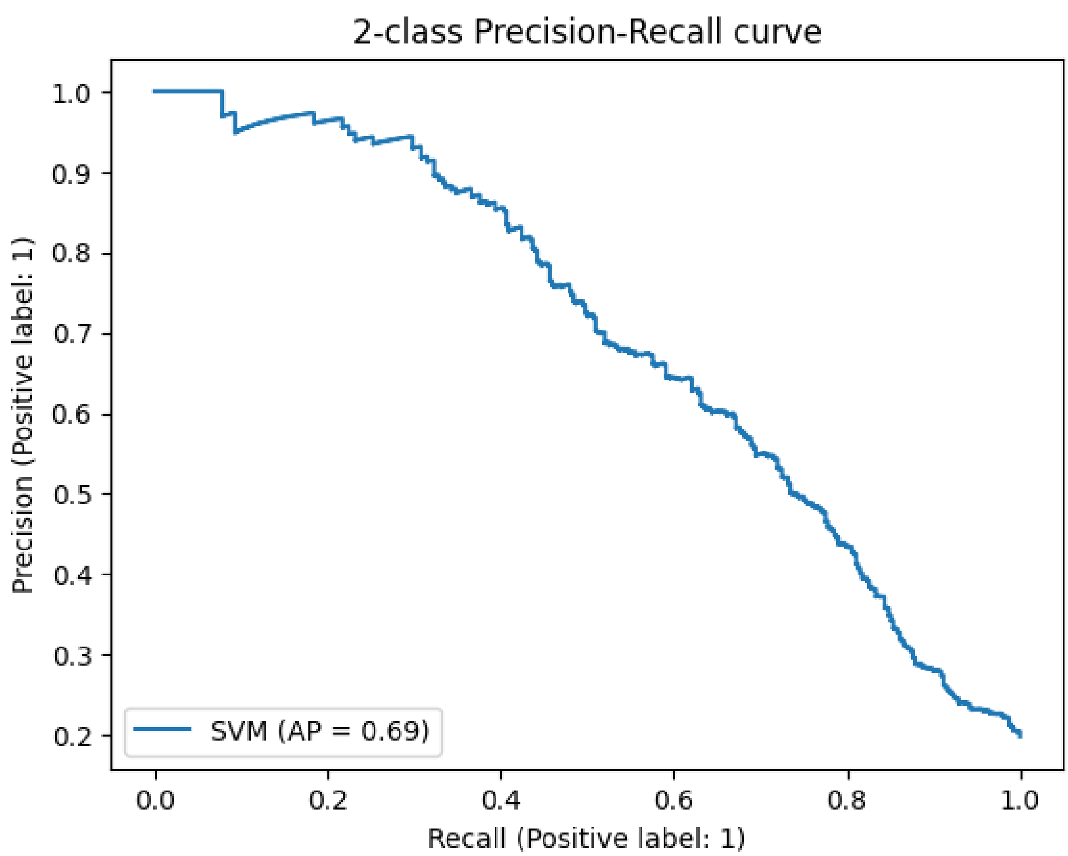
[107]:
```
svm_preds = svm.predict(X_test_mod_sc)
```

[108]:
```
svm_accuracy = accuracy_score(y_test, svm_preds) #RF fit to vars chosen by␣
 ↪Lasso still performs the best
print('Accuracy/Score is {}'.format(svm_accuracy))
print(confusion_matrix(y_test, svm_preds))
print(classification_report(y_test, svm_preds, zero_division=1))
```

```
Accuracy/Score is 0.869
[[1570   34]
 [ 228  168]]
              precision    recall  f1-score   support

           0       0.87      0.98      0.92      1604
           1       0.83      0.42      0.56       396

    accuracy                           0.87      2000
   macro avg       0.85      0.70      0.74      2000
weighted avg       0.86      0.87      0.85      2000
```

[109]:

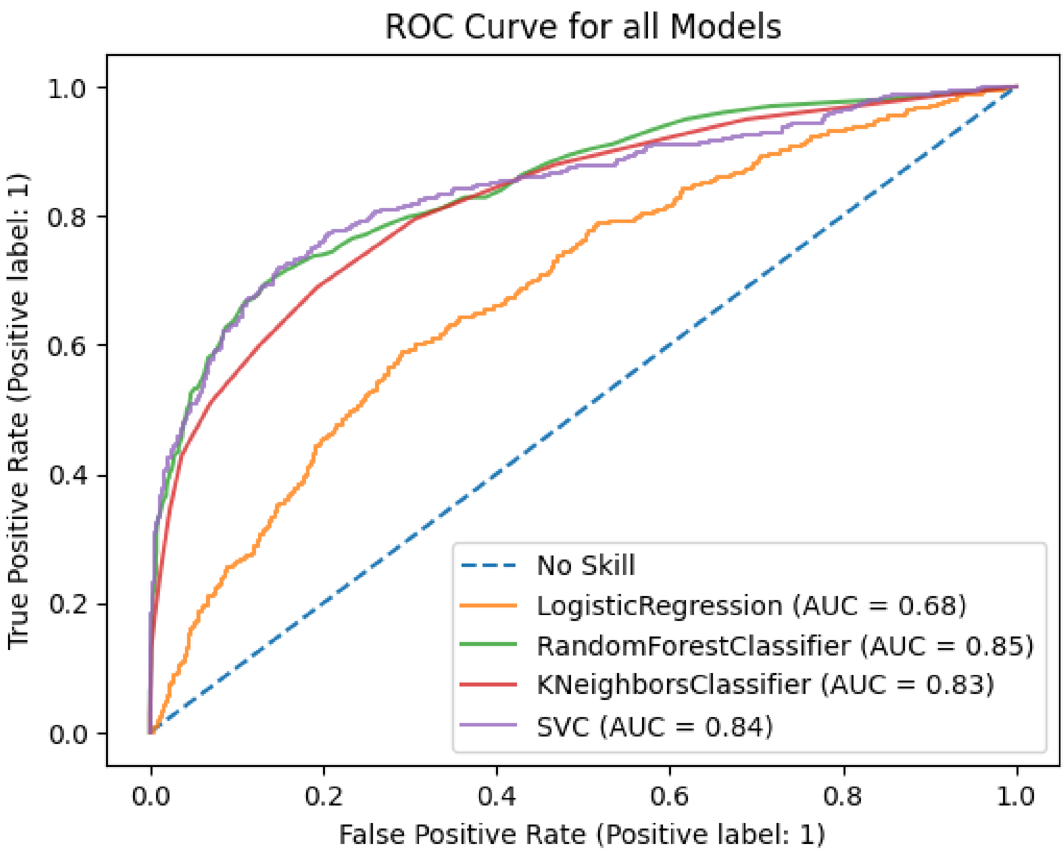
```
display = PrecisionRecallDisplay.from_estimator(
    svm, X_test_mod_sc, y_test, name="SVM"
)
_ = display.ax_.set_title("2-class Precision-Recall curve")
```



[110]:

```
ax = plt.gca()
plt.plot([0, 1], [0, 1], linestyle="--", label='No Skill')
```

```
log_disp.plot(ax=ax, alpha=0.8)
rfc_disp.plot(ax=ax, alpha=0.8)
knn_disp.plot(ax=ax, alpha=0.8)
svm_disp = RocCurveDisplay.from_estimator(svm, X_test_mod_sc, y_test, ax=ax,␣
 ↪alpha=0.8)
plt.title("ROC Curve for all Models")
plt.show()
```



Evaluation and Final Results

[111]:
```
#mainly discussion of above; create chart comparing/contrasting models
#discuss which model best for classification
#discuss interpretation of variables
```

Conclusion

[112]:
```
#write about what was accomplished/learned
#suggestions for improvement: additional variables, segmentation of models,␣
 ↪additional models to try
```