Company Bankruptcy Prediction: SVM, logistic regression model, Naïve Bayes model

Can there be a way to predict the bankruptcy of a company based on the current company snapshot? A crystal ball, maybe a mythical one, may tell it in the past. However, with modern ML models like logistic regression, SVMs, and Naive Bayes, we may be able to predict the future. We will utilize a dataset of thousands of samples and ~100 attributes to build models that can predict bankruptcy.

The Bankruptcy dataset contains 95 explanatory variables describing various financial, operating, and debt ratios of thousands of companies and their effects on the independent flag: Bankruptcy. In all 6819 records, there are no null values (Figure 1). Figure 2 shows a correlation plot of dependent variables and the bankruptcy flag revealing variables that are positively and negatively correlated to bankruptcy. Figure 3 displays that most companies are not bankrupt. Since it is a binary flag, we are solving a classification problem.

Checking the distribution of dependent variables shows ones like liabilities and asset flags are binary; some, like research expense to net income, can be much higher than 100%; some, like Operating Gross Margin, cannot be higher than 100%. Figures 4, 5, and 6 are boxplots showing that some variables are balanced and distributed while others are concentrated in one value, with a few outliers. Average collection days, for example, have most values under 100, but others have values as high as 100 Million! We will not treat these outliers in this research since the outliers represent natural variations in the population, and they should be left to make sure our model works. Before creating our models, we split the dataset into 80% training and 20% test sets.

Support Vector Machine (SVM) models work by finding the optimal hyperplane that best separates the classes in the feature space. This hyperplane is chosen to maximize the margin between the classes, effectively maximizing the model's generalization capability. We standard-scaled all feature spaces, ensuring all contribute equally to the model. We then use GridSearch to search through a specified parameter grid and select the combination of hyperparameters that yields the best performance, as determined by 5-fold cross-validation. This resulted in {'C': 2.5, 'gamma': 0.03, 'kernel': 'rbf'} (Figure 7). Despite achieving high precision for the majority non-bankruptcy class, and high precision & recall for bankruptcy class in the trained dataset, the model struggled with the bankruptcy class in test dataset, as evident from the low recall score. This model generates a significant number of false negatives. We also find the model does too well on the training dataset, indicating overfitting. In the future, we can adjust class weights or explore alternative modeling approaches tailored to handle imbalanced datasets.

We followed a similar approach to our other two models: logistic regression and Naive Bayes. Tuning the hyperparameters using GridSearch resulted in a model we could train and test on. As seen in Figures 10-14, logistic regression yielded slightly better results for the bankruptcy class than SVM. Naive Bayes performed the worst as shown in Figures 15-19 with the lowest recall. These results are supported by the ROC and precision/recall curves and F1 scores in Figures 20-26. We see pretty poor performance with 0.19 being the highest F1 score for logistic regression. Going forward we need to clean the data more carefully and continue to tune the hyperparameters.

Clearly, the classifiers struggle with the bankrupt class. This may be because there are not many examples of bankrupt companies in the dataset as well (The weight of bankruptcy flag, unbalanced data). It also may be because bankruptcy may result from many reasons that cannot be generalized (randomness of bankruptcy flag, irrelevant features). We believe it is possible to predict these accurately given more data, better tuning, and adjust the imbalance of data.

Appendix

```
from google.colab import drive

drive.mount('/content/drive')


import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import sklearn

%cd /content/drive/My Drive/

df = pd.read_csv('Company_Bankruptcy/data.csv')

df.head(5)
```

	Bankrupt?	ROA(C) before interest and depreciation before interest	ROA(A) before interest and % after tax	ROA(B) before interest and depreciation after tax	Operating Gross Margin	Realized Sales Gross Margin	Operating Profit Rate	Pre-tax net Interest Rate	After-tax net Interest Rate	Non-industry income and expenditure/revenue	...	Net Income to Total Assets	Total assets to GNP price	No-credit Interval	Gross Profit to Sales	Net Sto
0	1	0.370594	0.424389	0.405750	0.601457	0.601457	0.998969	0.796887	0.808809	0.302646	...	0.716845	0.009219	0.622879	0.601453	
1	1	0.464291	0.538214	0.516730	0.610235	0.610235	0.998946	0.797380	0.809301	0.303556	...	0.795297	0.008323	0.623652	0.610237	
2	1	0.426071	0.499019	0.472295	0.601450	0.601364	0.998857	0.796403	0.808388	0.302035	...	0.774670	0.040003	0.623841	0.601449	
3	1	0.399844	0.451265	0.457733	0.583541	0.583541	0.998700	0.796967	0.808966	0.303350	...	0.739555	0.003252	0.622929	0.583538	
4	1	0.465022	0.538432	0.522298	0.598783	0.598783	0.998973	0.797366	0.809304	0.303475	...	0.795016	0.003878	0.623521	0.598782	

5 rows × 96 columns

```
# Check datatypes, all numeric data

data_type_counts = df.dtypes.value_counts()

print(data_type_counts)
```

```
float64 93 int64 3 Name: count, dtype: int64
```

```
len(df.columns)

96



#Null Value Columns

nullseries= df.isna().sum()

print(nullseries[nullseries > 0])

Series([], dtype: int64)
```

Figure 1

```
x = df.drop(columns=['Bankrupt?'])
y = df['Bankrupt?']

# Use K-Fold Later
from sklearn.model_selection import train_test_split
# Split the dataset into 80% train and 20% test
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
random_state=42)

#Create heat map of all numeric variables
# Select only numeric variables
x_numeric = x_train.select_dtypes(include=['int64', 'float64'])

# Calculate the correlation matrix
train_data = pd.concat([x_numeric, y], axis=1)
correlation_matrix = train_data.corr()
correlation_matrix =
correlation_matrix['Bankrupt?'].drop('Bankrupt?').sort_values()
correlation_matrix=correlation_matrix.sort_values()

# Plot barplot for correlation
plt.figure(figsize=(15, 10))
bar_plot = sns.barplot(x=correlation_matrix.index, y=correlation_matrix,
palette='coolwarm')
```

```
plt.title('Correlation of Bankrupt? and other Numeric Variables')

bar_plot.set_xticklabels(bar_plot.get_xticklabels(), rotation=90,

ha='right') # Rotate x-axis labels

plt.xlabel('Numeric Factors')

plt.ylabel('Correlation with Bankrupt?')

for index, value in enumerate(correlation_matrix):

plt.text(index, value, f'{value:.2f}', rotation=90, ha='center',

va='bottom')

plt.tight_layout() # Adjust layout to prevent clipping of labels

plt.show()
```

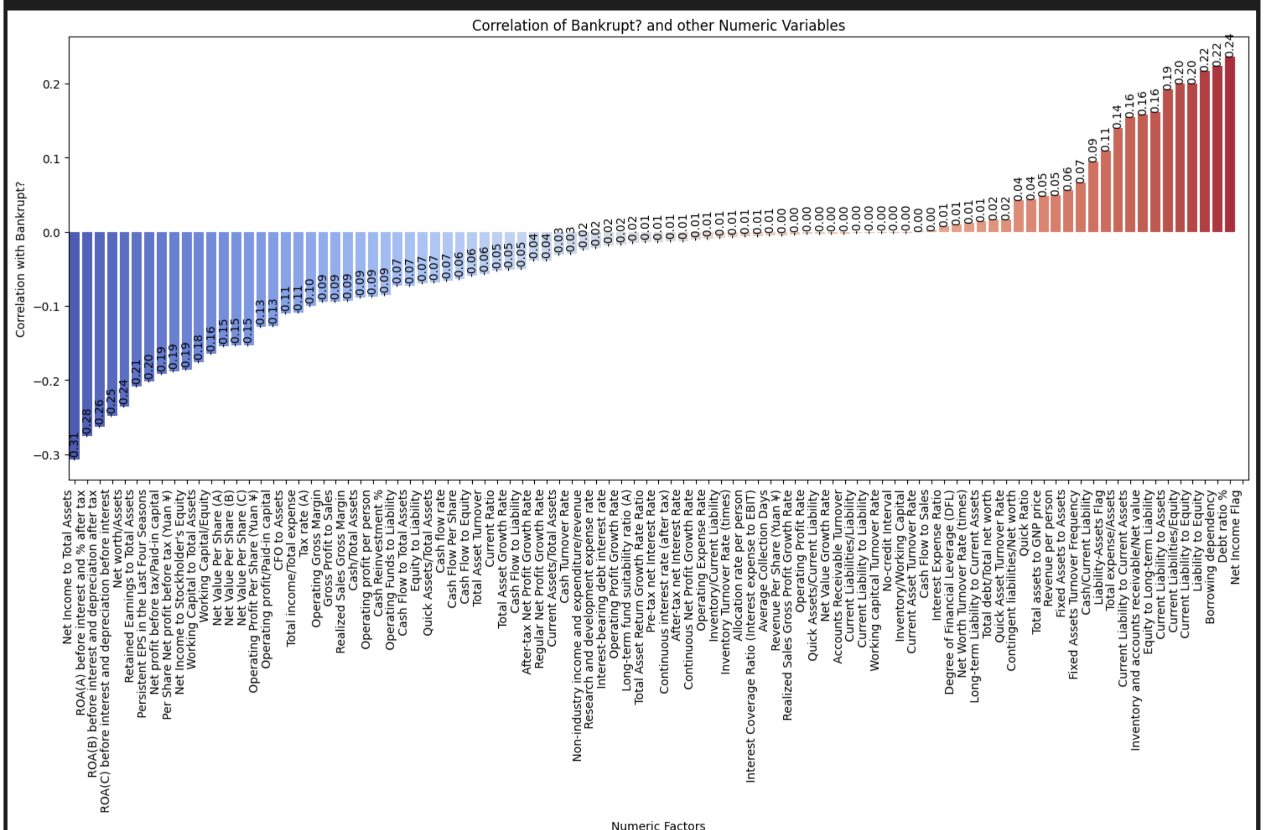


Figure 2

```
#Create Histogram

import matplotlib.pyplot as plt
```

```
# Assuming 'df' is your DataFrame and 'Bankrupt?' is the column name
plt.hist(df['Bankrupt?'], bins=2, edgecolor='black') # Assuming binary
data, adjust 'bins' as needed
plt.xlabel('Bankrupt?')
plt.ylabel('Frequency')
plt.title('Histogram of Bankrupt?')
plt.show()
```

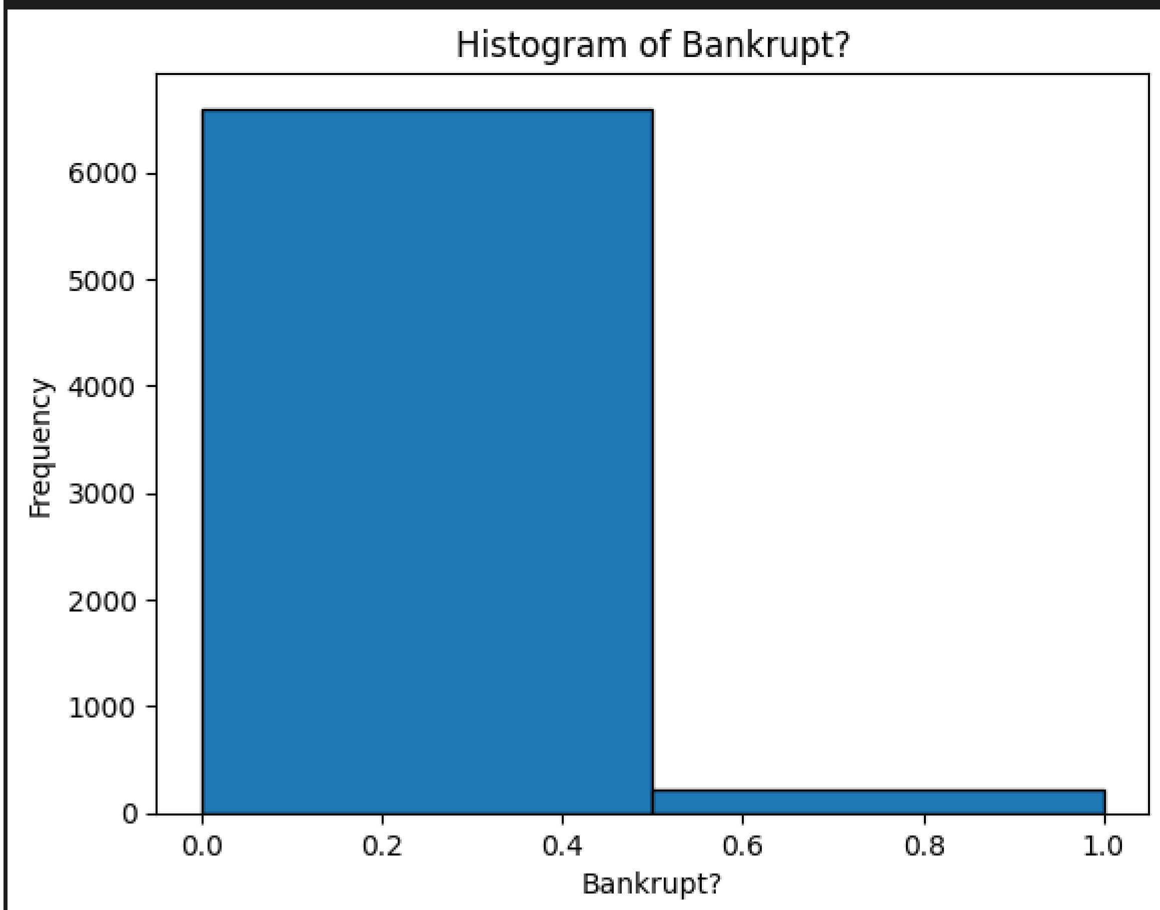


Figure 3

```
#Check Binary Column

binary_columns = df.select_dtypes(include=['int64']).columns


# Display the selected columns

print("Binary columns:")

print(binary_columns)


plt.figure(figsize=(10, 6))

sns.boxplot(data=df[binary_columns])

plt.title('Boxplot of Binary Columns')

plt.xlabel('Columns')

plt.ylabel('Values')

plt.xticks(rotation=45) # Rotate x-axis labels for better readability

plt.show()
```

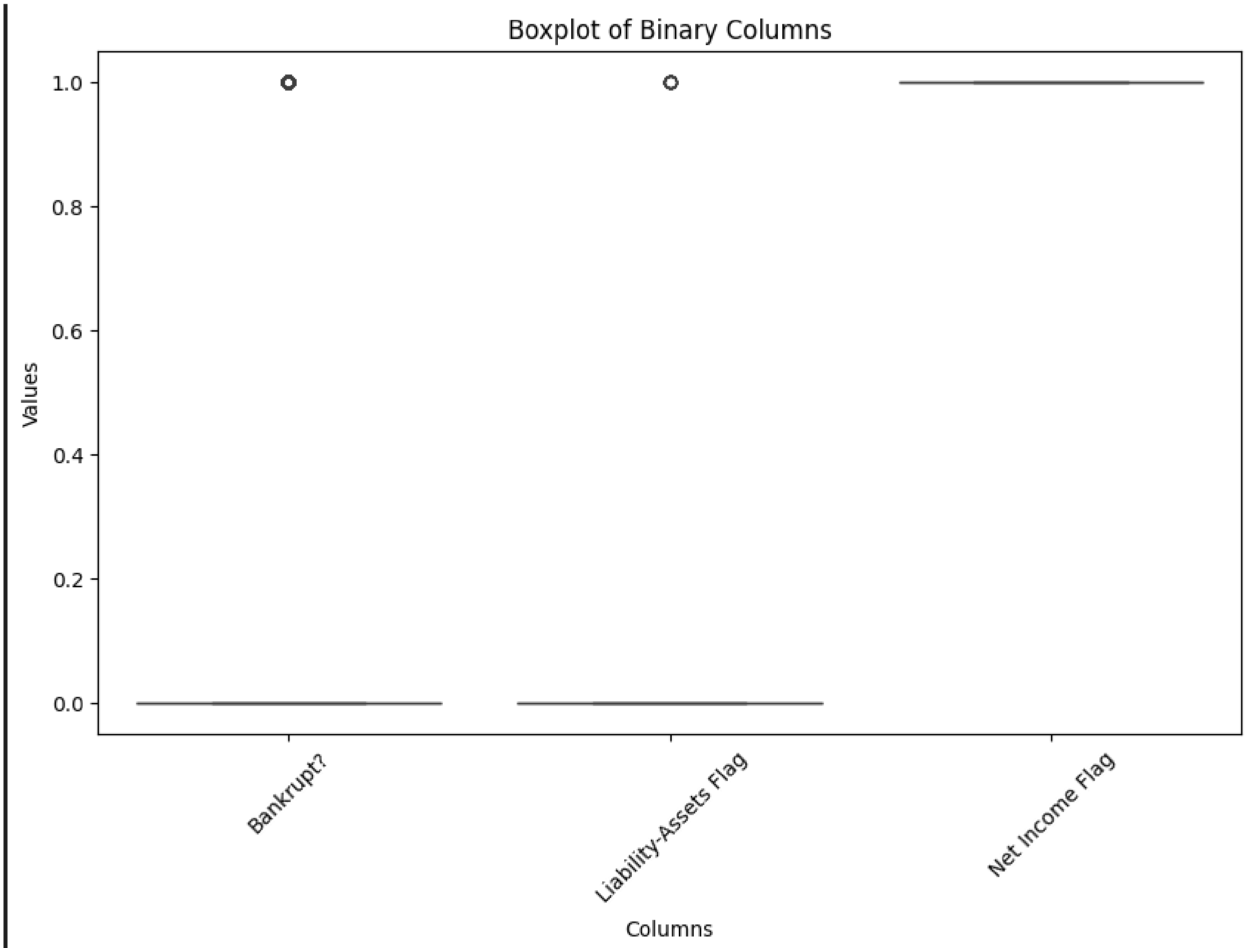


Figure 4

```
#Check Column with max value greater than 1
column_max = df.max()

# Filter out columns with average value greater than 1
columns_greater_than_1 = column_max[column_max > 1]

# Display the columns with max value greater than 1
print("Columns with max value greater than 1:")
print(columns_greater_than_1)
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df[columns_greater_than_1.index]/1000000)
plt.title('Boxplot of All Columns MAX > 1')
plt.xlabel('Columns')
plt.ylabel('Values Million')
plt.xticks(rotation=90) # Rotate x-axis labels for better readability
plt.show()
```

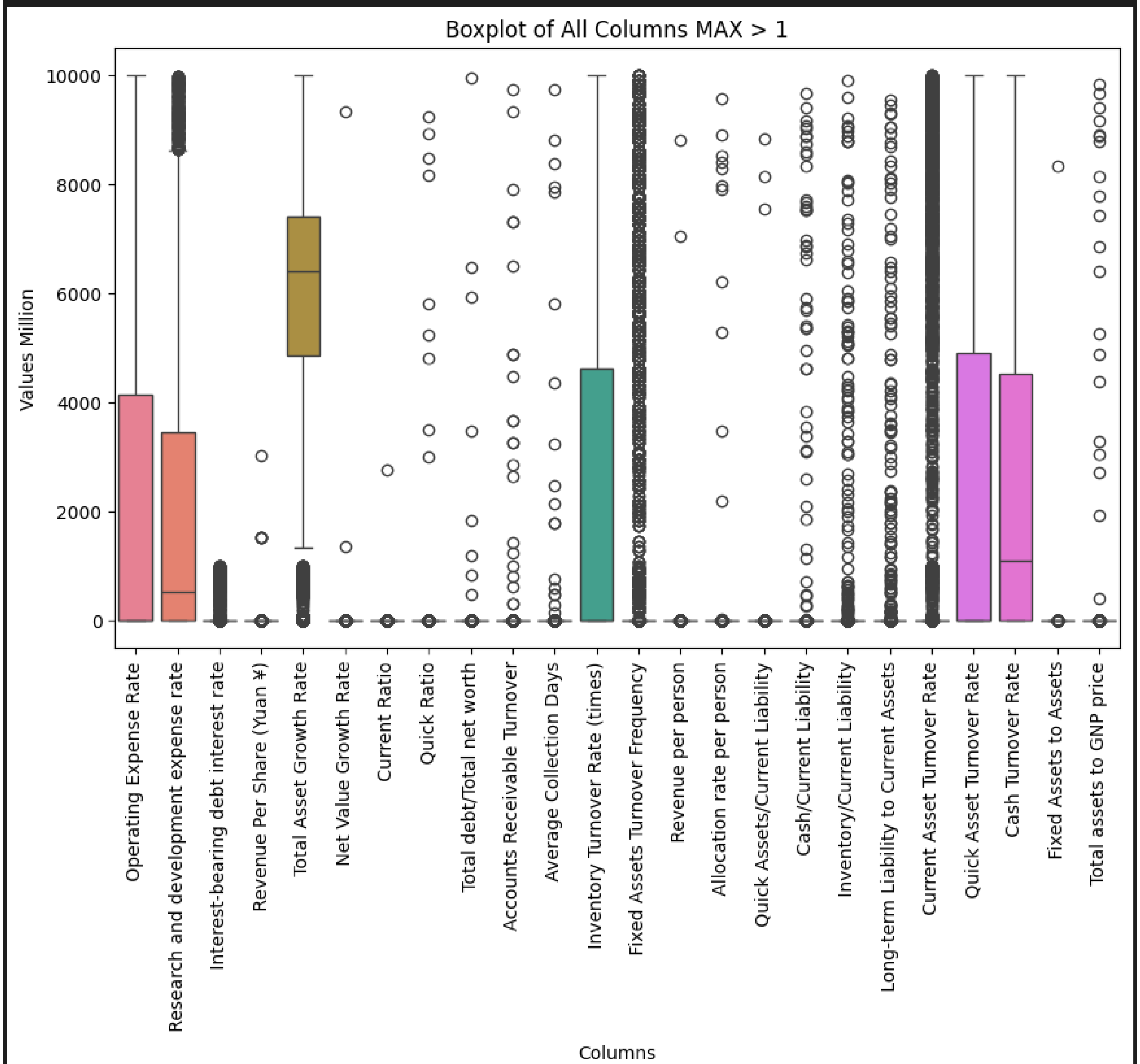


Figure 5

```
#Check Column with max value greater than 1
column_max = df.max()

# Filter out columns with average value greater than 1
columns_less_than_1 = column_max[column_max <= 1]

# Display the columns with max value greater than 1
print("Columns with max value less than 1:")
print(columns_less_than_1)

plt.figure(figsize=(20, 5))
sns.boxplot(data=df[columns_less_than_1.index])
plt.title('Boxplot of All Columns MAX <= 1')
plt.xlabel('Columns')
plt.ylabel('Values')
plt.xticks(rotation=90) # Rotate x-axis labels for better readability
plt.show()
```

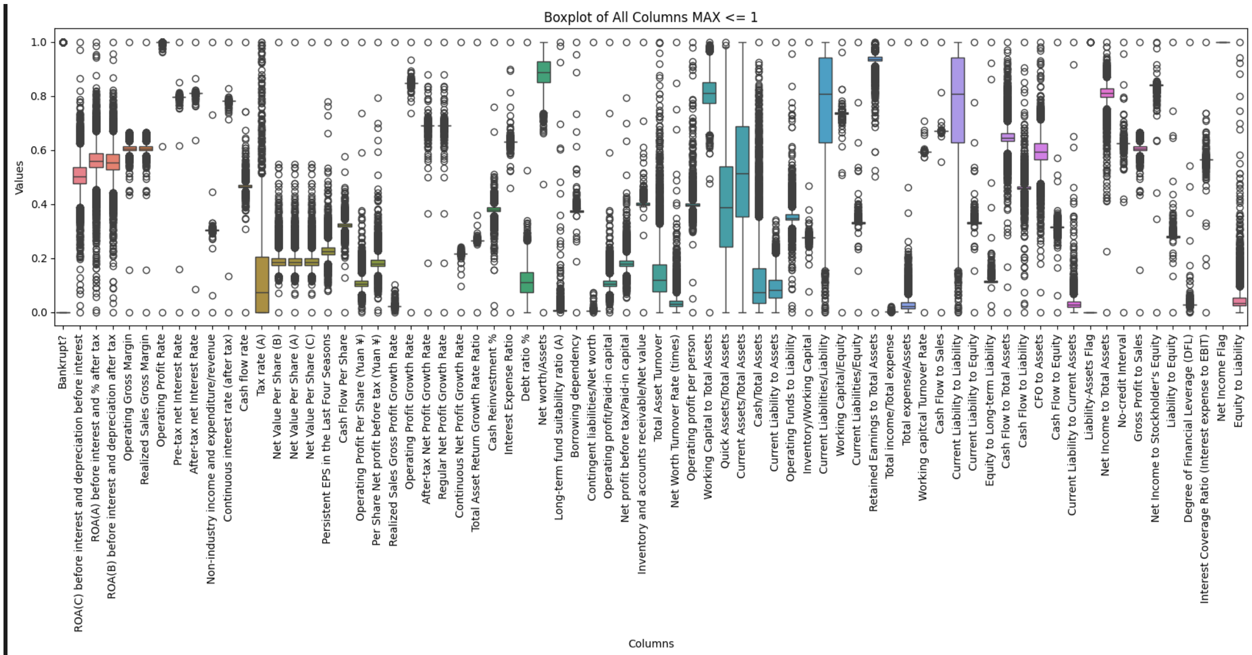


Figure 6

Support Vector Machine

```
# SVM
# Feature selection done in local environment using a better graphic cards.
# Since we have some dependent variable very large, standard scaler is required
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np

# Step 1: Standardize the features
```

```python
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)


# Step 2: Define the Support Vector Machine model
svm = SVC()

# Step 3: Define the parameters grid for GridSearchCV
param_grid = {
'C': [2.5],
'gamma': [0.03],
'kernel': ['rbf']
}

# Step 4: Perform GridSearchCV with K-fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)
grid_search = GridSearchCV(svm, param_grid, cv=kf, n_jobs=-1)
grid_search.fit(x_train_scaled, y_train)

# Step 5: Get the best parameters
best_params = grid_search.best_params_
best_params
```

```
{'C': 2.5, 'gamma': 0.03, 'kernel': 'rbf'}
```

Figure 7

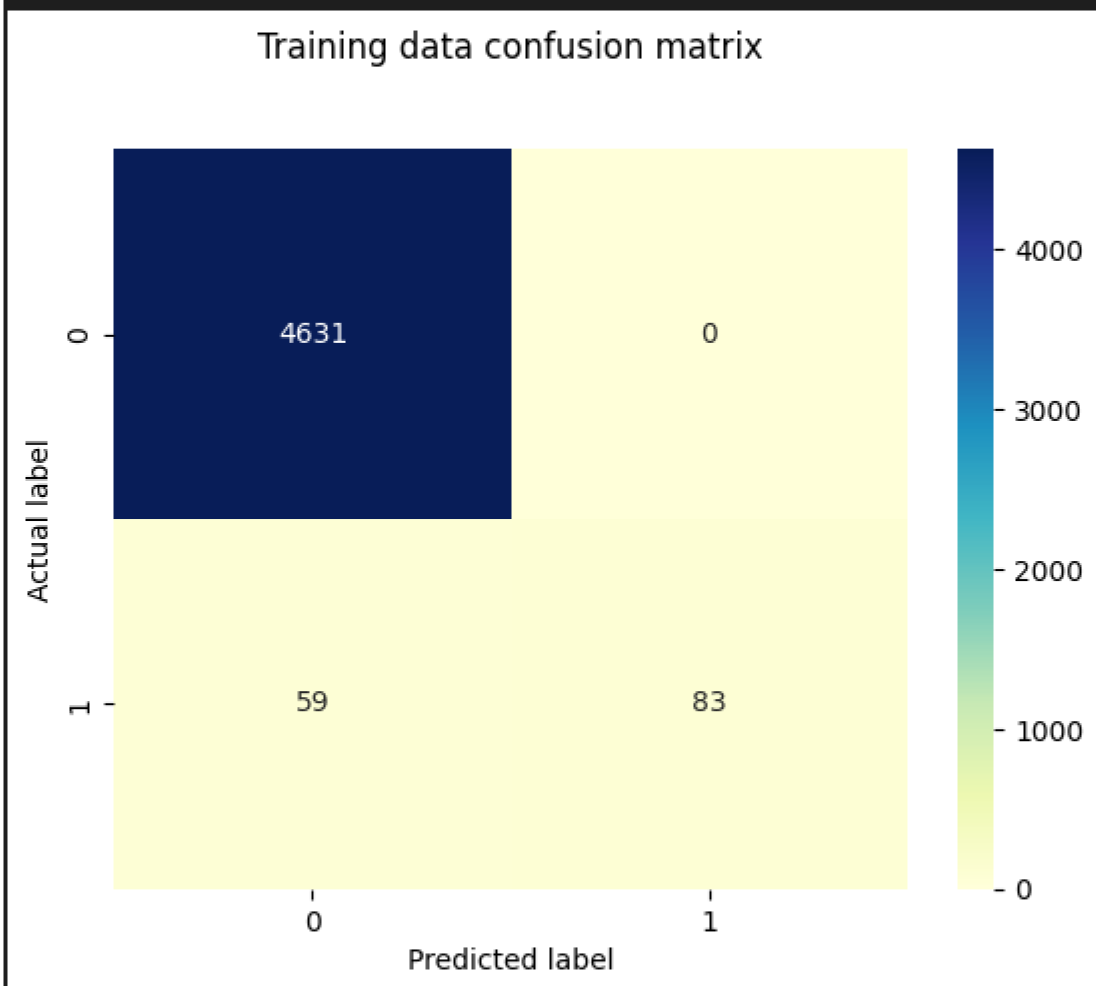
```
# Step 6: Train the SVM model with the best parameters
best_svm = SVC(**best_params)
best_svm.fit(x_train_scaled, y_train)

# Step 7: Predictions and evaluation
# On training data
y_train_pred = best_svm.predict(x_train_scaled)
train_conf_matrix = confusion_matrix(y_train, y_train_pred)
print("Confusion Matrix (Training Data):\n", train_conf_matrix)
print("\nClassification Report (Training Data):\n",
classification_report(y_train, y_train_pred))

sns.heatmap(pd.DataFrame(train_conf_matrix), annot=True,
cmap="YlGnBu" ,fmt='g')
plt.title('Training data confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()

# On testing data
y_test_pred = best_svm.predict(x_test_scaled)
test_conf_matrix = confusion_matrix(y_test, y_test_pred)
print("\nConfusion Matrix (Testing Data):\n", test_conf_matrix)
print("\nClassification Report (Testing Data):\n",
classification_report(y_test, y_test_pred))
```

```
sns.heatmap(pd.DataFrame(test_conf_matrix), annot=True,
cmap="YlGnBu" ,fmt='g')
plt.title('Test data confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
```




```
Confusion Matrix (Testing Data): [[1964 4] [ 75 3]] Classification Report
(Testing Data): precision recall f1-score support 0 0.96 1.00 0.98 1968 1
```

0.43 0.04 0.07 78 accuracy 0.96 2046 macro avg 0.70 0.52 0.53 2046

weighted avg 0.94 0.96 0.95 2046

Figure 8

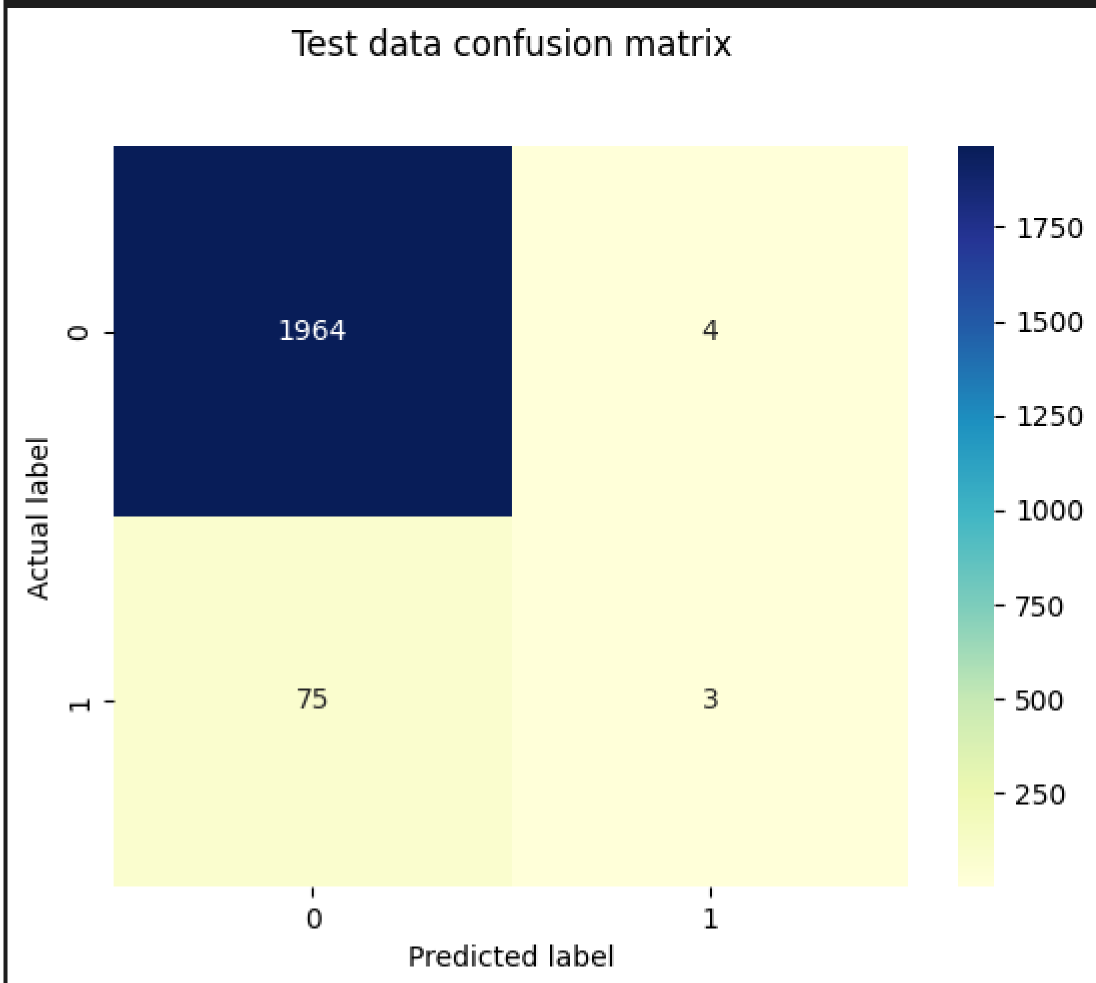


Figure 9

Logistic Regression

```
from sklearn.model_selection import GridSearchCV, KFold
```

```python
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report

# Standard Scale
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)


log_reg = LogisticRegression()

# Parameter Grid
param_grid = {
'C': [0.01, 0.1, 1, 10, 100, 1000]
}

# Grid Search
kf = KFold(n_splits=5, shuffle=True, random_state=42)
grid_search = GridSearchCV(log_reg, param_grid, cv=kf, n_jobs=-1)
grid_search.fit(x_train_scaled, y_train)

# Train on tuned logistic regression
best_params = grid_search.best_params_
tuned_log_reg = LogisticRegression(**best_params)
tuned_log_reg.fit(x_train_scaled, y_train)
```

```
LogisticRegression(C=0.01)
```

Figure 10

```python
# Predict
y_train_pred = tuned_log_reg.predict(x_train_scaled)
train_conf_matrix = confusion_matrix(y_train, y_train_pred)
print("Confusion Matrix (Training Data):\n", train_conf_matrix)
print("\nClassification Report (Training Data):\n",
classification_report(y_train, y_train_pred))

# On testing data
y_test_pred = tuned_log_reg.predict(x_test_scaled)
test_conf_matrix = confusion_matrix(y_test, y_test_pred)
print("\nConfusion Matrix (Testing Data):\n", test_conf_matrix)
print("\nClassification Report (Testing Data):\n",
classification_report(y_test, y_test_pred))
```

```
Confusion Matrix (Training Data): [[4620 11] [ 124 18]] Classification
Report (Training Data): precision recall f1-score support 0 0.97 1.00 0.99
4631 1 0.62 0.13 0.21 142 accuracy 0.97 4773 macro avg 0.80 0.56 0.60 4773
weighted avg 0.96 0.97 0.96 4773 Confusion Matrix (Testing Data): [[1964
4] [ 69 9]] Classification Report (Testing Data): precision recall f1-
score support 0 0.97 1.00 0.98 1968 1 0.69 0.12 0.20 78 accuracy 0.96 2046
macro avg 0.83 0.56 0.59 2046 weighted avg 0.96 0.96 0.95 2046
```

Figure 11

```
# import required modules
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

class_names=[0,1] # name of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(train_conf_matrix), annot=True,
cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Training data confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

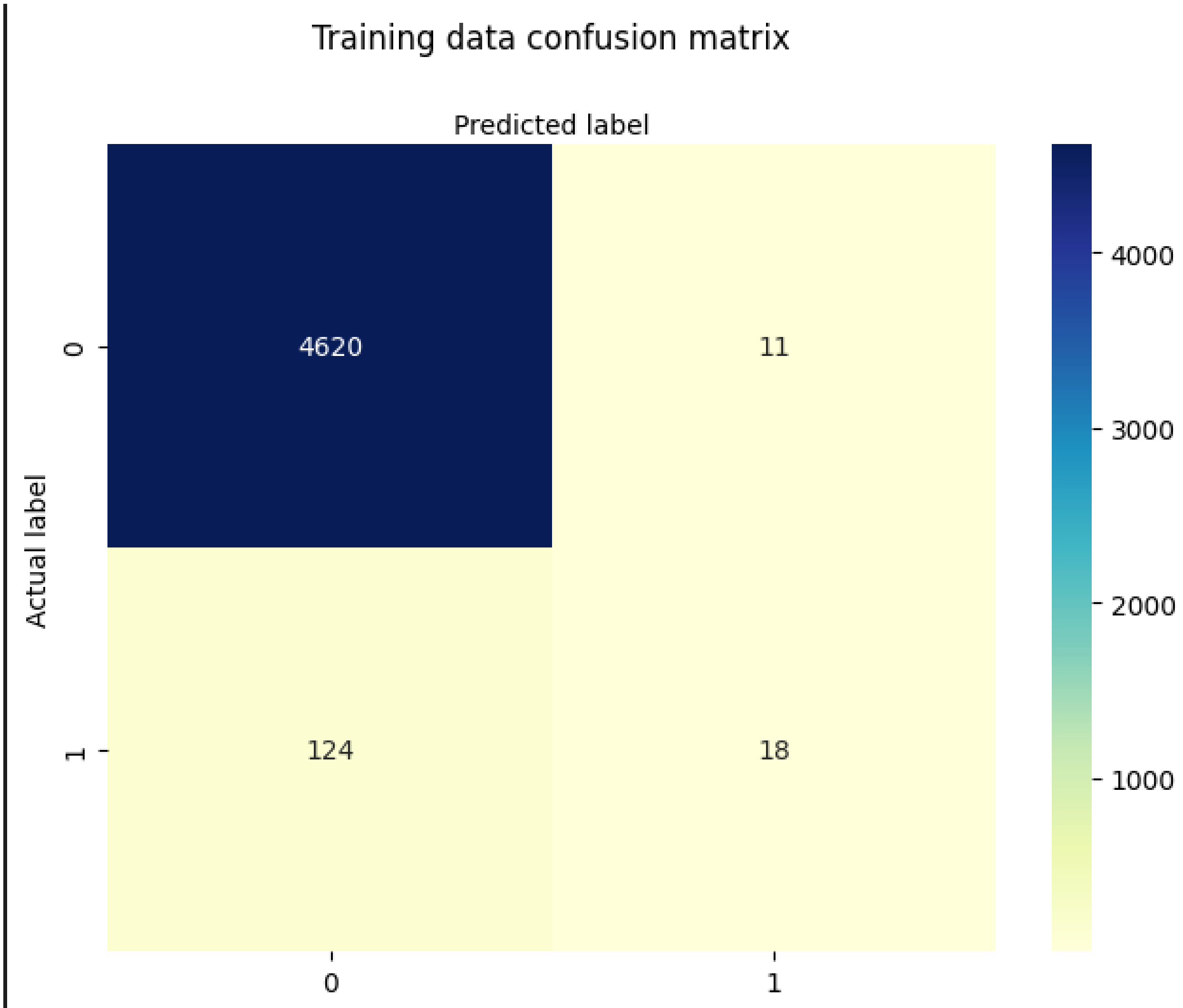


Figure 12

```
# import required modules
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

class_names=[0,1] # name of classes
```

```
fig, ax = plt.subplots()

tick_marks = np.arange(len(class_names))

plt.xticks(tick_marks, class_names)

plt.yticks(tick_marks, class_names)

# create heatmap

sns.heatmap(pd.DataFrame(train_conf_matrix), annot=True,

cmap="YlGnBu" ,fmt='g')

ax.xaxis.set_label_position("top")

plt.tight_layout()

plt.title('Training data confusion matrix', y=1.1)

plt.ylabel('Actual label')

plt.xlabel('Predicted label')
```

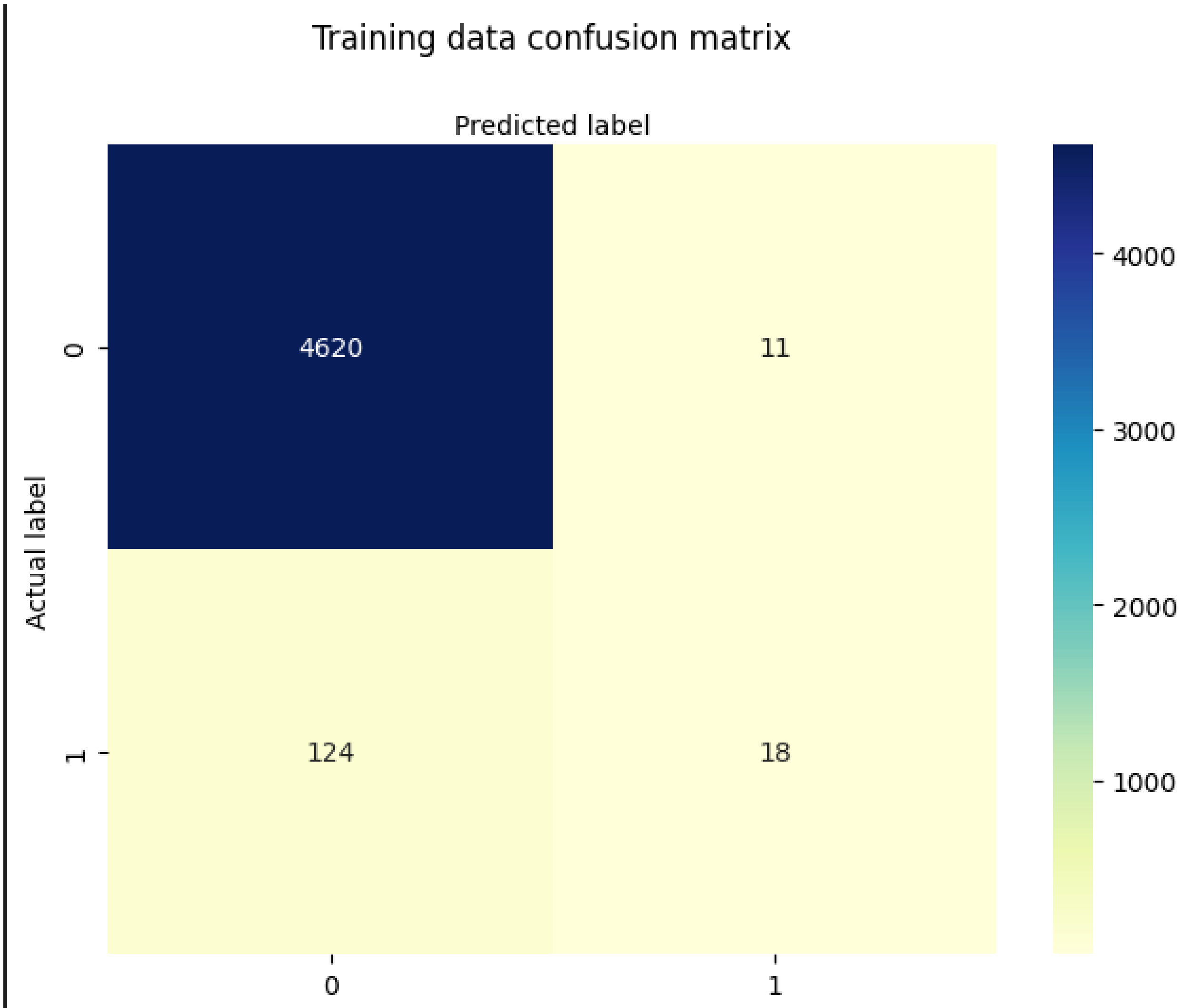


Figure 13

```
# import required modules
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

class_names=[0,1] # name of classes
fig, ax = plt.subplots()
```

```
tick_marks = np.arange(len(class_names))

plt.xticks(tick_marks, class_names)

plt.yticks(tick_marks, class_names)

# create heatmap

sns.heatmap(pd.DataFrame(test_conf_matrix), annot=True,

cmap="YlGnBu" ,fmt='g')

ax.xaxis.set_label_position("top")

plt.tight_layout()

plt.title('Test data confusion matrix', y=1.1)

plt.ylabel('Actual label')

plt.xlabel('Predicted label')
```

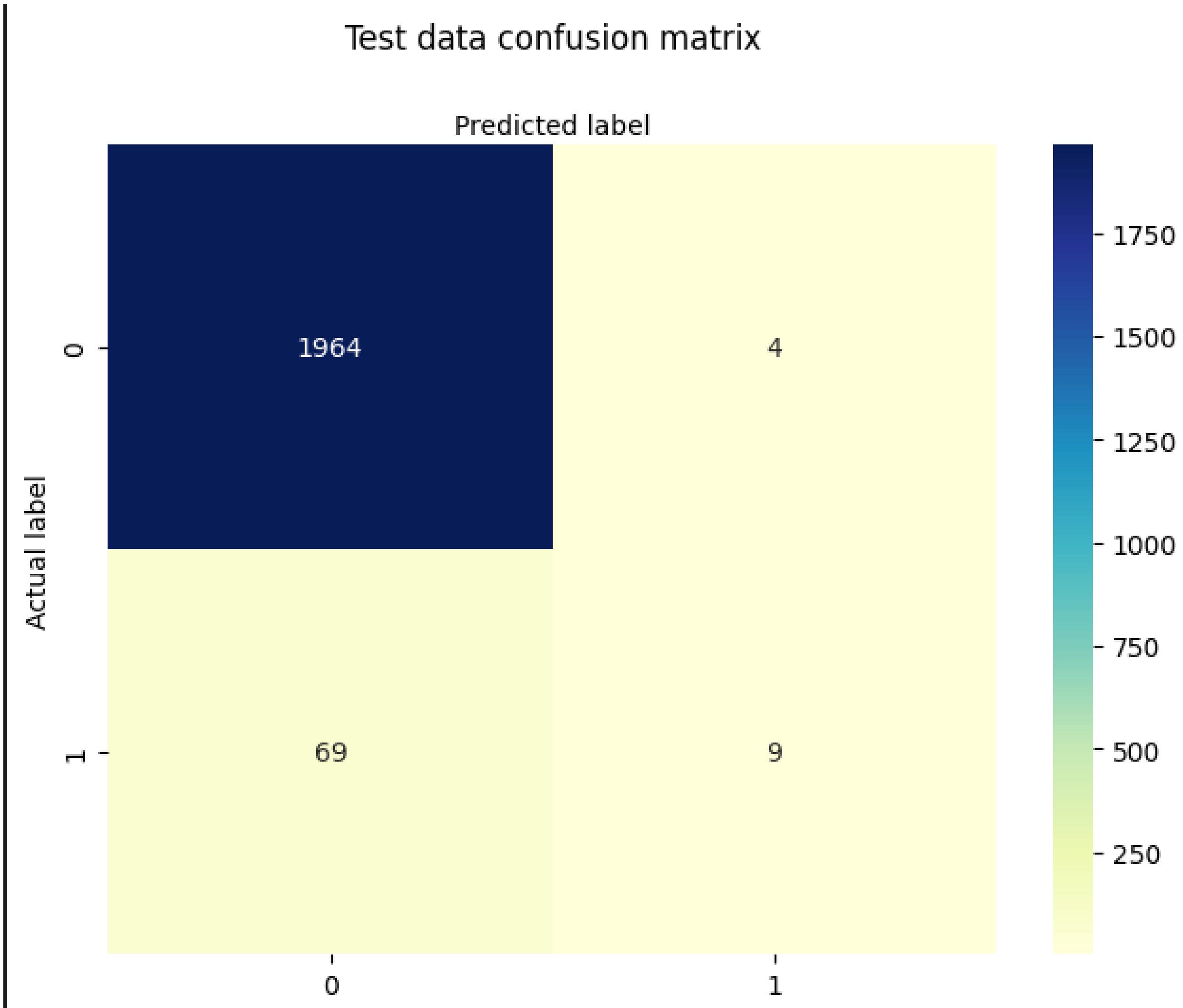


Figure 14

Naive Bayes

```
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report
```

```
# Standard Scale
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)


nb = GaussianNB()

# Parameter Grid
param_grid = {
'var_smoothing': [0.000000001, 0.000000001, 0.00000001, 0.0000001,
0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
}

# Grid Search
kf = KFold(n_splits=5, shuffle=True, random_state=42)
grid_search = GridSearchCV(nb, param_grid, cv=kf, n_jobs=-1)
grid_search.fit(x_train_scaled, y_train)

# Train on tuned logistic regression
best_params = grid_search.best_params_
tuned_nb = GaussianNB(**best_params)
tuned_nb.fit(x_train_scaled, y_train)
```

```
GaussianNB
GaussianNB(var_smoothing=100)
```

Figure 15

```
# Predict
y_train_pred = tuned_nb.predict(x_train_scaled)
train_conf_matrix = confusion_matrix(y_train, y_train_pred)
print("Confusion Matrix (Training Data):\n", train_conf_matrix)
print("\nClassification Report (Training Data):\n",
classification_report(y_train, y_train_pred))

# On testing data
y_test_pred = tuned_nb.predict(x_test_scaled)
test_conf_matrix = confusion_matrix(y_test, y_test_pred)
print("\nConfusion Matrix (Testing Data):\n", test_conf_matrix)
print("\nClassification Report (Testing Data):\n",
classification_report(y_test, y_test_pred))

Confusion Matrix (Training Data): [[4631 0] [ 140 2]] Classification
Report (Training Data): precision recall f1-score support 0 0.97 1.00 0.99
4631 1 1.00 0.01 0.03 142 accuracy 0.97 4773 macro avg 0.99 0.51 0.51 4773
weighted avg 0.97 0.97 0.96 4773 Confusion Matrix (Testing Data): [[1967
1] [ 76 2]] Classification Report (Testing Data): precision recall f1-
score support 0 0.96 1.00 0.98 1968 1 0.67 0.03 0.05 78 accuracy 0.96 2046
macro avg 0.81 0.51 0.52 2046 weighted avg 0.95 0.96 0.95 2046
```

Figure 16

```
# import required modules
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

class_names=[0,1] # name of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(train_conf_matrix), annot=True,
cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Training data confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

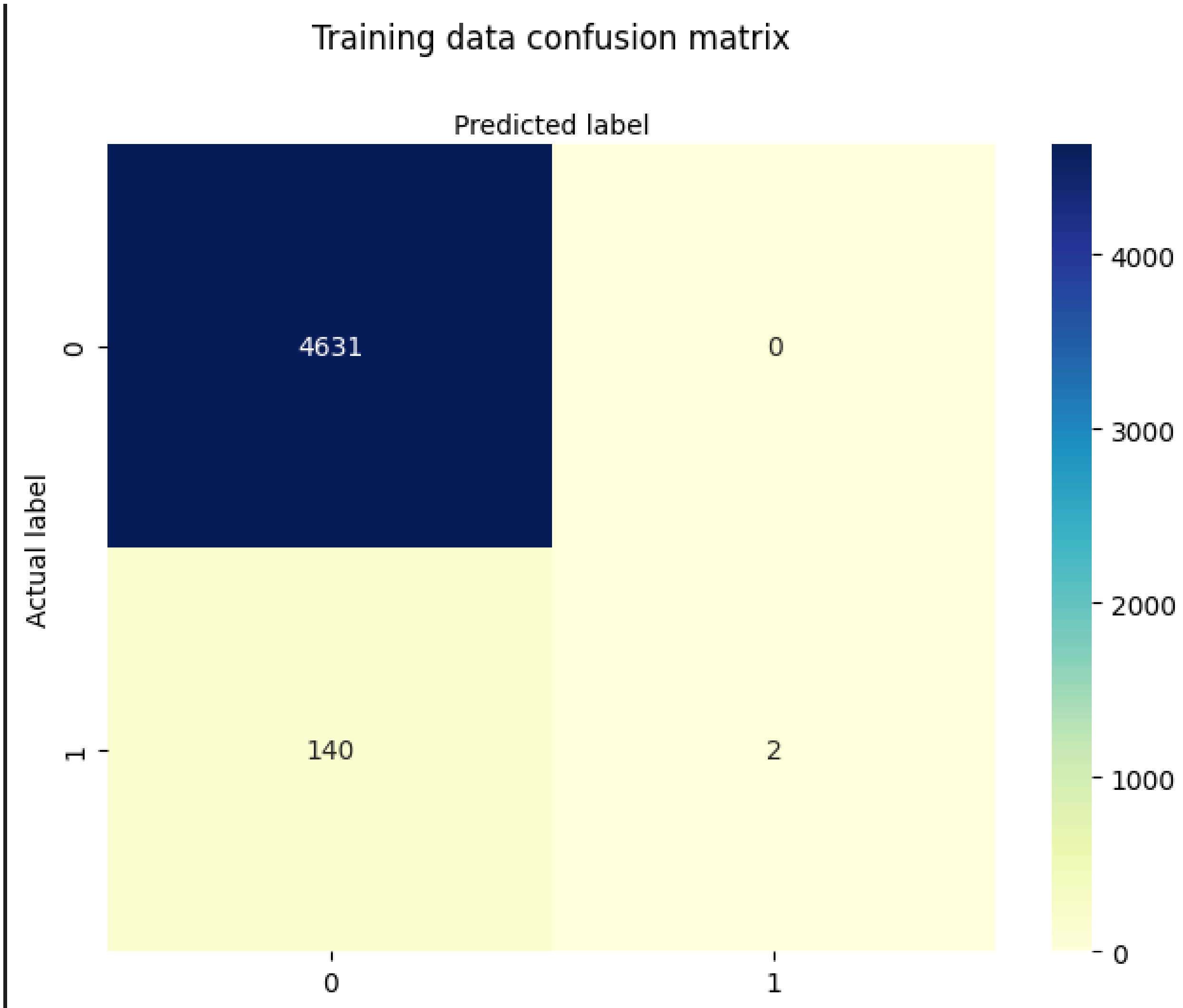


Figure 17

```
# import required modules
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

class_names=[0,1] # name of classes
fig, ax = plt.subplots()
```

```
tick_marks = np.arange(len(class_names))

plt.xticks(tick_marks, class_names)

plt.yticks(tick_marks, class_names)

# create heatmap

sns.heatmap(pd.DataFrame(train_conf_matrix), annot=True,

cmap="YlGnBu" ,fmt='g')

ax.xaxis.set_label_position("top")

plt.tight_layout()

plt.title('Training data confusion matrix', y=1.1)

plt.ylabel('Actual label')

plt.xlabel('Predicted label')
```

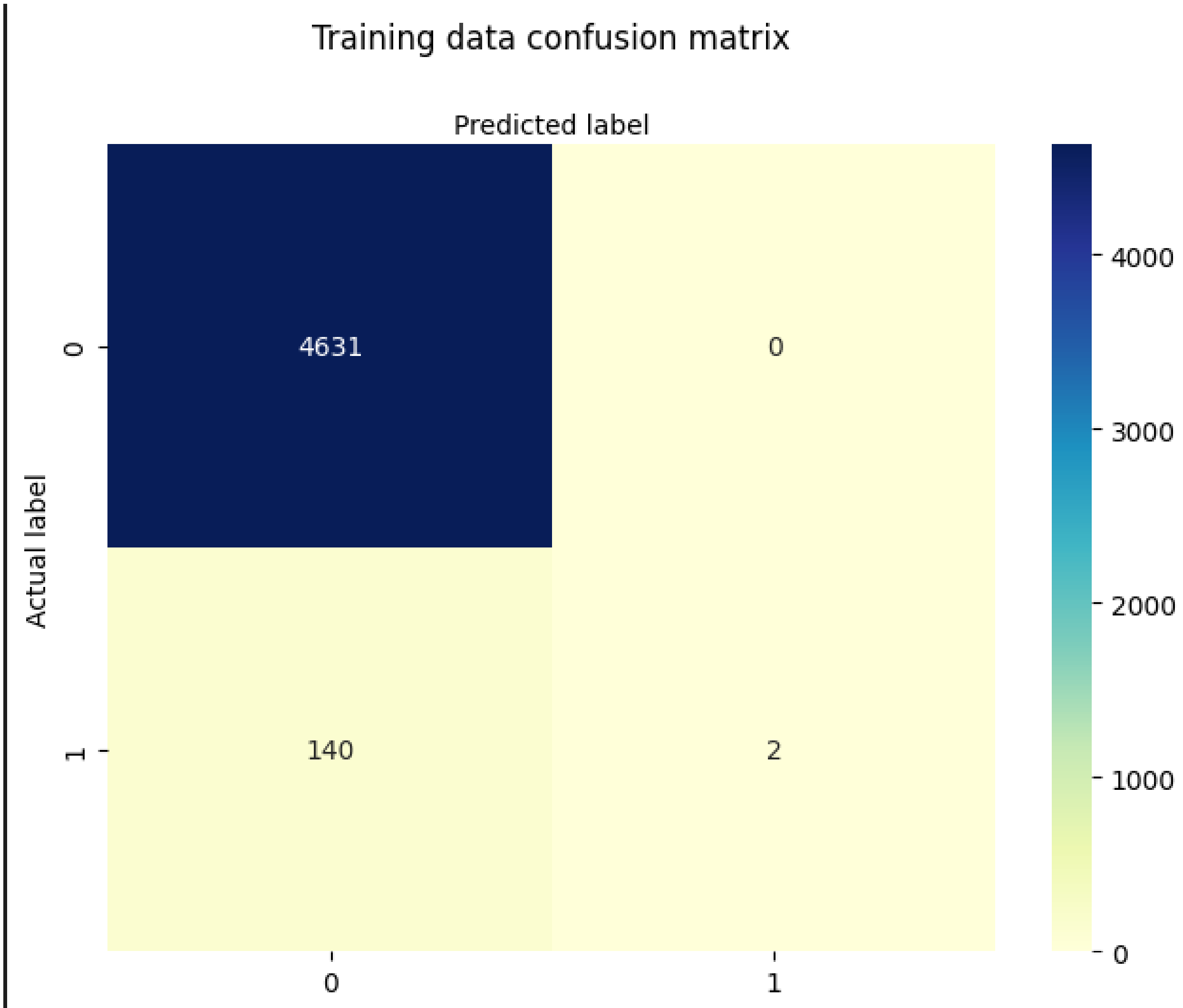


Figure 18

```
# import required modules
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

class_names=[0,1] # name of classes
fig, ax = plt.subplots()
```

```
tick_marks = np.arange(len(class_names))

plt.xticks(tick_marks, class_names)

plt.yticks(tick_marks, class_names)

# create heatmap

sns.heatmap(pd.DataFrame(test_conf_matrix), annot=True,

cmap="YlGnBu" ,fmt='g')

ax.xaxis.set_label_position("top")

plt.tight_layout()

plt.title('Test data confusion matrix', y=1.1)

plt.ylabel('Actual label')

plt.xlabel('Predicted label')
```

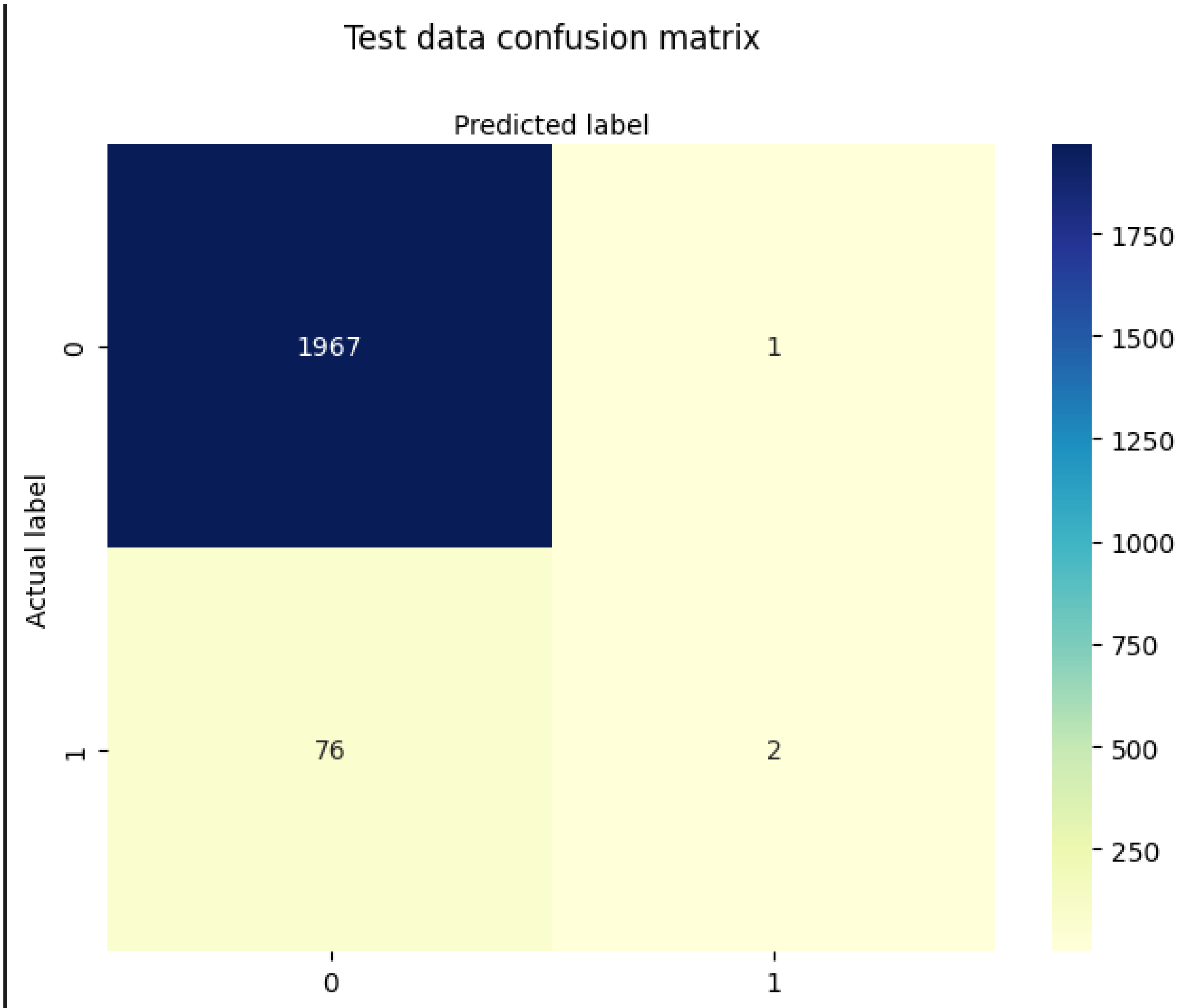


Figure 19

ROC Curves

```
import numpy as np
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

# SVM
```

```
fpr, tpr, thresholds = roc_curve(y_test, svm_y_test_pred)

plt.plot(fpr, tpr, linewidth=2, label="ROC")

plt.plot([0, 1], [0, 1], 'k--')

plt.show()
```

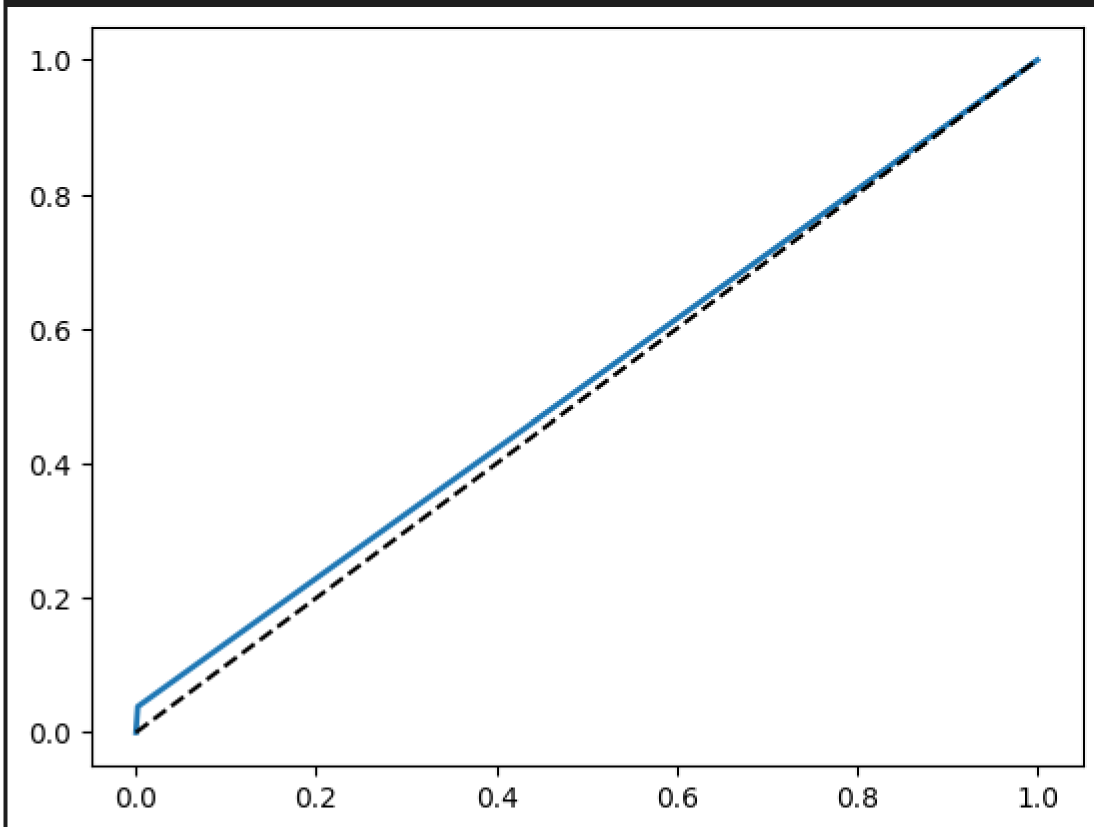


Figure 20

```
# Logistic Regression

fpr, tpr, thresholds = roc_curve(y_test, log_reg_y_test_pred)

plt.plot(fpr, tpr, linewidth=2, label="ROC")

plt.plot([0, 1], [0, 1], 'k--')

plt.show()
```

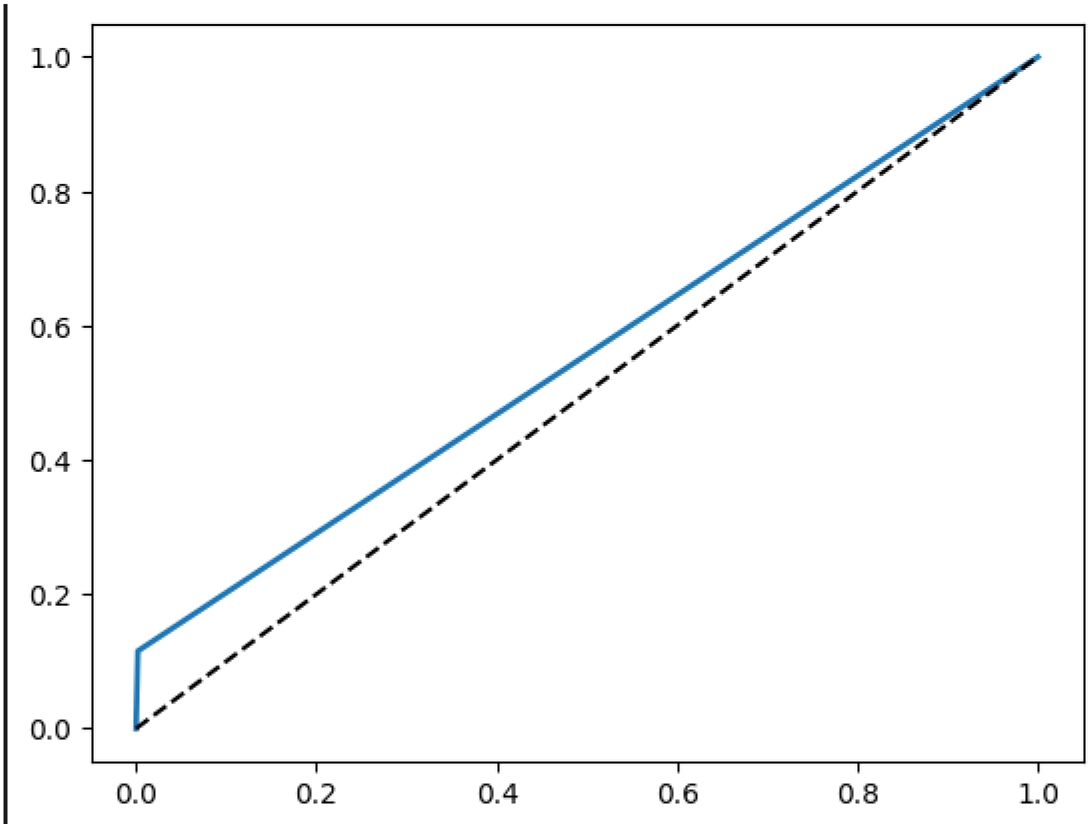


Figure 21

```
# Naive Bayes

fpr, tpr, thresholds = roc_curve(y_test, nb_y_test_pred)

plt.plot(fpr, tpr, linewidth=2, label="ROC")

plt.plot([0, 1], [0, 1], 'k--')

plt.show()
```

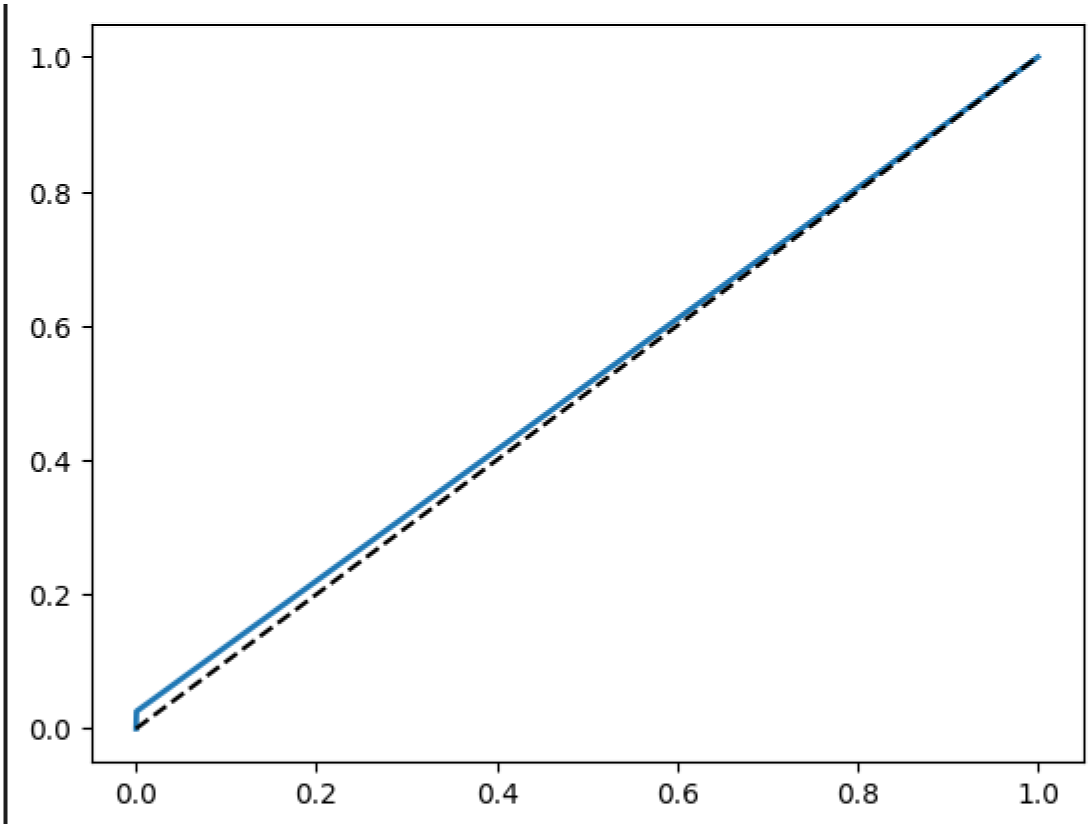


Figure 22

Precision/Recall

```
from sklearn.metrics import precision_recall_curve


# SVM
precision, recall, thresholds = precision_recall_curve(y_test,
svm_y_test_pred)
plt.plot(recall, precision, linewidth=2, label="Precision vs Recall")
plt.plot([0, 1], [0, 1], 'k--')
plt.show()
```

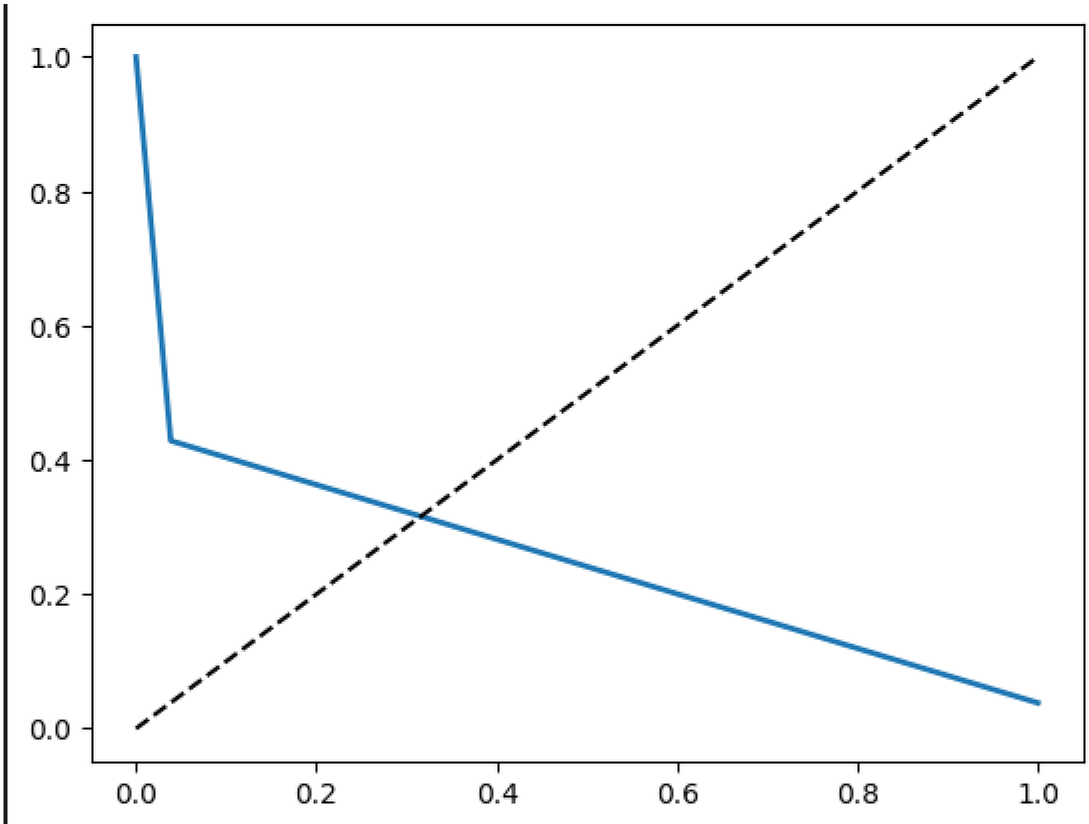


Figure 23

```
# Logistic Regression
precision, recall, thresholds = precision_recall_curve(y_test,
log_reg_y_test_pred)
plt.plot(recall, precision, linewidth=2, label="Precision vs Recall")
plt.plot([0, 1], [0, 1], 'k--')
plt.show()
```

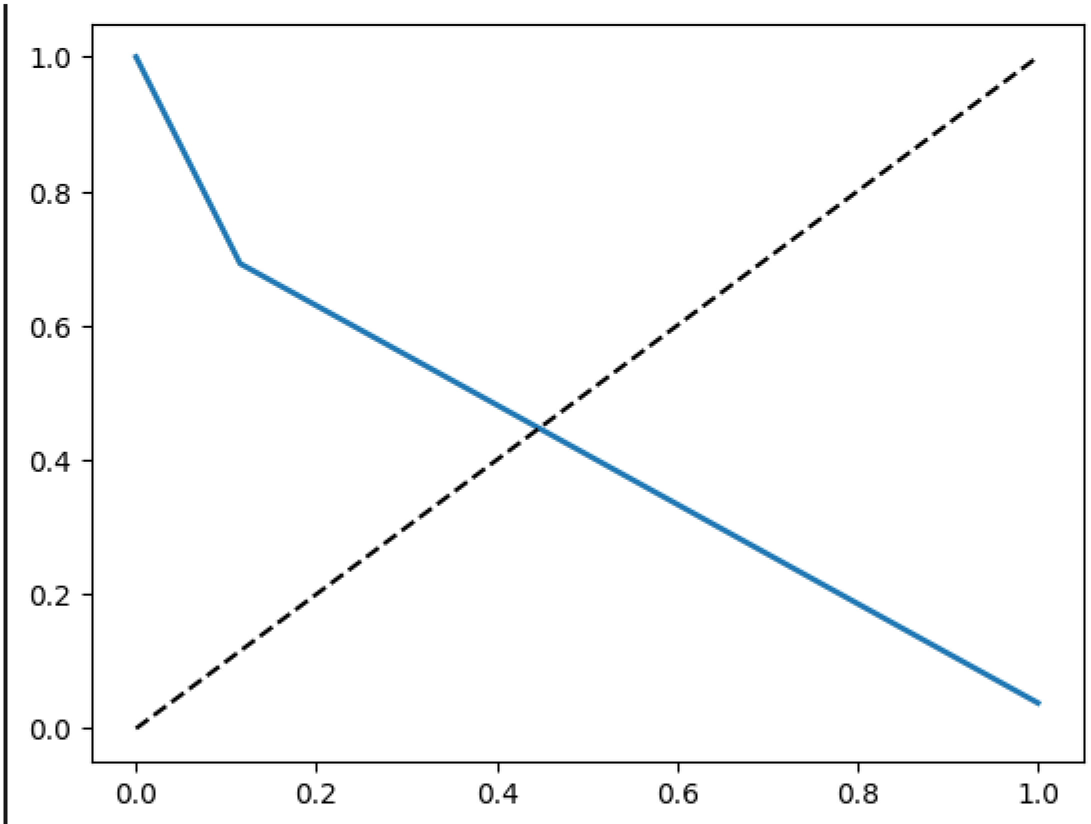


Figure 24

```
# Naive Bayes
precision, recall, thresholds = precision_recall_curve(y_test,
nb_y_test_pred)
plt.plot(recall, precision, linewidth=2, label="Precision vs Recall")
plt.plot([0, 1], [0, 1], 'k--')
plt.show()
```

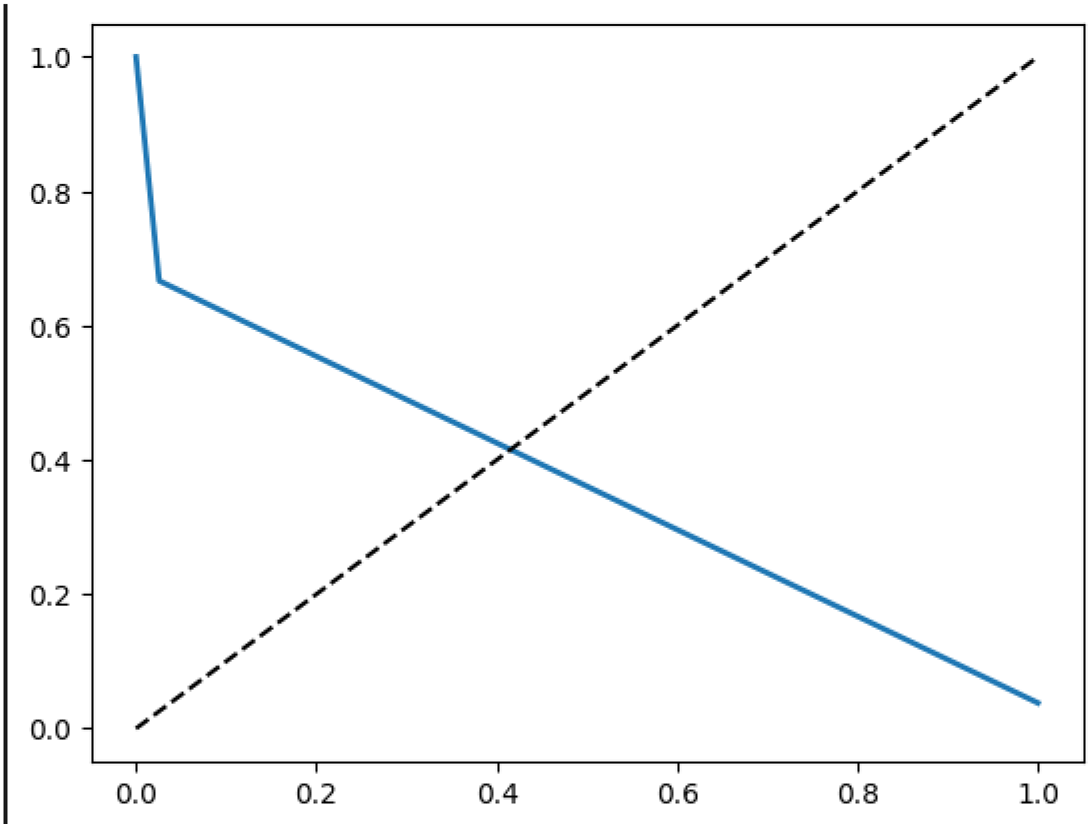


Figure 25

F1 Scores

```
from sklearn.metrics import f1_score

f1_score(y_test, svm_y_test_pred)

0.07058823529411765

f1_score(y_test, log_reg_y_test_pred)

0.19780219780219782
```

```
f1_score(y_test, nb_y_test_pred)
```

```
0.04938271604938271
```

Figure 26