Module 4 Assignment 2: Company Bankruptcy Prediction

Introduction:

In our week 4 assignment our group focused on building and evaluating several different machine learning models using scikit-learn. We sought to determine whether current liability variables could serve as effective predictors of a company's likelihood of bankruptcy by employing machine learning models. We started by splitting the dataset into training and validation sets and doing EDA to better understand the data. We built three models: SVM, logistic regression, and Naive Bayes, evaluating their assumptions and performance metrics on both training and validation sets. Additionally, we conducted hyperparameter tuning for the SVM model and visualized model performance using ROC and Precision-Recall curves. Finally, we assessed model performance using the F-1 score on the validation set, providing insights into predictive accuracy.

Splitting and EDA:

In the dataset, we found that we were not able to build out models since the column names all had invisible spaces so we had to create a function where we introduced an underscore in that space to recall those columns. From here we created two different datasets one called train_data and one val_data and split them into 80% and 20% respectively. From this, we created graphs based on the train data to see how many 0 and 1s were in the dataset. We found that a lot of them were populated as 0's. Furthermore, we created different graphs that highlighted the different features that were right-skewed. We created a histogram where it talked about the Current Liability to Current Assets. In addition, we created another histogram about the Quick ratio. Additionally, we added a heatmap to show the correlations of all columns to the column 'Bankrupt?'. We then created a top_correlated_feature variable where we were able to find the columns that correlated best with the column 'Bankrupt?'

SVM, Logistic Regression, and Naïve Bayes models:

The Support Vector Machine (SVM) model is employed to predict bankruptcy status based on selected financial features (i.e. current liability to assets, current liability to equity, current liability to equity, current liability to current assets, and liability to Equity). The selected features are used to train the SVM classifier utilizing the linear kernel. The accuracy achieved on the validation set is 0.9626, indicating a high level of correct prediction overall. However, the classification output reveals a significant issue with correctly identifying instances of bankruptcy (class 1), where the model exhibits low precision, recall, and F1 score, suggesting it had a hard time accurately classifying instances belonging to the smaller class. This may suggest a potential class polarity or the need for further model fine-tuning to better handle the minority class. Later on, we then created a hyper tuning against this model using grid_search which helps automate the process of hyperparameter tuning which can achieve a higher accuracy score (0.9648)

The Logistic model is used for binary classification, where the output is transformed using the logistic function to produce a probability of class membership. The training of the logistic regression model involved the use of the same selected features used in the SVM model as predictors in determining bankruptcy. With an accuracy of 0.9626, the model seems to perform well, indicating its effectiveness in classifying instances into bankruptcy or non-bankruptcy classes.

Likewise, the Naive Bayes classification achieved an accuracy of 0.9523 demonstrating a robust performance in categorizing instances into bankruptcy or non-bankruptcy classes.

Goodness of fit:

For the goodness of fit, we created two different variables y_pred_train and y_pred_val by using the SVM.predict. From that, we created different equations based on TPR, FPR, recall, and accuracy. After conducting we were able to print out the results for both the training set and the validation set. Some insights that we can grab from this is that the accuracy score is high on both sets which indicates overall good performance. Furthermore, if we take a look at TPR it is relatively low which indicates that the model performance can be affected by a low score. Different parts can affect this, including missed opportunities or imbalance issues. When looking at the FPR scores it shows a low score which is a good thing and helps create a better model for us.

ROC and Precision:

In our group project, we evaluated the performance of three machine learning models – SVM, Logistic Regression, and Naive Bayes – using ROC and Precision-Recall curves. Our ROC analysis revealed that SVM and Naive Bayes, with an AUC of 0.82, outperformed Logistic Regression by 0.78, indicating a better capability at distinguishing between classes. The Precision-Recall curves suggested a significant trade-off between precision and recall for all models, which is crucial for us to consider, especially if we're dealing with imbalanced datasets. Collectively, these insights will guide us in refining our models and choosing the best one for our specific application needs.

F1-score:

For the F1-score, we delved into the classification report to assess our model's performance based on the F1-score, which is a harmonic mean of precision and recall. We observed a high F1-score of 0.98 for the negative class (label 0), which indicates exceptional model precision and recall for this group. However, the positive class (label 1) presented a concern with a low F1-score of 0.11, suggesting the model's difficulty in correctly identifying instances of this class, likely due to its small representation in the dataset (only 51 instances). Overall, the accuracy is high at 0.96, but the macro-average F1-score is 0.55, hinting at an imbalance in performance across classes.

Company_Bankruptcy

April 20, 2024

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report
from sklearn import svm
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score, roc_auc_score, confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn import svm
from sklearn.metrics import roc_curve, precision_recall_curve, auc
from sklearn.metrics import f1_score
```

[2]:
```
df = pd.read_csv('data.csv')
df
```

[2]:

	Bankrupt?	ROA(C) before interest and depreciation before interest	\
0	1	0.370594	
1	1	0.464291	
2	1	0.426071	
3	1	0.399844	
4	1	0.465022	
…	…	…	
6814	0	0.493687	
6815	0	0.475162	
6816	0	0.472725	
6817	0	0.506264	
6818	0	0.493053	

	ROA(A) before interest and % after tax	\
0	0.424389	
1	0.538214	
2	0.499019	
3	0.451265	

```
4                                    0.538432
…                                       …
6814                                 0.539468
6815                                 0.538269
6816                                 0.533744
6817                                 0.559911
6818                                 0.570105

       ROA(B) before interest and depreciation after tax  \
0                                                 0.405750
1                                                 0.516730
2                                                 0.472295
3                                                 0.457733
4                                                 0.522298
…                                                    …
6814                                              0.543230
6815                                              0.524172
6816                                              0.520638
6817                                              0.554045
6818                                              0.549548

       Operating Gross Margin   Realized Sales Gross Margin  \
0                    0.601457                     0.601457
1                    0.610235                     0.610235
2                    0.601450                     0.601364
3                    0.583541                     0.583541
4                    0.598783                     0.598783
…                       …                          …
6814                 0.604455                     0.604462
6815                 0.598308                     0.598308
6816                 0.610444                     0.610213
6817                 0.607850                     0.607850
6818                 0.627409                     0.627409

       Operating Profit Rate   Pre-tax net Interest Rate  \
0                   0.998969                    0.796887
1                   0.998946                    0.797380
2                   0.998857                    0.796403
3                   0.998700                    0.796967
4                   0.998973                    0.797366
…                      …                         …
6814                0.998992                    0.797409
6815                0.998992                    0.797414
6816                0.998984                    0.797401
6817                0.999074                    0.797500
6818                0.998080                    0.801987
```

```
      After-tax net Interest Rate  \
0                        0.808809
1                        0.809301
2                        0.808388
3                        0.808966
4                        0.809304
…                           …
6814                     0.809331
6815                     0.809327
6816                     0.809317
6817                     0.809399
6818                     0.813800

      Non-industry income and expenditure/revenue  …  \
0                                        0.302646  …
1                                        0.303556  …
2                                        0.302035  …
3                                        0.303350  …
4                                        0.303475  …
…                                           …  …
6814                                     0.303510  …
6815                                     0.303520  …
6816                                     0.303512  …
6817                                     0.303498  …
6818                                     0.313415  …

      Net Income to Total Assets   Total assets to GNP price  \
0                        0.716845                   0.009219
1                        0.795297                   0.008323
2                        0.774670                   0.040003
3                        0.739555                   0.003252
4                        0.795016                   0.003878
…                           …                          …
6814                     0.799927                   0.000466
6815                     0.799748                   0.001959
6816                     0.797778                   0.002840
6817                     0.811808                   0.002837
6818                     0.815956                   0.000707

      No-credit Interval   Gross Profit to Sales  \
0               0.622879               0.601453
1               0.623652               0.610237
2               0.623841               0.601449
3               0.622929               0.583538
4               0.623521               0.598782
…                  …                      …
6814            0.623620               0.604455
```

```
6815            0.623931                0.598306
6816            0.624156                0.610441
6817            0.623957                0.607846
6818            0.626680                0.627408

      Net Income to Stockholder's Equity   Liability to Equity  \
0                                0.827890             0.290202
1                                0.839969             0.283846
2                                0.836774             0.290189
3                                0.834697             0.281721
4                                0.839973             0.278514
…                                   …                    …
6814                             0.840359             0.279606
6815                             0.840306             0.278132
6816                             0.840138             0.275789
6817                             0.841084             0.277547
6818                             0.841019             0.275114

      Degree of Financial Leverage (DFL)  \
0                                0.026601
1                                0.264577
2                                0.026555
3                                0.026697
4                                0.024752
…                                   …
6814                             0.027064
6815                             0.027009
6816                             0.026791
6817                             0.026822
6818                             0.026793

      Interest Coverage Ratio (Interest expense to EBIT)   Net Income Flag  \
0                                                0.564050                1
1                                                0.570175                1
2                                                0.563706                1
3                                                0.564663                1
4                                                0.575617                1
…                                                   …                  …
6814                                             0.566193                1
6815                                             0.566018                1
6816                                             0.565158                1
6817                                             0.565302                1
6818                                             0.565167                1

      Equity to Liability
0                0.016469
1                0.020794
```

```
2                 0.016474
3                 0.023982
4                 0.035490
…                      …
6814              0.029890
6815              0.038284
6816              0.097649
6817              0.044009
6818              0.233902

[6819 rows x 96 columns]
```

1 Q 1

Split the training set into an 80% training and 20% validation set and conduct / improve upon the previous EDA.

[3]:
```
new_columns = []
for c in df.columns:
    new_columns.append(c.replace(' ', '_'))

df.columns = new_columns

print(df.head())
```

```
   Bankrupt?  _ROA(C)_before_interest_and_depreciation_before_interest  \
0          1                                           0.370594
1          1                                           0.464291
2          1                                           0.426071
3          1                                           0.399844
4          1                                           0.465022

   _ROA(A)_before_interest_and_%_after_tax  \
0                                 0.424389
1                                 0.538214
2                                 0.499019
3                                 0.451265
4                                 0.538432

   _ROA(B)_before_interest_and_depreciation_after_tax  \
0                                           0.405750
1                                           0.516730
2                                           0.472295
3                                           0.457733
4                                           0.522298

   _Operating_Gross_Margin  _Realized_Sales_Gross_Margin  \
```

```
0                 0.601457                      0.601457
1                 0.610235                      0.610235
2                 0.601450                      0.601364
3                 0.583541                      0.583541
4                 0.598783                      0.598783

   _Operating_Profit_Rate  _Pre-tax_net_Interest_Rate  \
0                0.998969                    0.796887
1                0.998946                    0.797380
2                0.998857                    0.796403
3                0.998700                    0.796967
4                0.998973                    0.797366

   _After-tax_net_Interest_Rate  _Non-industry_income_and_expenditure/revenue  \
0                      0.808809                                       0.302646
1                      0.809301                                       0.303556
2                      0.808388                                       0.302035
3                      0.808966                                       0.303350
4                      0.809304                                       0.303475

   …  _Net_Income_to_Total_Assets  _Total_assets_to_GNP_price  \
0  …                     0.716845                    0.009219
1  …                     0.795297                    0.008323
2  …                     0.774670                    0.040003
3  …                     0.739555                    0.003252
4  …                     0.795016                    0.003878

   _No-credit_Interval  _Gross_Profit_to_Sales  \
0             0.622879                0.601453
1             0.623652                0.610237
2             0.623841                0.601449
3             0.622929                0.583538
4             0.623521                0.598782

   _Net_Income_to_Stockholder's_Equity  _Liability_to_Equity  \
0                             0.827890              0.290202
1                             0.839969              0.283846
2                             0.836774              0.290189
3                             0.834697              0.281721
4                             0.839973              0.278514

   _Degree_of_Financial_Leverage_(DFL)  \
0                             0.026601
1                             0.264577
2                             0.026555
3                             0.026697
4                             0.024752
```

```
   _Interest_Coverage_Ratio_(Interest_expense_to_EBIT)  _Net_Income_Flag  \
0                                           0.564050                  1
1                                           0.570175                  1
2                                           0.563706                  1
3                                           0.564663                  1
4                                           0.575617                  1

   _Equity_to_Liability
0              0.016469
1              0.020794
2              0.016474
3              0.023982
4              0.035490

[5 rows x 96 columns]
```

[4]:
```
train_data, val_data = train_test_split(df , test_size = 0.2, random_state = 42)

print("Train Data Summary: ")
print(train_data.describe())
```

```
Train Data Summary:
         Bankrupt?  _ROA(C)_before_interest_and_depreciation_before_interest  \
count  5455.000000                                        5455.000000
mean      0.030981                                           0.505661
std       0.173281                                           0.061079
min       0.000000                                           0.000000
25%       0.000000                                           0.476698
50%       0.000000                                           0.503144
75%       0.000000                                           0.535538
max       1.000000                                           1.000000

       _ROA(A)_before_interest_and_%_after_tax  \
count                              5455.000000
mean                                  0.559095
std                                   0.065707
min                                   0.000000
25%                                   0.535843
50%                                   0.560347
75%                                   0.589730
max                                   1.000000

       _ROA(B)_before_interest_and_depreciation_after_tax  \
count                                         5455.000000
mean                                            0.554039
std                                             0.061805
min                                             0.000000
25%                                             0.527544
```

```
50%                                                0.552599
75%                                                0.584185
max                                                1.000000

       _Operating_Gross_Margin  _Realized_Sales_Gross_Margin  \
count              5455.000000                   5455.000000
mean                  0.607881                      0.607862
std                   0.017576                      0.017556
min                   0.000000                      0.000000
25%                   0.600455                      0.600448
50%                   0.606041                      0.606005
75%                   0.613831                      0.613734
max                   1.000000                      1.000000

       _Operating_Profit_Rate  _Pre-tax_net_Interest_Rate  \
count             5455.000000                 5455.000000
mean                 0.998705                    0.797142
std                  0.014538                    0.014375
min                  0.000000                    0.000000
25%                  0.998970                    0.797386
50%                  0.999023                    0.797465
75%                  0.999096                    0.797580
max                  1.000000                    1.000000

       _After-tax_net_Interest_Rate  \
count                   5455.000000
mean                       0.809030
std                        0.015196
min                        0.000000
25%                        0.809312
50%                        0.809376
75%                        0.809470
max                        1.000000

       _Non-industry_income_and_expenditure/revenue  …  \
count                                   5455.000000  …
mean                                       0.303644  …
std                                        0.012474  …
min                                        0.000000  …
25%                                        0.303466  …
50%                                        0.303525  …
75%                                        0.303587  …
max                                        1.000000  …

       _Net_Income_to_Total_Assets  _Total_assets_to_GNP_price  \
count                  5455.000000                5.455000e+03
mean                      0.808033                1.748744e+07
std                       0.040405                3.774208e+08
```

min	0.000000	0.000000e+00
25%	0.796924	9.023314e-04
50%	0.810806	2.082324e-03
75%	0.826804	5.256510e-03
max	0.982879	9.820000e+09

	_No-credit_Interval	_Gross_Profit_to_Sales
count	5455.000000	5455.000000
mean	0.623951	0.607879
std	0.013233	0.017576
min	0.000000	0.000000
25%	0.623638	0.600456
50%	0.623881	0.606040
75%	0.624180	0.613832
max	1.000000	1.000000

	_Net_Income_to_Stockholder's_Equity	_Liability_to_Equity
count	5455.000000	5455.000000
mean	0.840329	0.280484
std	0.015865	0.015373
min	0.000000	0.133503
25%	0.840124	0.276952
50%	0.841201	0.278781
75%	0.842345	0.281457
max	1.000000	1.000000

	_Degree_of_Financial_Leverage_(DFL)
count	5455.000000
mean	0.027622
std	0.017363
min	0.000000
25%	0.026791
50%	0.026808
75%	0.026913
max	1.000000

	_Interest_Coverage_Ratio_(Interest_expense_to_EBIT)	_Net_Income_Flag
count	5455.000000	5455.0
mean	0.565323	1.0
std	0.012292	0.0
min	0.000000	1.0
25%	0.565158	1.0
50%	0.565254	1.0
75%	0.565724	1.0
max	0.736985	1.0

	_Equity_to_Liability
count	5455.000000

```
mean                 0.047751
std                  0.050763
min                  0.000000
25%                  0.024449
50%                  0.033776
75%                  0.052739
max                  1.000000

[8 rows x 96 columns]
```

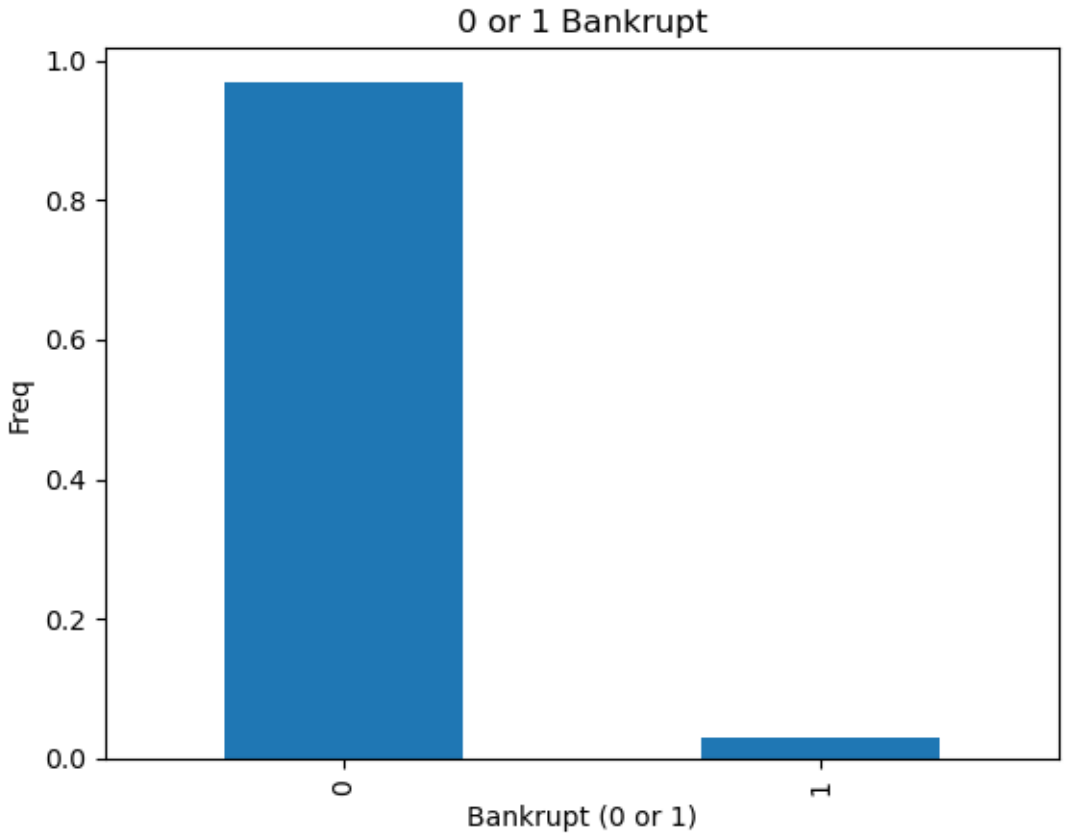
[5]:
```
pd.set_option('display.max_rows', None)
train_data.isna().sum()
```

[5]:
```
Bankrupt?                                                  0
_ROA(C)_before_interest_and_depreciation_before_interest   0
_ROA(A)_before_interest_and_%_after_tax                    0
_ROA(B)_before_interest_and_depreciation_after_tax         0
_Operating_Gross_Margin                                    0
_Realized_Sales_Gross_Margin                               0
_Operating_Profit_Rate                                     0
_Pre-tax_net_Interest_Rate                                 0
_After-tax_net_Interest_Rate                               0
_Non-industry_income_and_expenditure/revenue               0
_Continuous_interest_rate_(after_tax)                      0
_Operating_Expense_Rate                                    0
_Research_and_development_expense_rate                     0
_Cash_flow_rate                                            0
_Interest-bearing_debt_interest_rate                       0
_Tax_rate_(A)                                              0
_Net_Value_Per_Share_(B)                                   0
_Net_Value_Per_Share_(A)                                   0
_Net_Value_Per_Share_(C)                                   0
_Persistent_EPS_in_the_Last_Four_Seasons                   0
_Cash_Flow_Per_Share                                       0
_Revenue_Per_Share_(Yuan_¥)                                0
_Operating_Profit_Per_Share_(Yuan_¥)                       0
_Per_Share_Net_profit_before_tax_(Yuan_¥)                  0
_Realized_Sales_Gross_Profit_Growth_Rate                   0
_Operating_Profit_Growth_Rate                              0
_After-tax_Net_Profit_Growth_Rate                          0
_Regular_Net_Profit_Growth_Rate                            0
_Continuous_Net_Profit_Growth_Rate                         0
_Total_Asset_Growth_Rate                                   0
_Net_Value_Growth_Rate                                     0
_Total_Asset_Return_Growth_Rate_Ratio                      0
_Cash_Reinvestment_%                                       0
_Current_Ratio                                             0
```

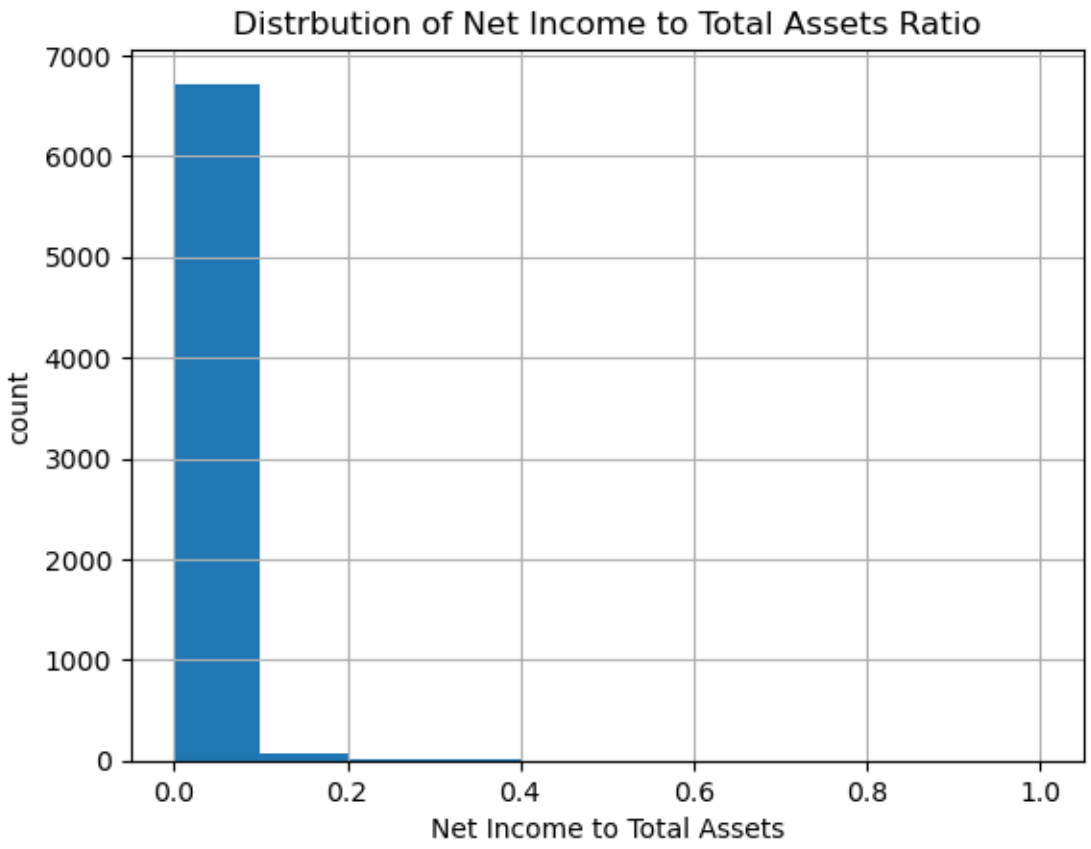
_Quick_Ratio	0
_Interest_Expense_Ratio	0
_Total_debt/Total_net_worth	0
_Debt_ratio_%	0
_Net_worth/Assets	0
_Long-term_fund_suitability_ratio_(A)	0
_Borrowing_dependency	0
_Contingent_liabilities/Net_worth	0
_Operating_profit/Paid-in_capital	0
_Net_profit_before_tax/Paid-in_capital	0
_Inventory_and_accounts_receivable/Net_value	0
_Total_Asset_Turnover	0
_Accounts_Receivable_Turnover	0
_Average_Collection_Days	0
_Inventory_Turnover_Rate_(times)	0
_Fixed_Assets_Turnover_Frequency	0
_Net_Worth_Turnover_Rate_(times)	0
_Revenue_per_person	0
_Operating_profit_per_person	0
_Allocation_rate_per_person	0
_Working_Capital_to_Total_Assets	0
_Quick_Assets/Total_Assets	0
_Current_Assets/Total_Assets	0
_Cash/Total_Assets	0
_Quick_Assets/Current_Liability	0
_Cash/Current_Liability	0
_Current_Liability_to_Assets	0
_Operating_Funds_to_Liability	0
_Inventory/Working_Capital	0
_Inventory/Current_Liability	0
_Current_Liabilities/Liability	0
_Working_Capital/Equity	0
_Current_Liabilities/Equity	0
_Long-term_Liability_to_Current_Assets	0
_Retained_Earnings_to_Total_Assets	0
_Total_income/Total_expense	0
_Total_expense/Assets	0
_Current_Asset_Turnover_Rate	0
_Quick_Asset_Turnover_Rate	0
_Working_capitcal_Turnover_Rate	0
_Cash_Turnover_Rate	0
_Cash_Flow_to_Sales	0
_Fixed_Assets_to_Assets	0
_Current_Liability_to_Liability	0
_Current_Liability_to_Equity	0
_Equity_to_Long-term_Liability	0
_Cash_Flow_to_Total_Assets	0

```
_Cash_Flow_to_Liability                                    0
_CFO_to_Assets                                            0
_Cash_Flow_to_Equity                                      0
_Current_Liability_to_Current_Assets                      0
_Liability-Assets_Flag                                    0
_Net_Income_to_Total_Assets                               0
_Total_assets_to_GNP_price                                0
_No-credit_Interval                                       0
_Gross_Profit_to_Sales                                    0
_Net_Income_to_Stockholder's_Equity                       0
_Liability_to_Equity                                      0
_Degree_of_Financial_Leverage_(DFL)                       0
_Interest_Coverage_Ratio_(Interest_expense_to_EBIT)       0
_Net_Income_Flag                                          0
_Equity_to_Liability                                      0
dtype: int64
```

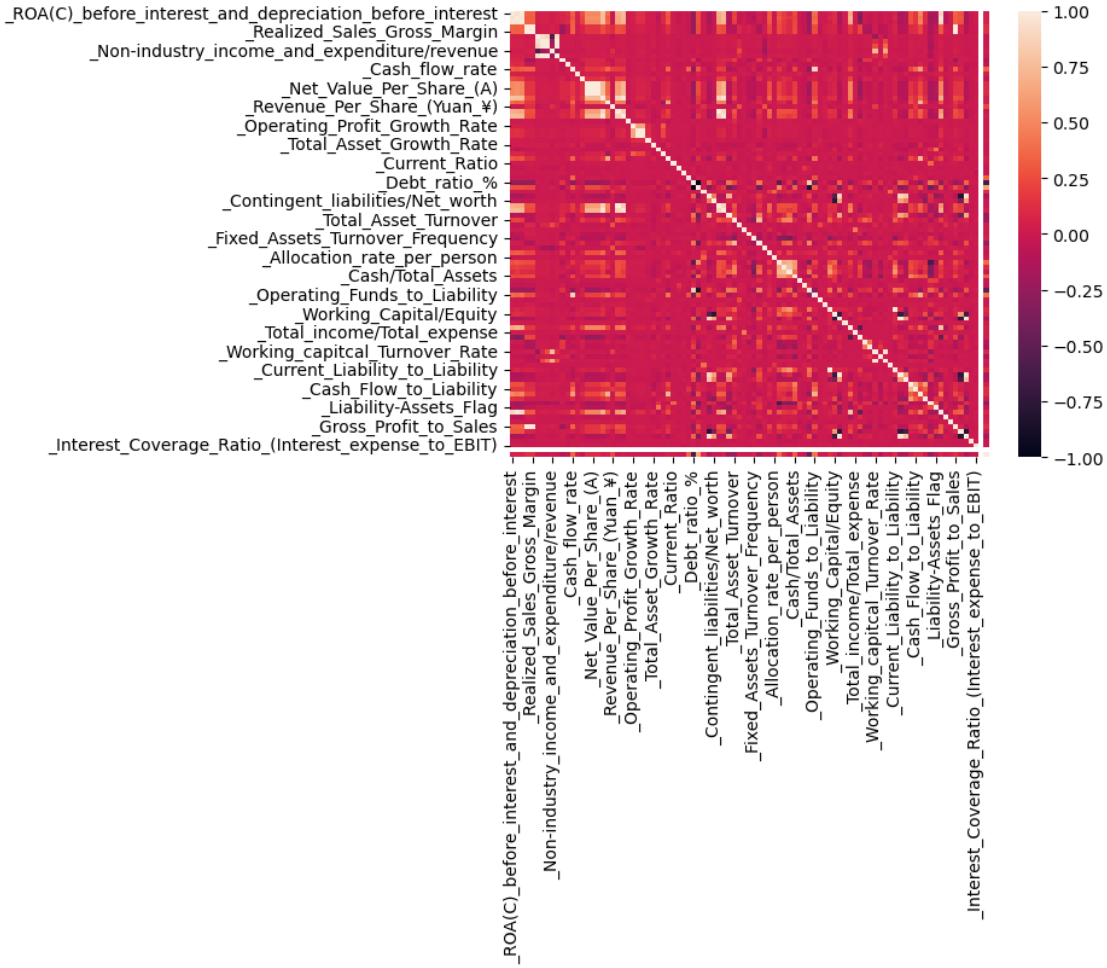
[6]:
```
train_data['Bankrupt?'].value_counts(normalize= True).plot(kind= 'bar')
plt.xlabel("Bankrupt (0 or 1)")
plt.ylabel("Freq")
plt.title("0 or 1 Bankrupt");
```



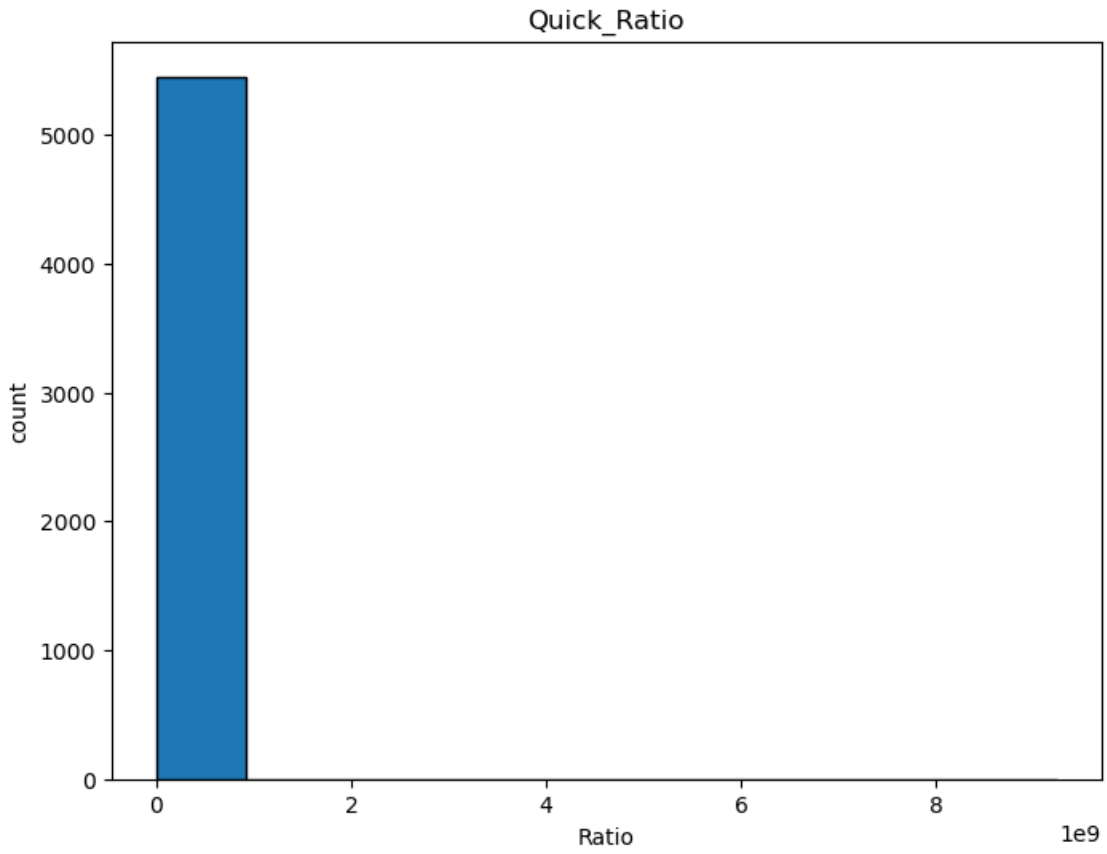
[7]:
```
df["_Current_Liability_to_Current_Assets"].hist()
plt.xlabel("Net Income to Total Assets")
plt.ylabel("count")
plt.title("Distrbution of Net Income to Total Assets Ratio");
```



[8]:
```
corr = df.drop(columns=['Bankrupt?']).corr()
sns.heatmap(corr);
```



[9]:
```
plt.figure(figsize=(8,6))
plt.hist(train_data['_Quick_Ratio'], edgecolor='black')
plt.title('Quick_Ratio')
plt.xlabel('Ratio')
plt.ylabel('count')
plt.show()
```

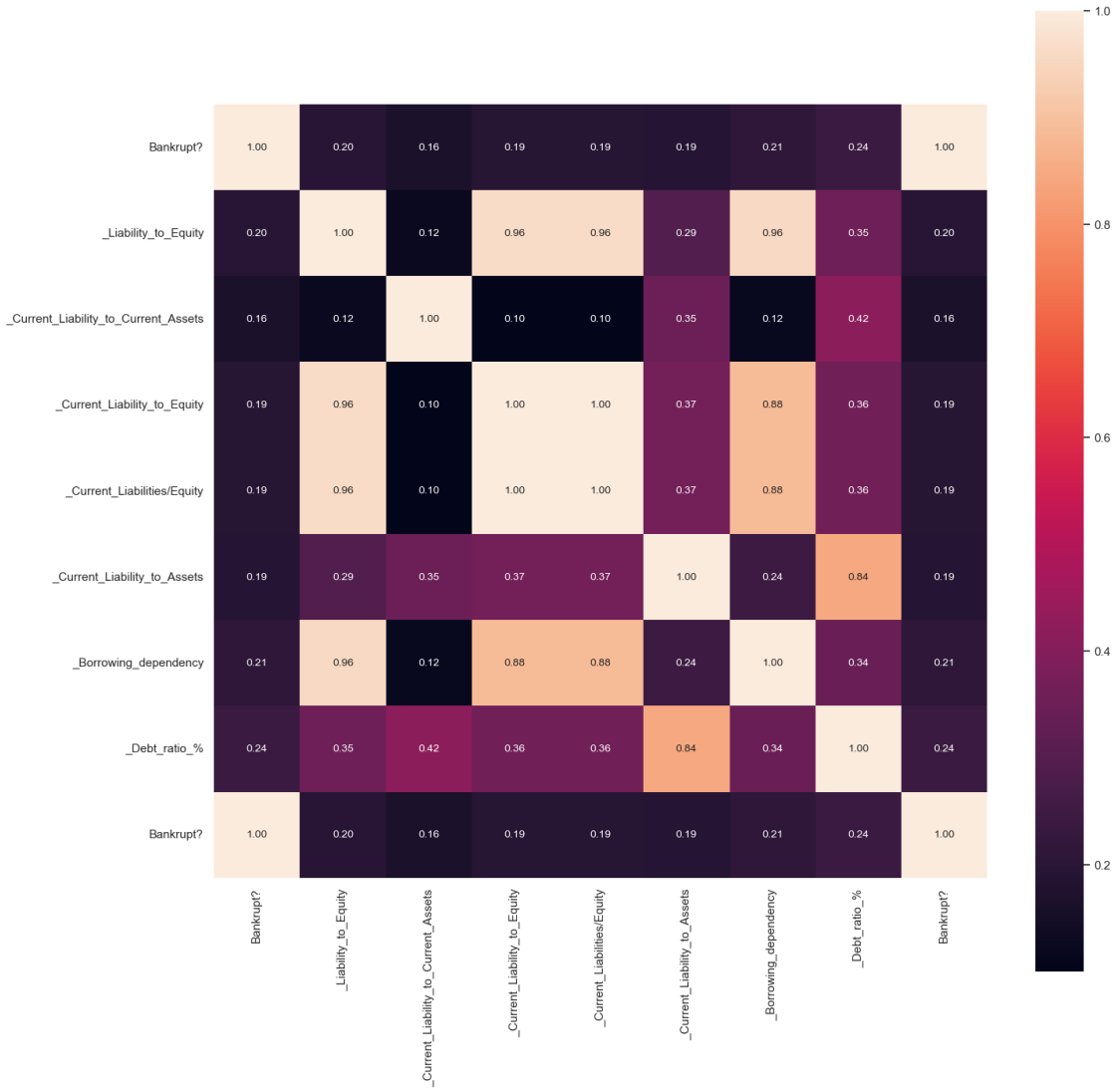


[10]:
```
top_correlations = train_data.corr()
top_feature_columns = top_correlations['Bankrupt?'][top_correlations['Bankrupt?
 ↪'].values > 0.15].index.values
top_feature_columns
```

[10]:
```
array(['Bankrupt?', '_Debt_ratio_%', '_Borrowing_dependency',
       '_Current_Liability_to_Assets', '_Current_Liabilities/Equity',
       '_Current_Liability_to_Equity',
       '_Current_Liability_to_Current_Assets', '_Liability_to_Equity'],
      dtype=object)
```

[11]:
```
heat_map_with_top_correlated_features = np.append(top_feature_columns[-12:], np.
 ↪array(['Bankrupt?']))
pearson_correlation_coefficients = np.
 ↪corrcoef(train_data[heat_map_with_top_correlated_features[::-1]].T)
plt.figure(figsize=(16,16))
sns.set(font_scale=1)
with sns.axes_style('white'):
    sns.heatmap(pearson_correlation_coefficients,␣
 ↪yticklabels=heat_map_with_top_correlated_features[::-1],
```

```
                xticklabels=heat_map_with_top_correlated_features[::-1], fmt='.
↪2f', annot_kws={'size': 10},
                annot=True, square=True, cmap=None)
```



2 Q 2

Build at least three models: an SVM, a logistic regression model, a Naïve Bayes model.

3 SVM Model

[12]:
```
selected_features = ['_Current_Liability_to_Assets',
                     '_Current_Liabilities/Equity',␣
 ↪'_Current_Liability_to_Equity',
                     '_Current_Liability_to_Current_Assets',␣
 ↪'_Liability_to_Equity']

# Subset the sampled training data to include only selected features
X_train_sampled = train_data[selected_features]
y_train_sampled = train_data['Bankrupt?']

# Subset the validation data to include only selected features
X_val = val_data[selected_features]
y_val = val_data['Bankrupt?']


# Initialize the SVM classifier
clf = svm.SVC(kernel='linear')

# Train the SVM classifier using the sampled training data
clf.fit(X_train_sampled, y_train_sampled)

# Predict labels for validation set
y_pred = clf.predict(X_val)

# Calculate accuracy on validation set
accuracy = accuracy_score(y_val, y_pred)
print("Accuracy:", accuracy)

report1 = classification_report(y_val, y_pred)
print("Classification Report:\n", report1)
```

```
Accuracy: 0.9626099706744868
Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98      1313
           1       0.00      0.00      0.00        51

    accuracy                           0.96      1364
   macro avg       0.48      0.50      0.49      1364
weighted avg       0.93      0.96      0.94      1364


C:\Users\Aaron\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
```

```
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Aaron\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Aaron\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

4 Logistic Regression model

[13]:
```
# Initializing the logistic regression model
log_reg_model = LogisticRegression()

# Training the model on the training data
log_reg_model.fit(X_train_sampled, y_train_sampled)

# Making predictions on the test data
predictions = log_reg_model.predict(X_val)

# Evaluating the model performance
accuracy = accuracy_score(y_val, predictions)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9626099706744868
```

5 Naïve Bayes model

[14]:
```
from sklearn.naive_bayes import GaussianNB
# Initialize the Naive Bayes classifier
nb_classifier = GaussianNB()

# Train the Naive Bayes classifier using the training data
nb_classifier.fit(X_train_sampled, y_train_sampled)

# Predict labels for validation set
y_pred = nb_classifier.predict(X_val)

# Calculate accuracy on validation set
accuracy = accuracy_score(y_val, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9523460410557185
```

6 Q 4

[15]:
```
# Define the parameter grid
param_grid = {
    'C': [0.1, 1, 10],  # Regularization parameter values to try
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],  # SVM kernel types to try
    'gamma': ['scale', 'auto']  # Kernel coefficient values for rbf, poly, and␣
 ↪sigmoid kernels
}

# Initialize SVM classifier
svm_classifier = svm.SVC()

# Initialize GridSearchCV
grid_search = GridSearchCV(svm_classifier, param_grid, cv=5, scoring='accuracy')

# Perform grid search
grid_search.fit(X_train_sampled, y_train_sampled)

# Print the best parameters found
print("Best parameters:", grid_search.best_params_)

# Get the best estimator
best_svm_classifier = grid_search.best_estimator_

# Evaluate the best estimator on the validation set
y_pred = best_svm_classifier.predict(X_val)

# Calculate accuracy
accuracy = accuracy_score(y_val, y_pred)
print("Accuracy:", accuracy)

# Generate classification report
report2 = classification_report(y_val, y_pred)
print("Classification Report:\n", report2)
```

```
Best parameters: {'C': 10, 'gamma': 'scale', 'kernel': 'sigmoid'}
Accuracy: 0.9648093841642229
Classification Report:
               precision    recall  f1-score   support

           0       0.96      1.00      0.98      1313
           1       1.00      0.06      0.11        51

    accuracy                           0.96      1364
   macro avg       0.98      0.53      0.55      1364
weighted avg       0.97      0.96      0.95      1364
```

7 Q 5

Evaluate goodness of fit metrics including TPR, FPR, precision, recall, and accuracy on the training and validation sets.

[16]:
```
y_pred_train = best_svm_classifier.predict(X_train_sampled)
y_pred_val = best_svm_classifier.predict(X_val)
```

[17]:
```
#goodness of fit
def calculate_metrics(y_true, y_pred):
    # Confusion matrix
    tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()

    # Accuracy
    accuracy = accuracy_score(y_true, y_pred)

    # Precision
    precision = precision_score(y_true, y_pred)

    # Recall
    recall = recall_score(y_true, y_pred)

    # True Positive Rate (TPR) / Recall / Sensitivity
    tpr = tp / (tp + fn)

    # False Positive Rate (FPR)
    fpr = fp / (fp + tn)

    return {
        "Accuracy": accuracy,
        "Precision": precision,
        "TPR": tpr,
        "FPR": fpr
    }

# Calculate metrics for both sets
metrics_train = calculate_metrics(y_train_sampled, y_pred_train)
metrics_val = calculate_metrics(y_val, y_pred_val)

print("Training Metrics:", metrics_train)
print("Validation Metrics:", metrics_val)
```

```
Training Metrics: {'Accuracy': 0.9692025664527956, 'Precision':
0.6666666666666666, 'TPR': 0.011834319526627219, 'FPR': 0.0001891789632992811 3}
Validation Metrics: {'Accuracy': 0.9648093841642229, 'Precision': 1.0, 'TPR':
0.058823529411764705, 'FPR': 0.0}
```

8 Q 6

Build ROC and Precision / Recall graphs.

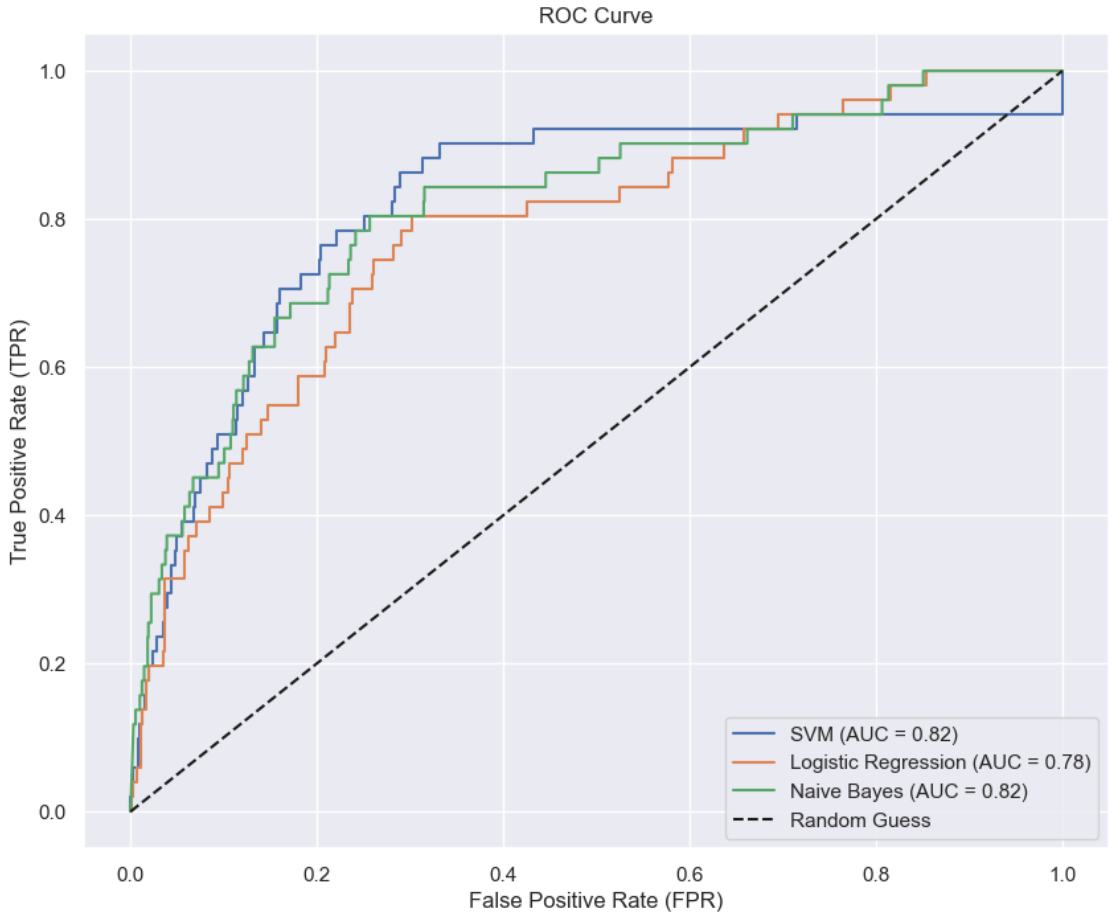
[18]:
```
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB

# Train SVM model
svm_model = SVC(kernel='linear')
svm_model.fit(X_train_sampled, y_train_sampled)

# Train logistic regression model
logistic_regression_model = LogisticRegression()
logistic_regression_model.fit(X_train_sampled, y_train_sampled)

# Train Naive Bayes model
naive_bayes_model = GaussianNB()
naive_bayes_model.fit(X_train_sampled, y_train_sampled)
```

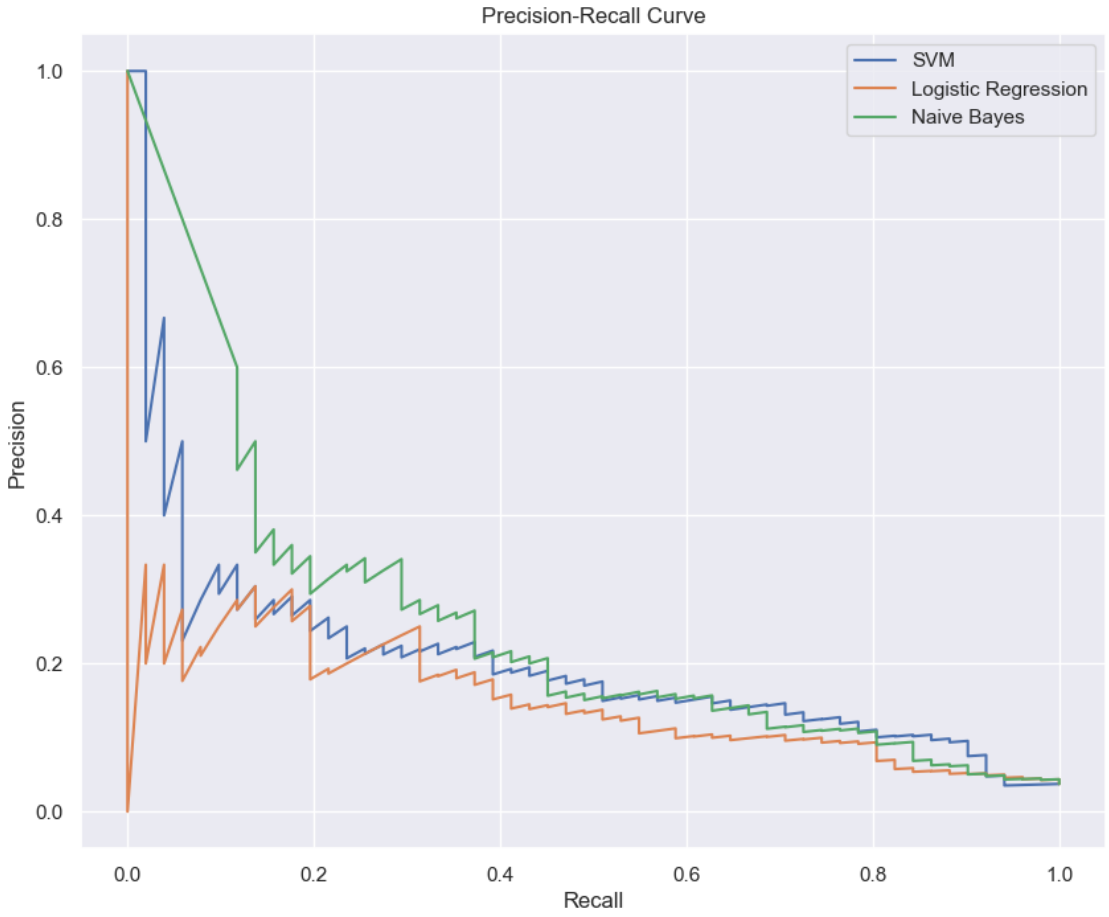
[18]:
```
GaussianNB()
```

[19]:
```
from sklearn.metrics import roc_curve, precision_recall_curve, auc

# Calculate ROC curve for each model
svm_fpr, svm_tpr, _ = roc_curve(y_val, svm_model.decision_function(X_val))
logistic_fpr, logistic_tpr, _ = roc_curve(y_val, logistic_regression_model.
 ↪decision_function(X_val))
nb_fpr, nb_tpr, _ = roc_curve(y_val, naive_bayes_model.predict_proba(X_val)[:,␣
 ↪1])

# Calculate Precision-Recall curve for each model
svm_precision, svm_recall, _ = precision_recall_curve(y_val, svm_model.
 ↪decision_function(X_val))
logistic_precision, logistic_recall, _ = precision_recall_curve(y_val,␣
 ↪logistic_regression_model.decision_function(X_val))
nb_precision, nb_recall, _ = precision_recall_curve(y_val, naive_bayes_model.
 ↪predict_proba(X_val)[:, 1])

# Calculate AUC scores for each model
svm_auc = auc(svm_fpr, svm_tpr)
logistic_auc = auc(logistic_fpr, logistic_tpr)
nb_auc = auc(nb_fpr, nb_tpr)
```

[20]:
```
# Plot ROC curves
plt.figure(figsize=(10, 8))
plt.plot(svm_fpr, svm_tpr, label=f'SVM (AUC = {svm_auc:.2f})')
```

```
plt.plot(logistic_fpr, logistic_tpr, label=f'Logistic Regression (AUC =␣
 ↪{logistic_auc:.2f})')
plt.plot(nb_fpr, nb_tpr, label=f'Naive Bayes (AUC = {nb_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve')
plt.legend()
plt.grid(True)
plt.show()

# Plot Precision-Recall curves
plt.figure(figsize=(10, 8))
plt.plot(svm_recall, svm_precision, label=f'SVM')
plt.plot(logistic_recall, logistic_precision, label=f'Logistic Regression')
plt.plot(nb_recall, nb_precision, label=f'Naive Bayes')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.grid(True)
plt.show()
```


ROC Curve
True Positive Rate (TPR)
False Positive Rate (FPR)
0.0
0.2
0.4
0.6
0.8
1.0
SVM (AUC = 0.82)
Logistic Regression (AUC = 0.78)
Naive Bayes (AUC = 0.82)
Random Guess



9 Q 7

Evaluate your models' performance on the validation set using the F1-score.

[21]:
```
report3 = classification_report(y_val,y_pred)
print("Classification Report:\n", report3)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98      1313
           1       1.00      0.06      0.11        51

    accuracy                           0.96      1364
   macro avg       0.98      0.53      0.55      1364
weighted avg       0.97      0.96      0.95      1364
```