Company Bankruptcy Prediction: Random Forest Classifier, Gradient Boosted Trees, Extra Trees

Previously we explored how models like logistic regression, SVMs, and Naive Bayes were able to predict whether a company is bankrupt. Now we use other models to get more accurate results. We use Random Forest Classifiers, Gradient Boosted Trees, and Extra Trees to predict the future for these companies. 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.

Extra Trees, short for Extremely Randomized Trees, is an ensemble decision tree method which constructs a multitude of decision trees during training and outputs the class that is the mode of the classes (classification problem) or mean prediction (regression problem) of the individual trees. There are several benefits: 1. It randomly selects a subset of features at each node to consider for splitting, adding an extra layer of randomness to the model. 2. For each selected feature, it also randomly selects the split point rather than choosing the optimal one based on some criterion increasing the diversity among the trees. 3. During prediction, each tree in the ensemble predicts the outcome, and the final prediction is typically made by taking the majority vote (for classification) or the average (for regression) of all the individual tree predictions.

After hyperparameter tuning, we got the best model with {'criterion': 'entropy', 'max_depth': 20, 'max_features': 'log2', 'n_estimators': 20}. The training result shows a good recall score of 0.92 and a precision of 1, indicating it captures most of the positive bankruptcy cases in the training set. However, we only get 9 correct bankruptcy predictions out of 78 in the test set. There is a recall score of 0.12 and a precision of 0.64. This means for when the model predicts bankruptcy, 64% are true, and for those actually bankrupt, the model only captures 12% showing it is moderately accurate but incomprehensive. The F-1 score is low at 0.2.

Random Forest Classifiers use collections of decision trees and are well suited for large datasets. Figures 7 and 8 show the confusion matrices and Figures 9-12 show the ROC and precision/recall curves along with an F1 score of 0.19. The next classifier, Gradient Boosting, tries to improve upon the previous model based on some sort of measurement. Figures 13-14 show that overfitting on the training dataset definitely occurred. However the curves and F1 score in Figures 15-18 shows a 0.25 score which was better than random forest.

Some main reasons for the lack of performance can be: 1. Imbalanced data with non-bankruptcy class much more prevalent. 2. Hyperparameter tuning may be inadequate, with more resources we can try more. 3.Insufficient data: Our dataset does not have enough bankruptcy cases for the model to be effectively trained.

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')
```

```python
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()
```

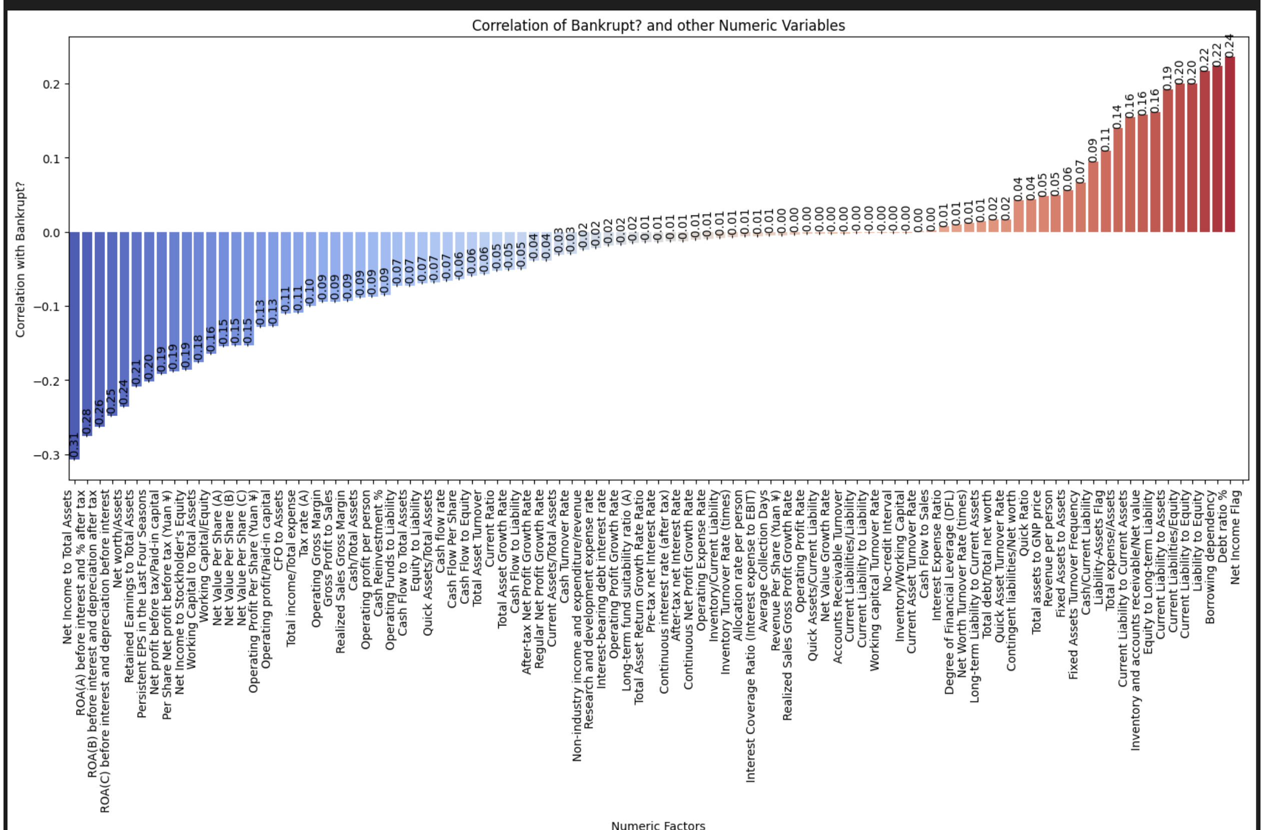

Figure 2

```python
#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()
```

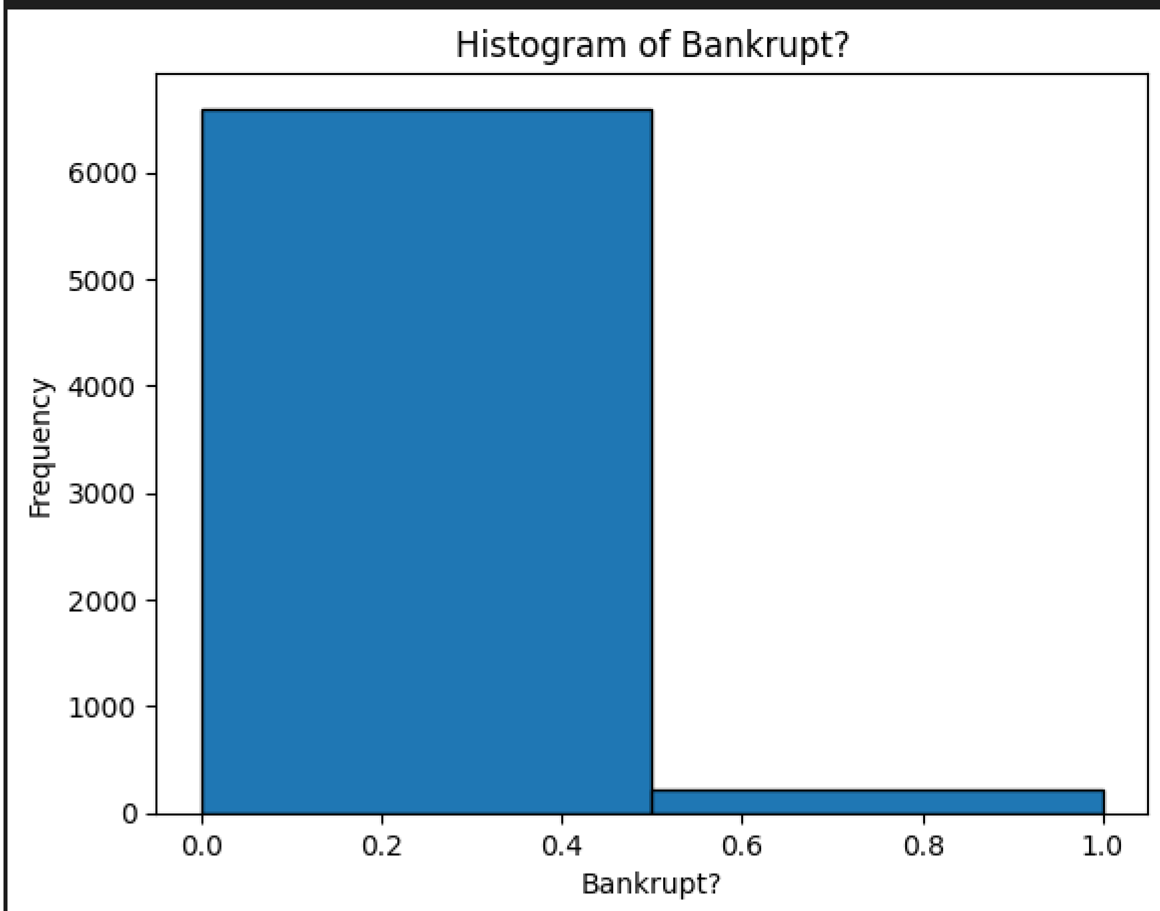

Figure 3

```python
#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()
```

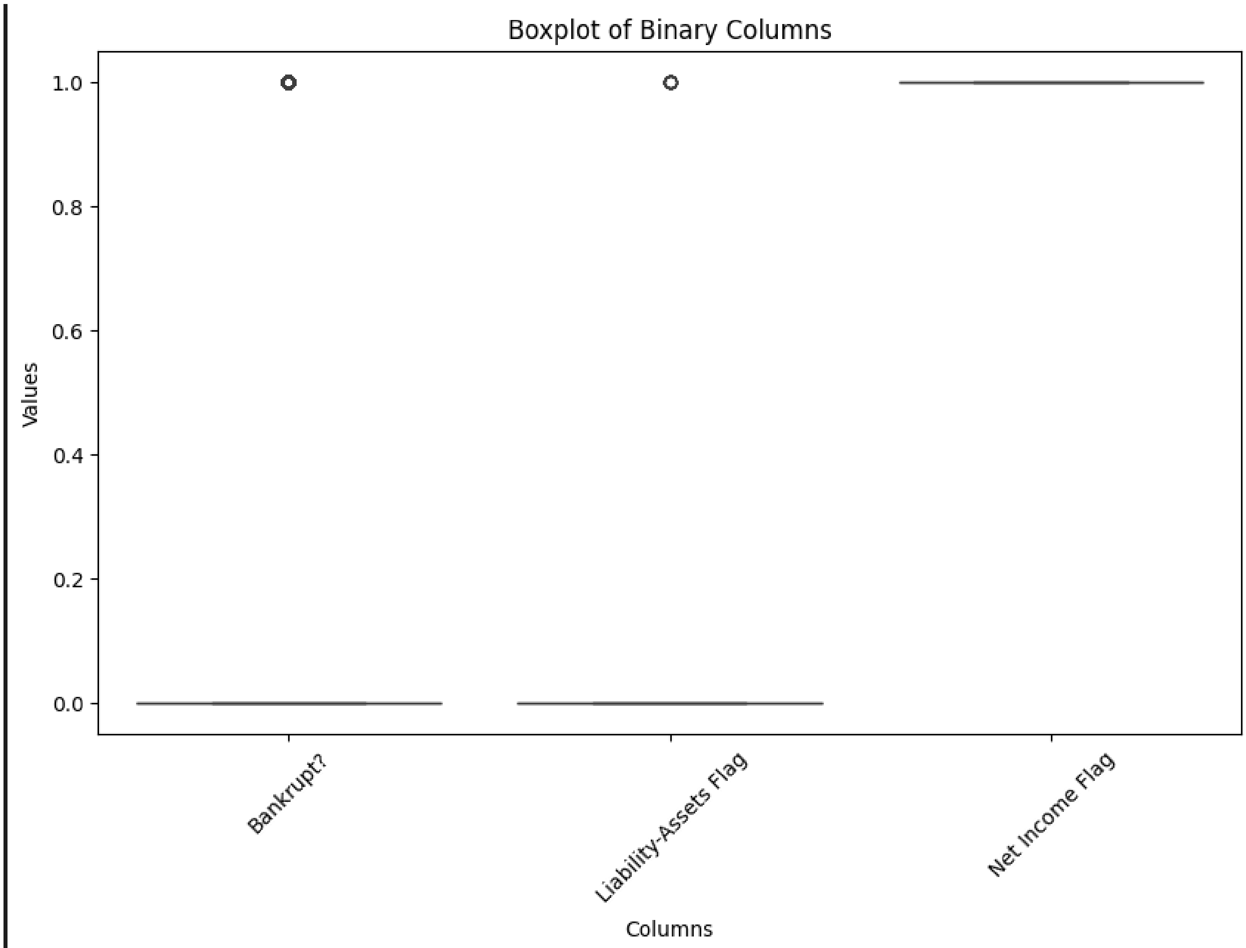

Figure 4

```
#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)
```

```python
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()
```

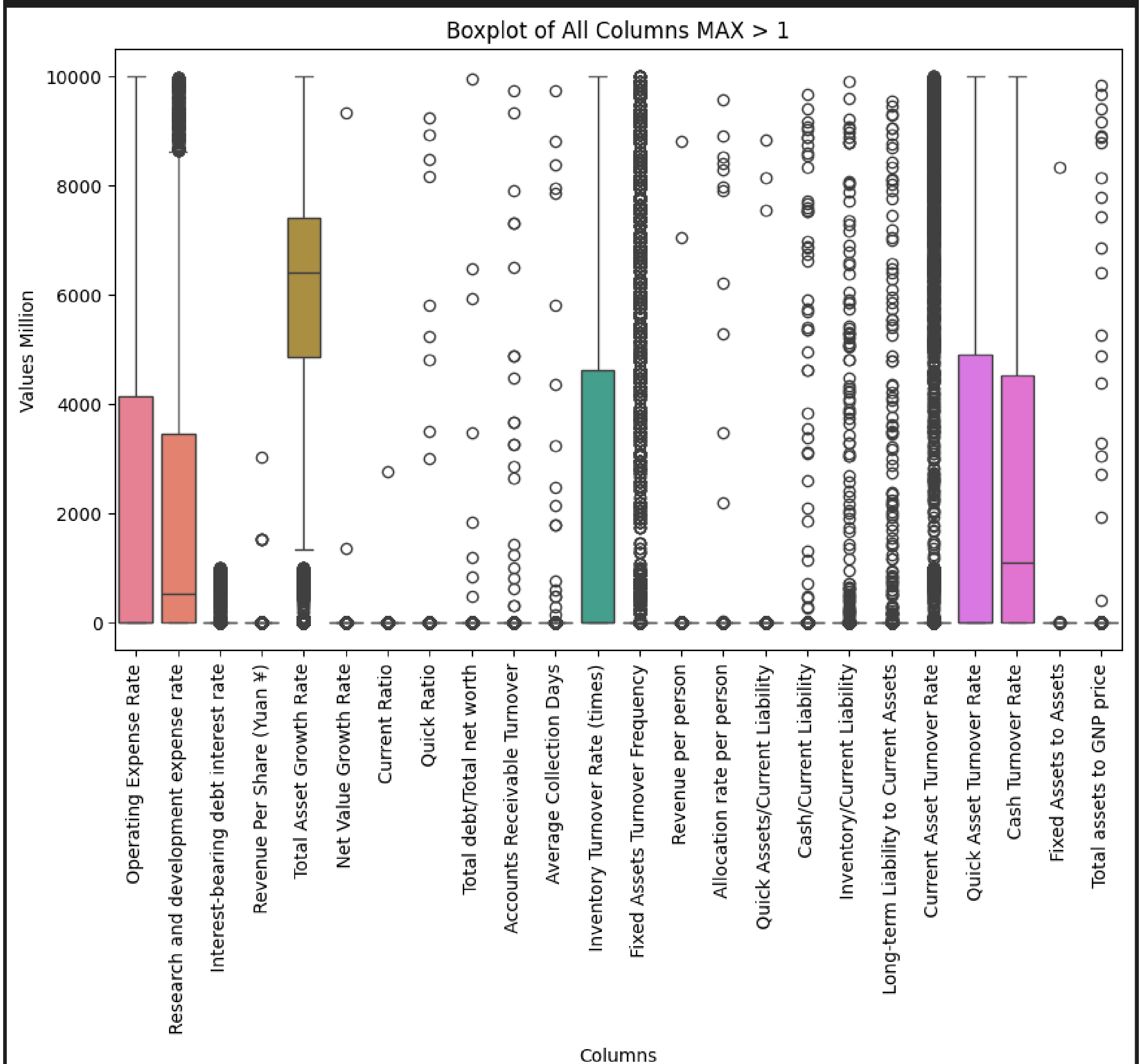

Figure 5

```
#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()
```

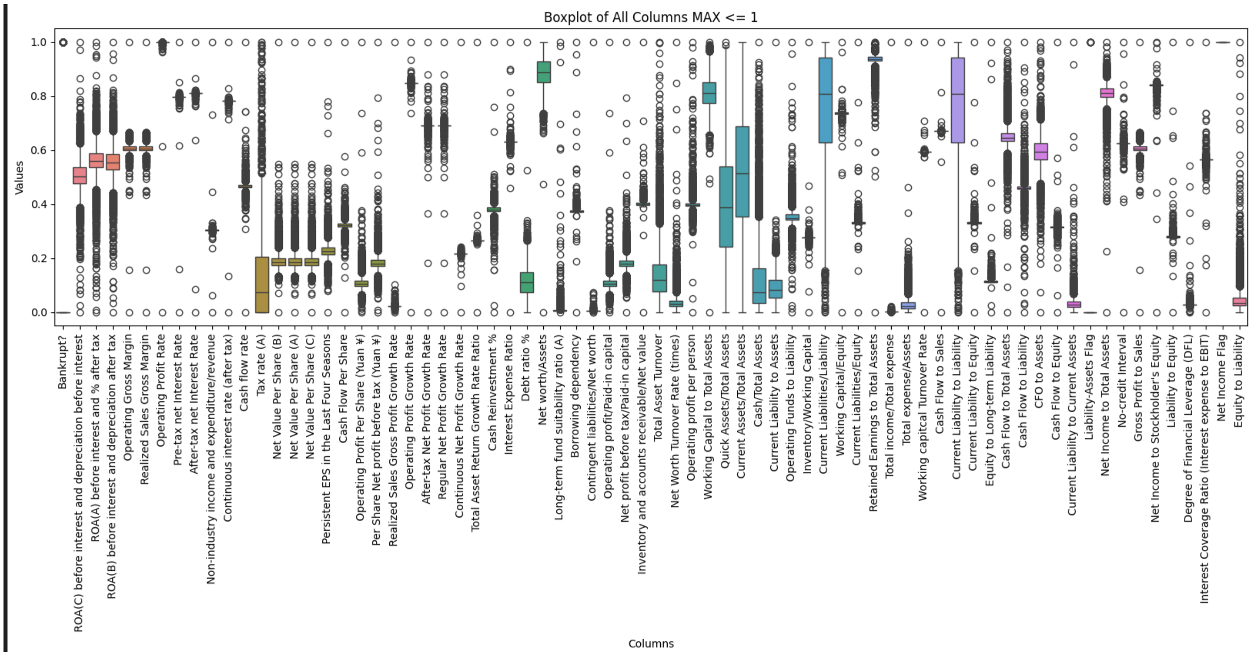

Figure 6

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, KFold
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)


rand_forest = RandomForestClassifier()

# Parameter Grid
param_grid = {
'n_estimators': [10, 100],
'max_features': ['sqrt', 'log2'],
```

```python
'max_depth': [2, 4, 6, 8, 10],
'criterion': ['gini', 'entropy', 'log_loss']
}

# Grid Search
kf = KFold(n_splits=5, shuffle=True, random_state=42)
grid_search = GridSearchCV(rand_forest, 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_rand_forest = RandomForestClassifier(**best_params)
tuned_rand_forest.fit(x_train_scaled, y_train)

# hyperparameter tuning
# 1. n_estimators (number of trees)
# 2. max_features (maximum features considered for splitting a node)
# 3. max_depth (maximum number of levels in each tree)
# 4. splitting criteria (entropy or gini)
```

RandomForestClassifier

```
RandomForestClassifier(max_depth=6)
```

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

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

```
Confusion Matrix (Training Data): [[4631 0] [ 84 58]] Classification
Report (Training Data): precision recall f1-score support 0 0.98 1.00 0.99
4631 1 1.00 0.41 0.58 142 accuracy 0.98 4773 macro avg 0.99 0.70 0.79 4773
weighted avg 0.98 0.98 0.98 4773 Confusion Matrix (Testing Data): [[1965
3] [ 69 9]] Classification Report (Testing Data): precision recall f1-
score support 0 0.97 1.00 0.98 1968 1 0.75 0.12 0.20 78 accuracy 0.96 2046
macro avg 0.86 0.56 0.59 2046 weighted avg 0.96 0.96 0.95 2046# import
required modules for performance evaluation
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score


# Training Data confusion matrix for random forest
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')
```

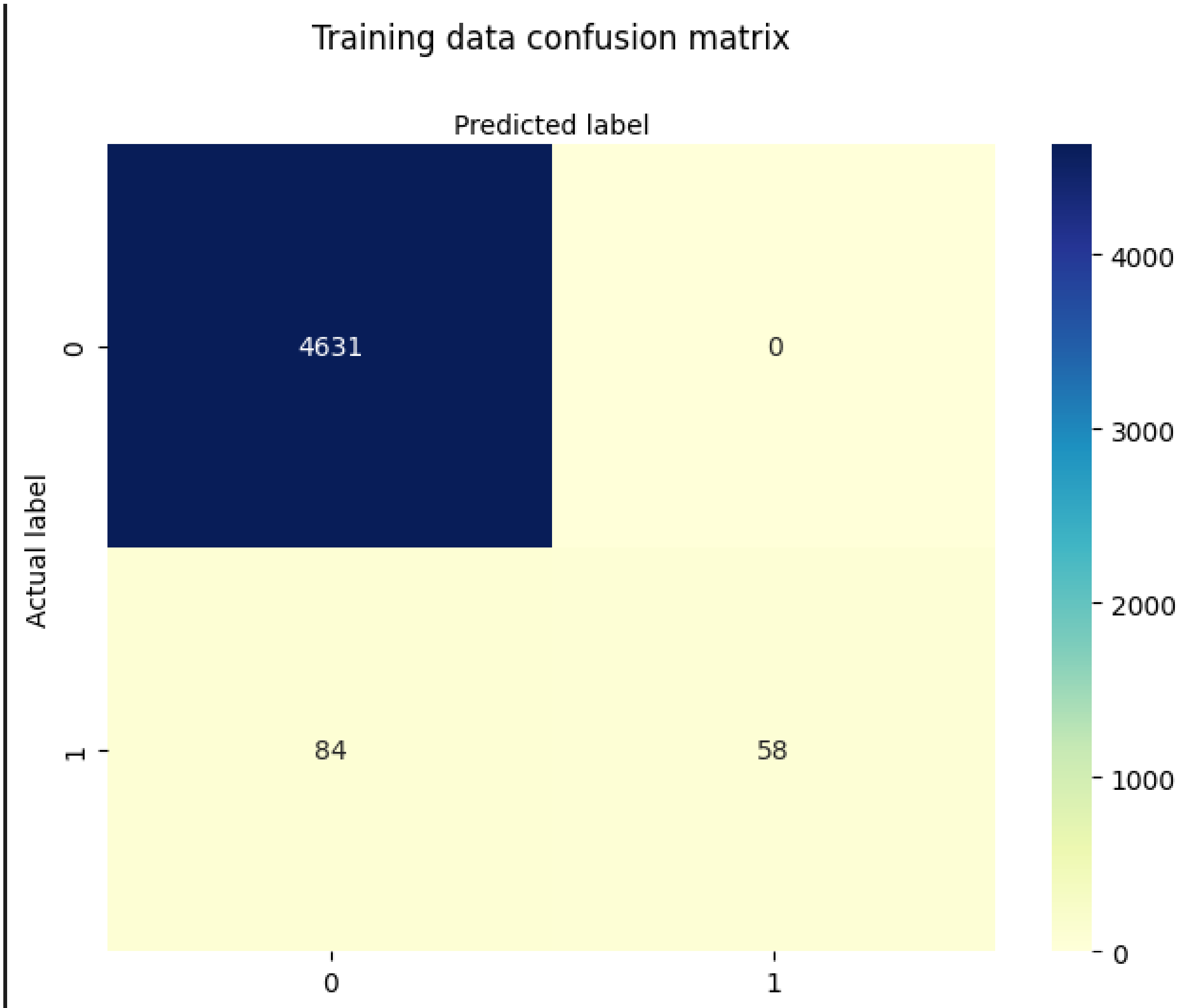

Figure 7

```
# Testing Data confusion matrix for random forest
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')
```

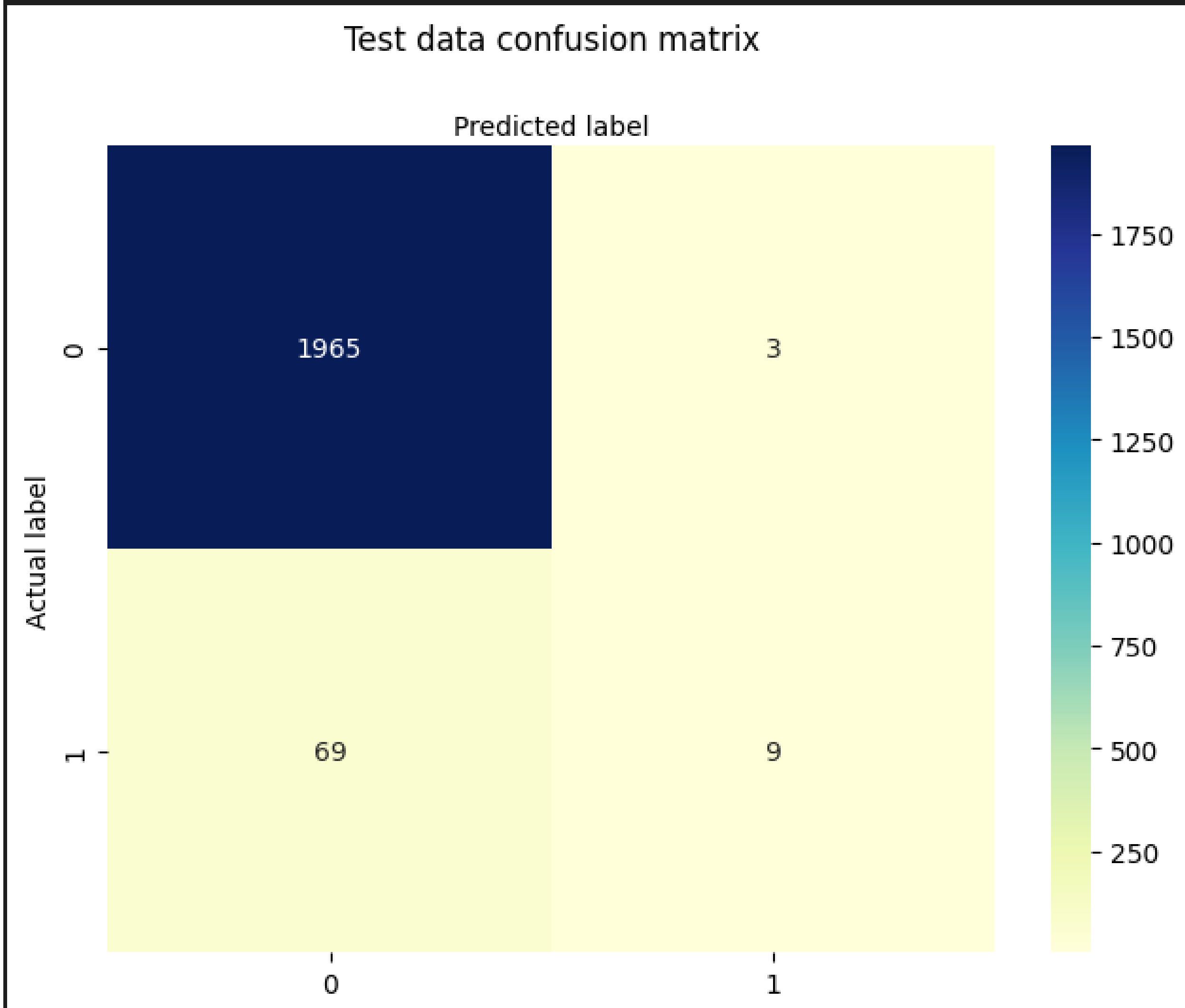

Figure 8

```
# Random Forest ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, rand_forest_y_test_pred)
plt.plot(fpr, tpr, linewidth=2, label="ROC")
plt.plot([0, 1], [0, 1], 'k--')
plt.show()
```

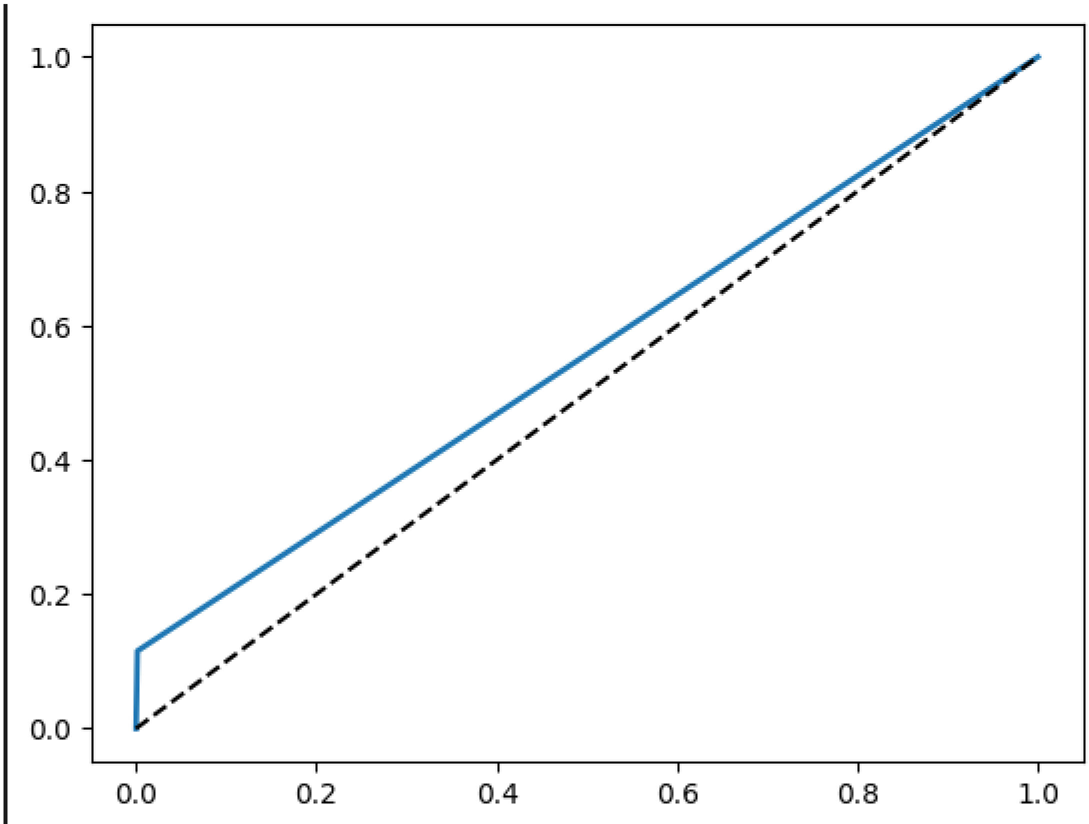

Figure 9

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

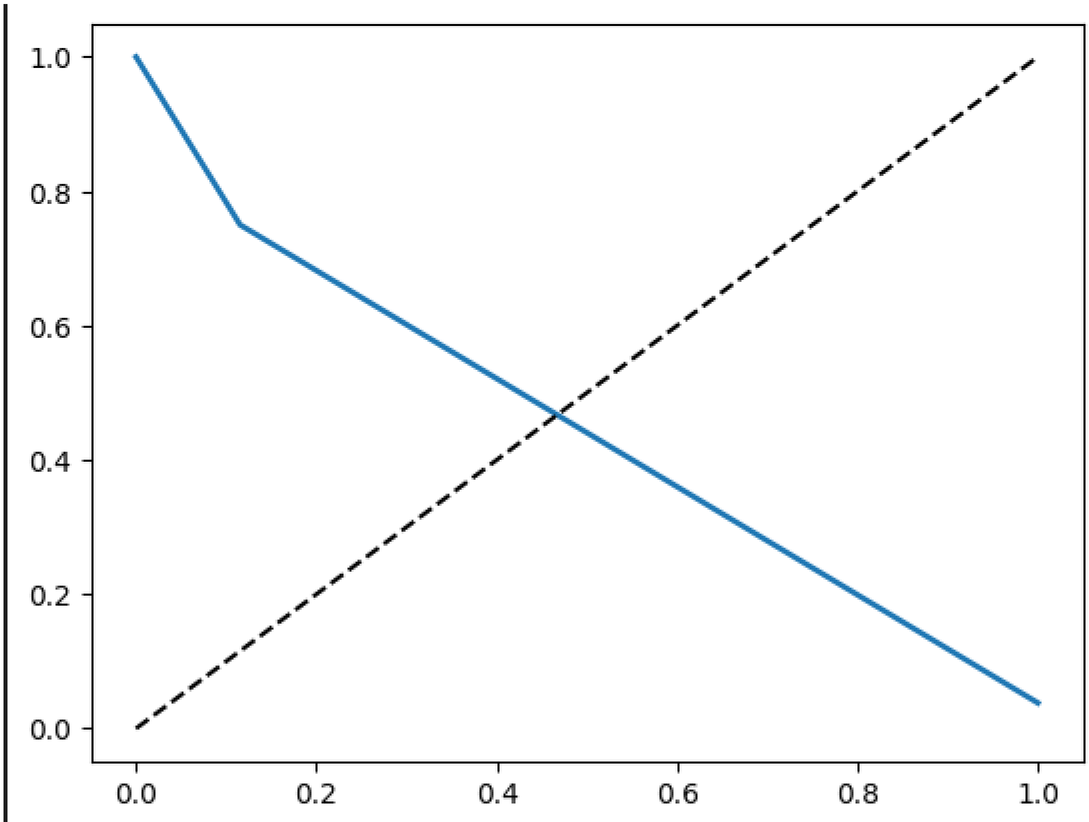

Figure 10

```
# Random Forest F1 Score
f1_score(y_test, rand_forest_y_test_pred)
```

0.19999999999999998

Figure 11

```
# Random Forest Best Params
best_params
```

{'criterion': 'gini',

'max_depth': 6,

'max_features': 'sqrt',

'n_estimators': 100}

Figure 12

Gradient Boosted Trees

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV, KFold
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)


grad_boost = GradientBoostingClassifier()

# Parameter Grid
param_grid = {
'n_estimators': [10, 100],
'max_features': ['sqrt', 'log2'],
'max_depth': [2, 4, 6, 8, 10],
'criterion': ['friedman_mse', 'squared_error'],
}

# Grid Search
kf = KFold(n_splits=5, shuffle=True, random_state=42)
grid_search = GridSearchCV(grad_boost, 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_grad_boost = GradientBoostingClassifier(**best_params)
tuned_grad_boost.fit(x_train_scaled, y_train)
```

```
GradientBoostingClassifier
```

```
GradientBoostingClassifier(max_depth=8, max_features='log2')
```

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

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

```
Confusion Matrix (Training Data): [[4631 0] [ 0 142]] Classification
Report (Training Data): precision recall f1-score support 0 1.00 1.00 1.00
4631 1 1.00 1.00 1.00 142 accuracy 1.00 4773 macro avg 1.00 1.00 1.00 4773
weighted avg 1.00 1.00 1.00 4773 Confusion Matrix (Testing Data): [[1962
6] [ 66 12]] Classification Report (Testing Data): precision recall f1-
score support 0 0.97 1.00 0.98 1968 1 0.67 0.15 0.25 78 accuracy 0.96 2046
macro avg 0.82 0.58 0.62 2046 weighted avg 0.96 0.96 0.95 2046
```

```
# import required modules for performance evaluation
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.metrics import roc_curve
```

```python
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score


# Training Data confusion matrix for gradient boost
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')
```

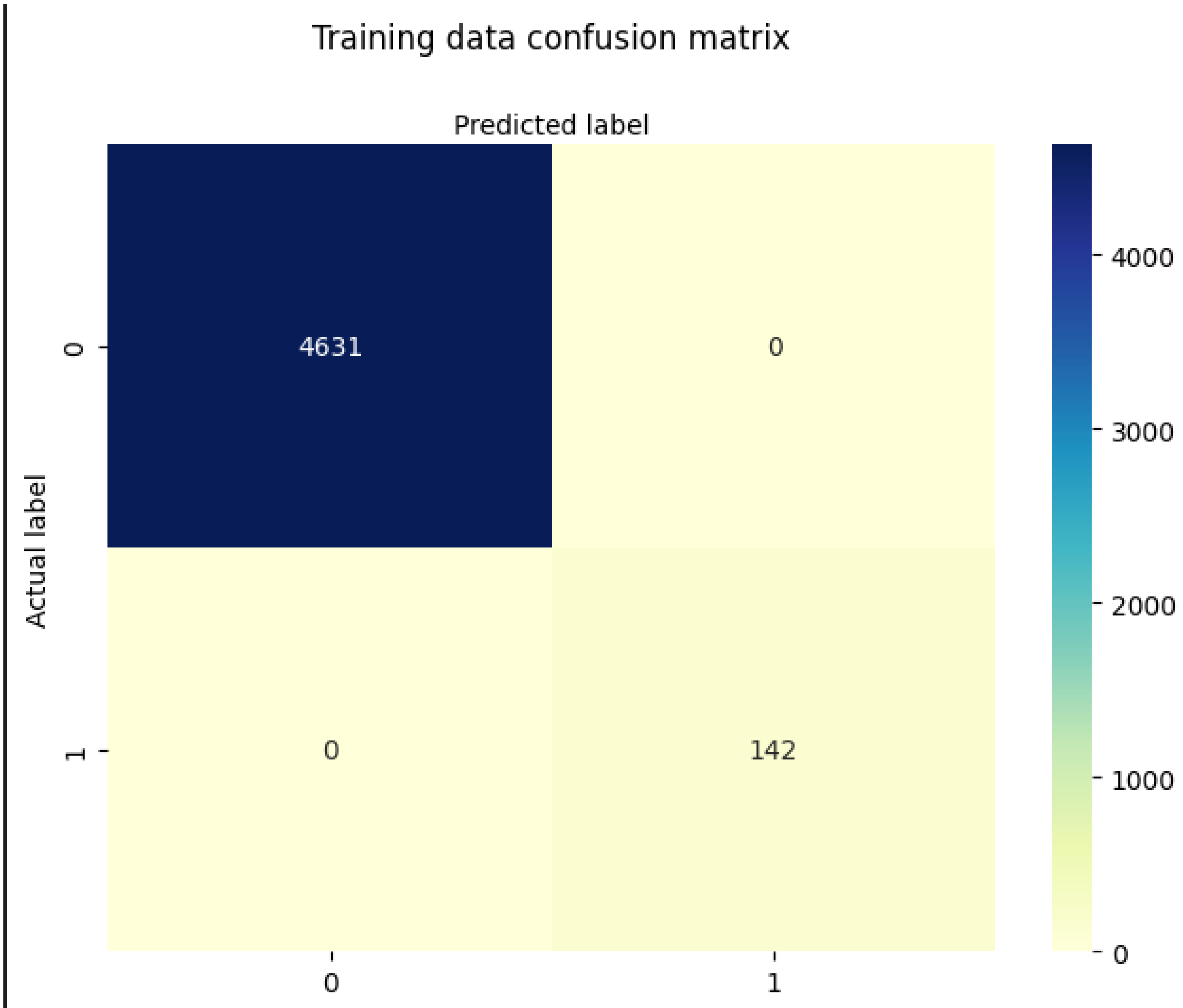

Figure 13

```
# Test Data confusion matrix for gradient boost
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')
```

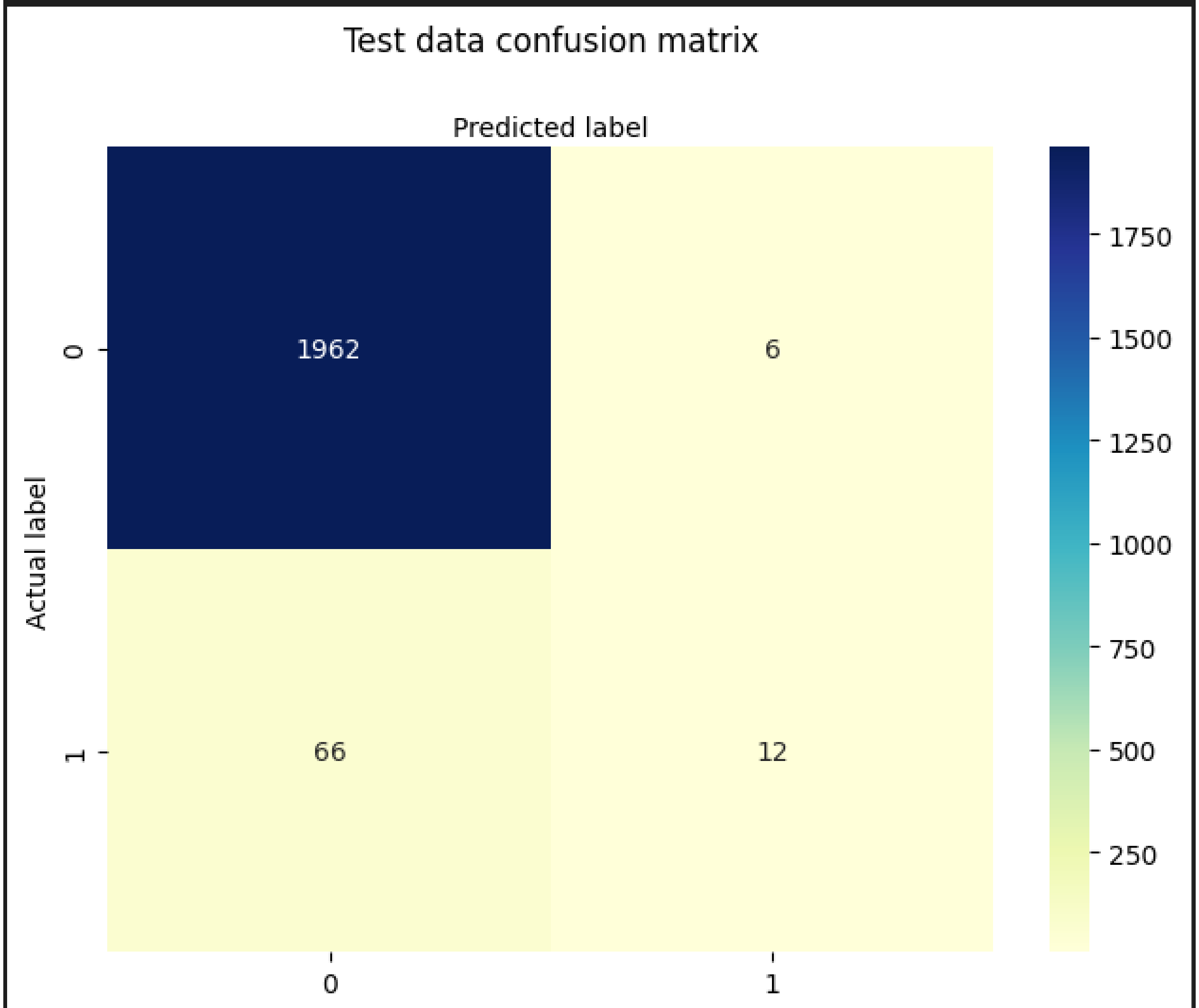

Figure 14

```
# Gradient Boosting ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, grad_boost_y_test_pred)
plt.plot(fpr, tpr, linewidth=2, label="ROC")
plt.plot([0, 1], [0, 1], 'k--')
plt.show()
```

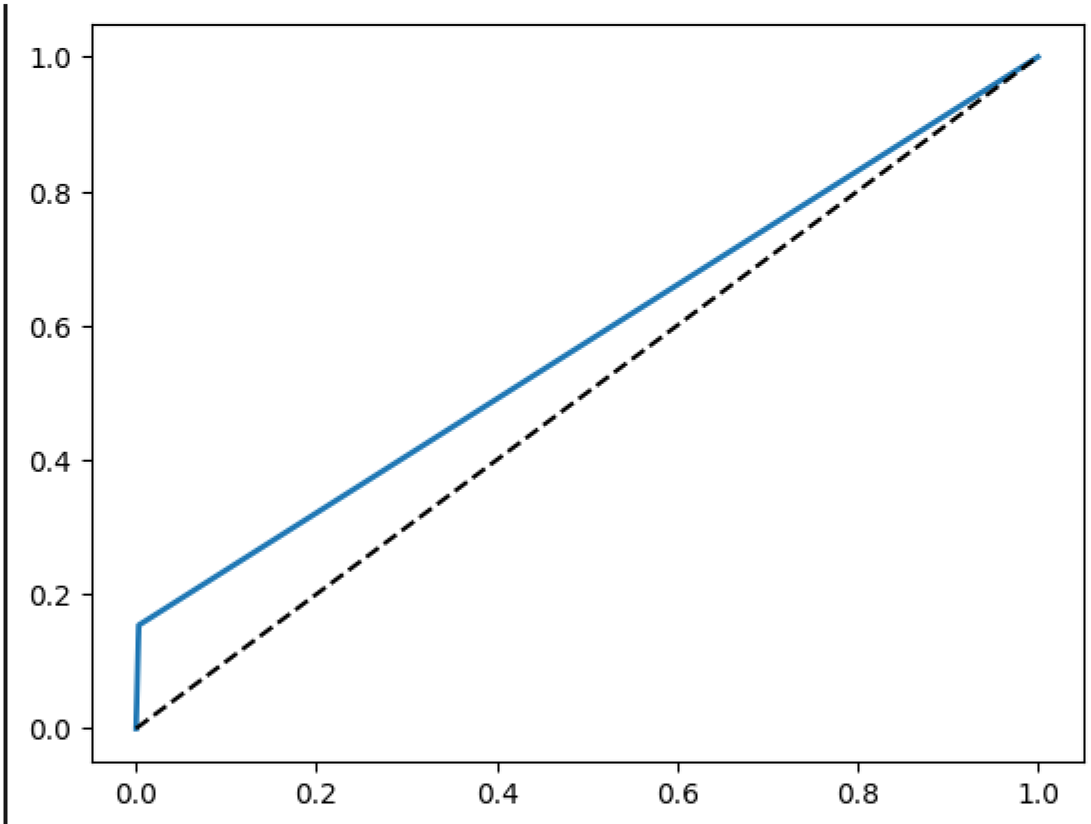

Figure 15

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

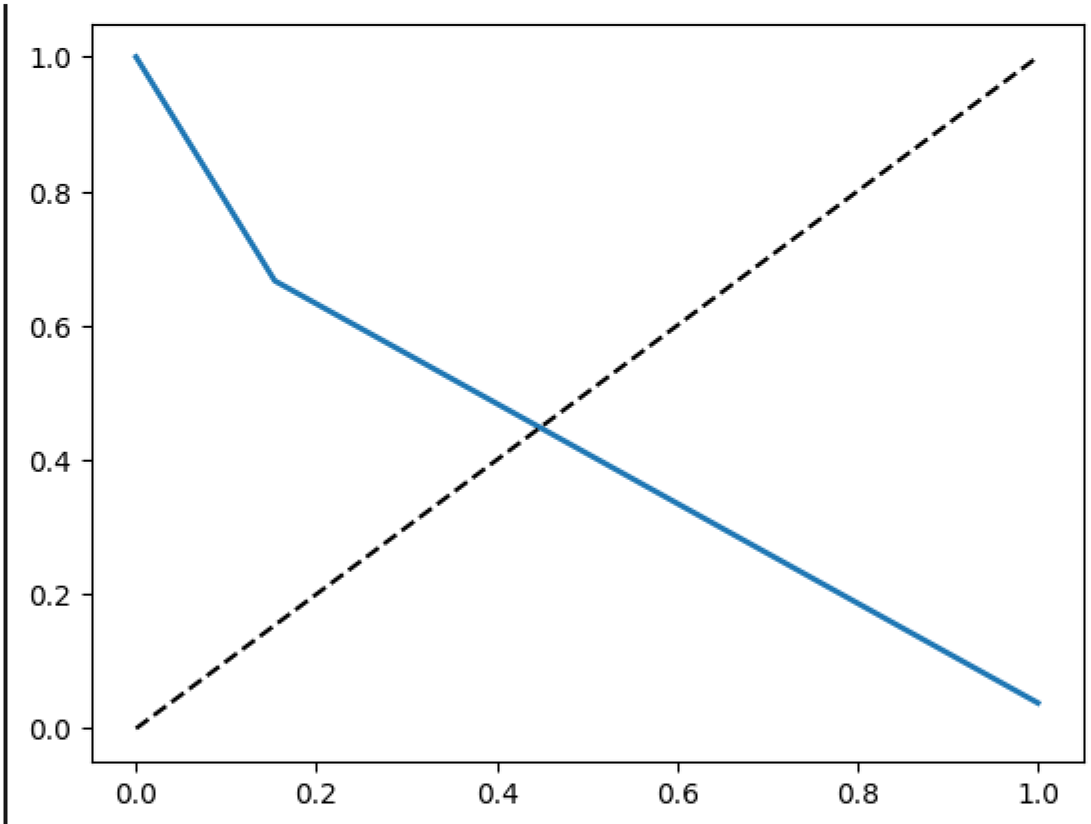

Figure 16

```
# Gradient Boosting F1 Score
f1_score(y_test, grad_boost_y_test_pred)
0.25
```

Figure 17

```
# Gradient Boosting Best Params
best_params

{'criterion': 'friedman_mse',
 'max_depth': 8,
 'max_features': 'log2',
 'n_estimators': 100}
```

Figure 18

Extra Trees

```
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import GridSearchCV, KFold
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)


extra_tree = ExtraTreesClassifier()

# Parameter Grid
param_grid = {
'n_estimators': [5,10,15,20,100],
'max_features': ['sqrt', 'log2'],
'max_depth': [2, 4, 6, 8, 10,12,14,16,18,20],
'criterion': ['gini', 'entropy','log_loss'],
}

# Grid Search
kf = KFold(n_splits=5, shuffle=True, random_state=42)
grid_search = GridSearchCV(extra_tree, 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_extra_tree = ExtraTreesClassifier(**best_params)
tuned_extra_tree.fit(x_train_scaled, y_train)
```

ExtraTreesClassifier

```
ExtraTreesClassifier(criterion='entropy', max_depth=20,

max_features='log2',
```

```
                          n_estimators=20)
```

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

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

```
Confusion Matrix (Training Data): [[4631 0] [ 12 130]] Classification
Report (Training Data): precision recall f1-score support 0 1.00 1.00 1.00
4631 1 1.00 0.92 0.96 142 accuracy 1.00 4773 macro avg 1.00 0.96 0.98 4773
weighted avg 1.00 1.00 1.00 4773 Confusion Matrix (Testing Data): [[1963
5] [ 69 9]] Classification Report (Testing Data): precision recall f1-
score support 0 0.97 1.00 0.98 1968 1 0.64 0.12 0.20 78 accuracy 0.96 2046
macro avg 0.80 0.56 0.59 2046 weighted avg 0.95 0.96 0.95 2046
```

```python
# import required modules for performance evaluation
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
```

```python
# Training Data confusion matrix for Extra Tree
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')
```

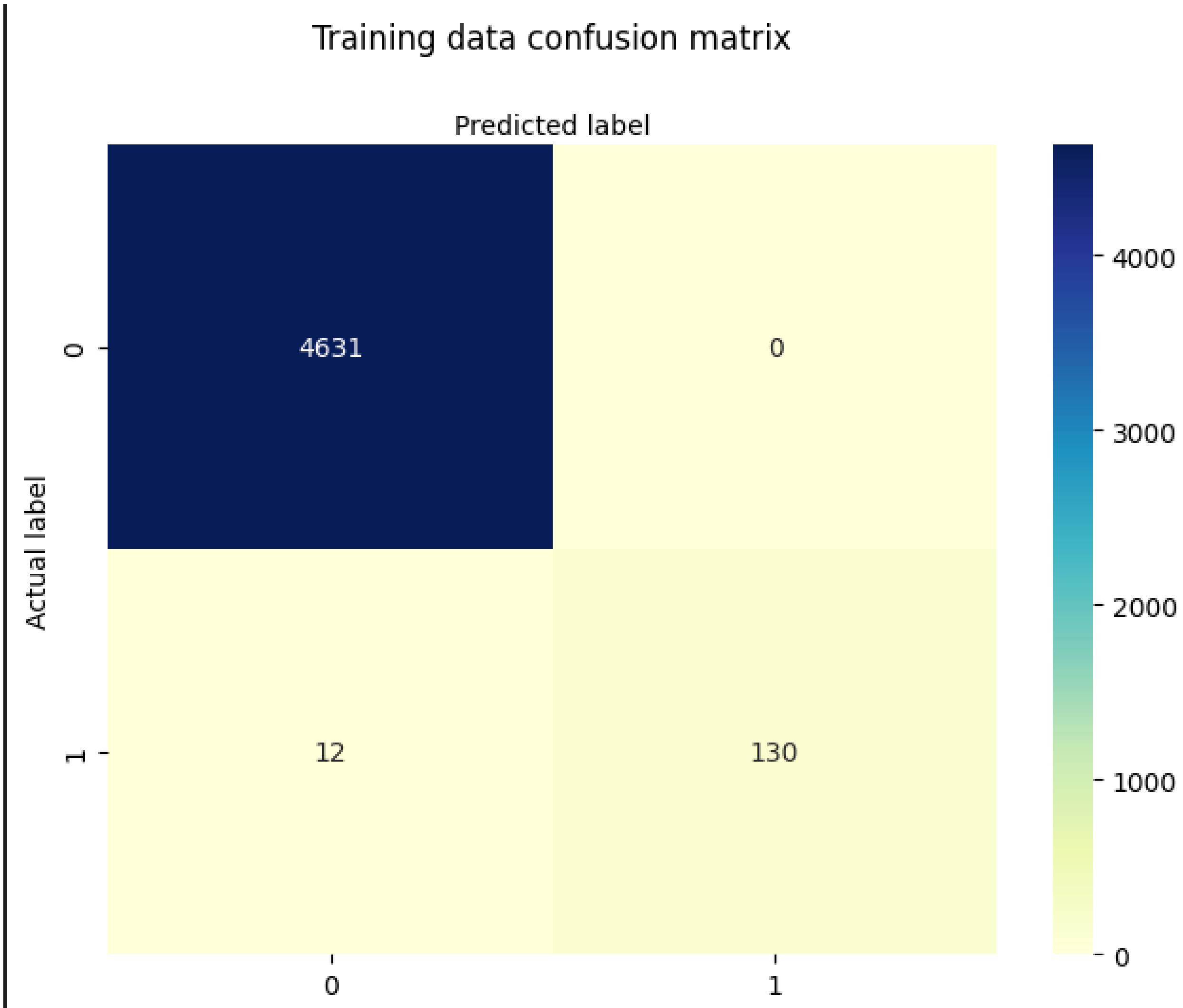

Figure 19

```
# Test Data confusion matrix for Extra Tree
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')
```

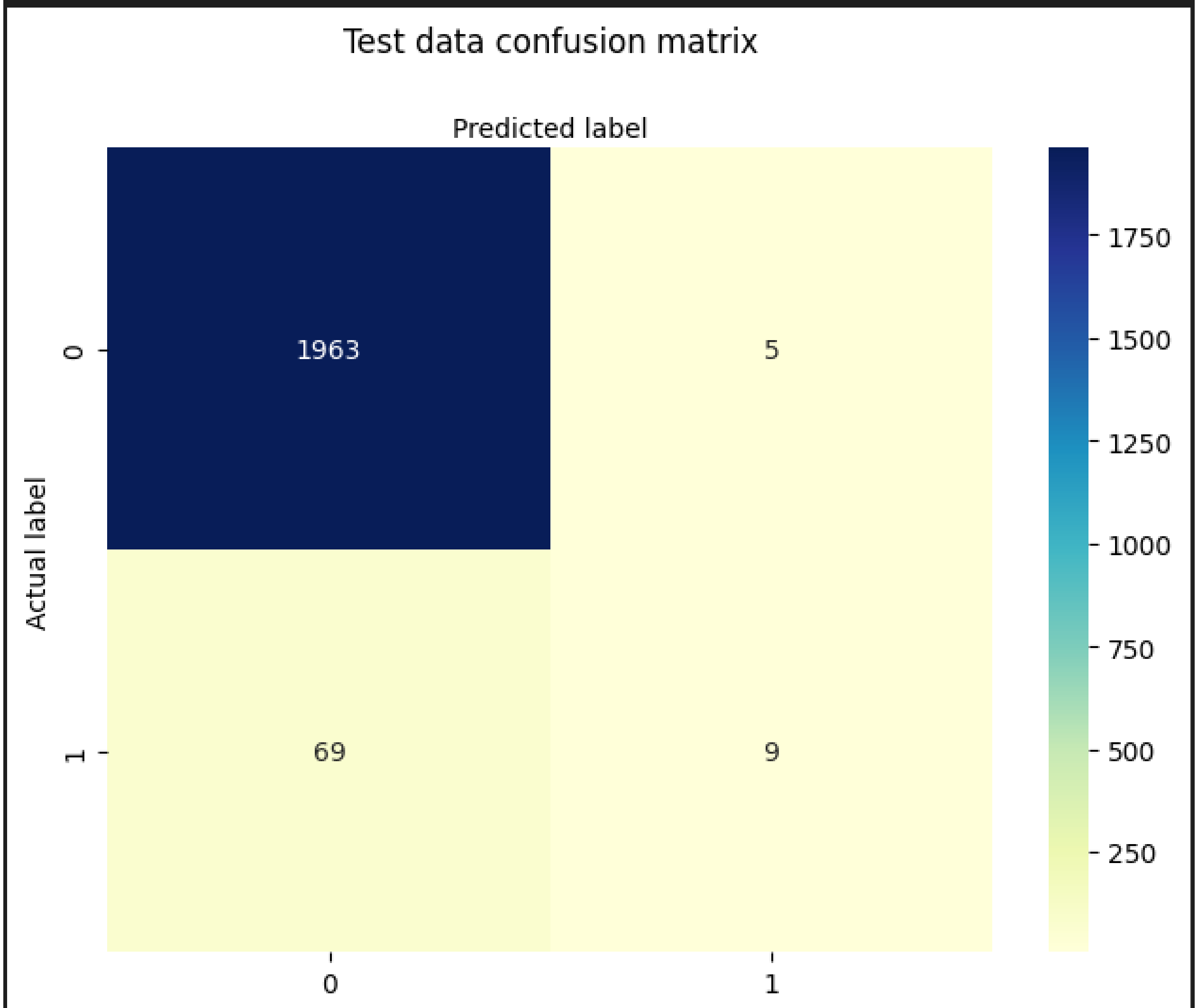

Figure 20

```
# Extra Tree ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, extra_tree_y_test_pred)
plt.plot(fpr, tpr, linewidth=2, label="ROC")
plt.plot([0, 1], [0, 1], 'k--')
plt.show()
```

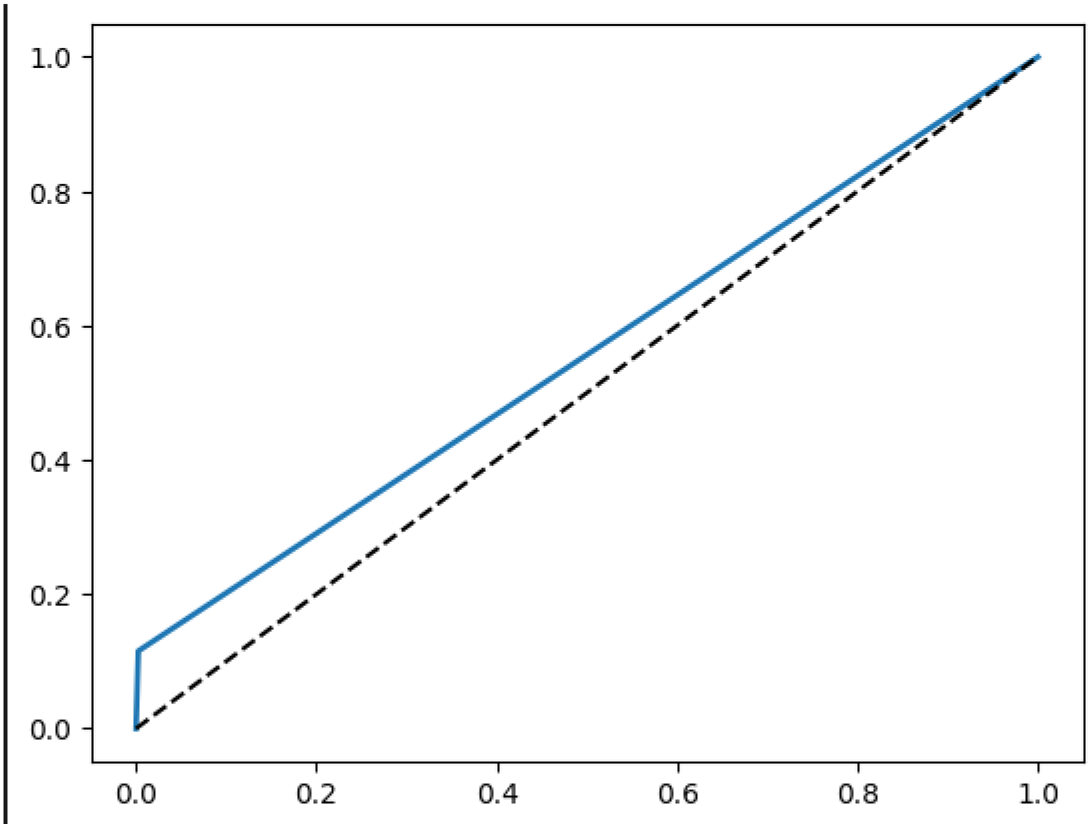

Figure 21

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

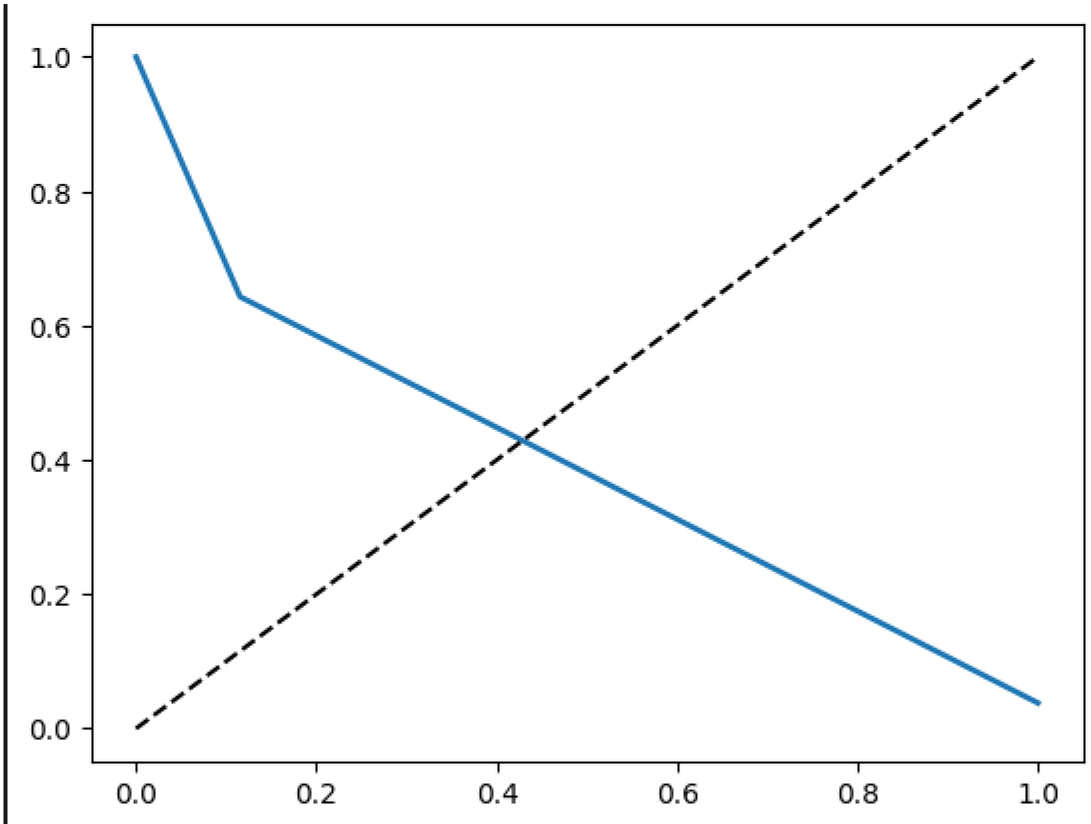

Figure 22

```
# Extra Tree F1 Score
f1_score(y_test, extra_tree_y_test_pred)
0.1956521739130435
```

Figure 23

```
# Extra Trees Best Params
best_params
{'criterion': 'entropy',
 'max_depth': 20,
 'max_features': 'log2',
 'n_estimators': 20}
```

Figure 24