liver_disease_prediction_randomforest_model

March 15, 2025

Final Project Liver Disease Prediction

Group 2: Selina Meng, Tina Zhang, Tianchuzi Qin, Yuyang Chen

1 Import

[63]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[64]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as stats
import numpy as np
from scipy.stats import zscore
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.decomposition import PCA
from mpl_toolkits.mplot3d import Axes3D
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import RFE
```

[172]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report, roc_curve, auc, precision_recall_curve
```

[65]:
```
df = pd.read_csv('/content/drive/MyDrive/MSDS422_FINAL/Liver Patient Dataset␣
 ↪(LPD)_train.csv', encoding='latin-1')
# or try 'ISO-8859-1' if 'latin-1' doesn't work
```

2 EDA

2.1 Data presentation

[66]:
```
df
```

[66]:

	Age of the patient	Gender of the patient	Total Bilirubin	\
0	65.0	Female	0.7	
1	62.0	Male	10.9	
2	62.0	Male	7.3	
3	58.0	Male	1.0	
4	72.0	Male	3.9	
…	…	…	…	
30686	50.0	Male	2.2	
30687	55.0	Male	2.9	
30688	54.0	Male	6.8	
30689	48.0	Female	1.9	
30690	30.0	Male	3.1	

	Direct Bilirubin	Alkphos Alkaline Phosphotase	\
0	0.1	187.0	
1	5.5	699.0	
2	4.1	490.0	
3	0.4	182.0	
4	2.0	195.0	
…	…	…	
30686	1.0	610.0	
30687	1.3	482.0	
30688	3.0	542.0	
30689	1.0	231.0	
30690	1.6	253.0	

	Sgpt Alamine Aminotransferase	Sgot Aspartate Aminotransferase	\
0	16.0	18.0	
1	64.0	100.0	
2	60.0	68.0	
3	14.0	20.0	
4	27.0	59.0	
…	…	…	
30686	17.0	28.0	
30687	22.0	34.0	
30688	116.0	66.0	
30689	16.0	55.0	
30690	80.0	406.0	

	Total Protiens	ALB Albumin	A/G Ratio Albumin and Globulin Ratio	\
0	6.8	3.3	0.90	
1	7.5	3.2	0.74	

```
2                 7.0           3.3                                    0.89
3                 6.8           3.4                                    1.00
4                 7.3           2.4                                    0.40
…                 …             …                                      …
30686             7.3           2.6                                    0.55
30687             7.0           2.4                                    0.50
30688             6.4           3.1                                    0.90
30689             4.3           1.6                                    0.60
30690             6.8           3.9                                    1.30

       Result
0           1
1           1
2           1
3           1
4           1
…         …
30686       1
30687       1
30688       1
30689       1
30690       1

[30691 rows x 11 columns]
```

[67]:
```
print(df.columns)
```

```
Index(['Age of the patient', 'Gender of the patient', 'Total Bilirubin',
       'Direct Bilirubin', ' Alkphos Alkaline Phosphotase',
       ' Sgpt Alamine Aminotransferase', 'Sgot Aspartate Aminotransferase',
       'Total Protiens', ' ALB Albumin',
       'A/G Ratio Albumin and Globulin Ratio', 'Result'],
      dtype='object')
```

[68]:
```
df = df.rename({'Age of the patient': 'Age',
                'Gender of the patient': 'Gender'}, axis='columns')
df.columns = df.columns.str.strip()
```

[69]:
```
 # statistics summary
df.describe()
```

[69]:
```
                Age  Total Bilirubin  Direct Bilirubin  \
count  30689.000000     30043.000000      30130.000000
mean      44.107205         3.370319          1.528042
std       15.981043         6.255522          2.869592
min        4.000000         0.400000          0.100000
25%       32.000000         0.800000          0.200000
50%       45.000000         1.000000          0.300000
```

```
75%       55.000000         2.700000          1.300000
max       90.000000        75.000000         19.700000

       Alkphos Alkaline Phosphotase  Sgpt Alamine Aminotransferase  \
count                  29895.000000                   30153.000000
mean                     289.075364                      81.488641
std                      238.537589                     182.158850
min                       63.000000                      10.000000
25%                      175.000000                      23.000000
50%                      209.000000                      35.000000
75%                      298.000000                      62.000000
max                     2110.000000                    2000.000000

       Sgot Aspartate Aminotransferase  Total Protiens   ALB Albumin  \
count                     30229.000000    30228.000000  30197.000000
mean                        111.469979        6.480237      3.130142
std                         280.851078        1.081980      0.792281
min                          10.000000        2.700000      0.900000
25%                          26.000000        5.800000      2.600000
50%                          42.000000        6.600000      3.100000
75%                          88.000000        7.200000      3.800000
max                        4929.000000        9.600000      5.500000

       A/G Ratio Albumin and Globulin Ratio        Result
count                          30132.000000  30691.000000
mean                               0.943467      1.285882
std                                0.323164      0.451841
min                                0.300000      1.000000
25%                                0.700000      1.000000
50%                                0.900000      1.000000
75%                                1.100000      2.000000
max                                2.800000      2.000000
```

[70]:
```
# check null
df.isnull().sum()
```

[70]:
```
Age                                  2
Gender                             902
Total Bilirubin                    648
Direct Bilirubin                   561
Alkphos Alkaline Phosphotase       796
Sgpt Alamine Aminotransferase      538
Sgot Aspartate Aminotransferase    462
Total Protiens                     463
ALB Albumin                        494
A/G Ratio Albumin and Globulin Ratio  559
Result                               0
```

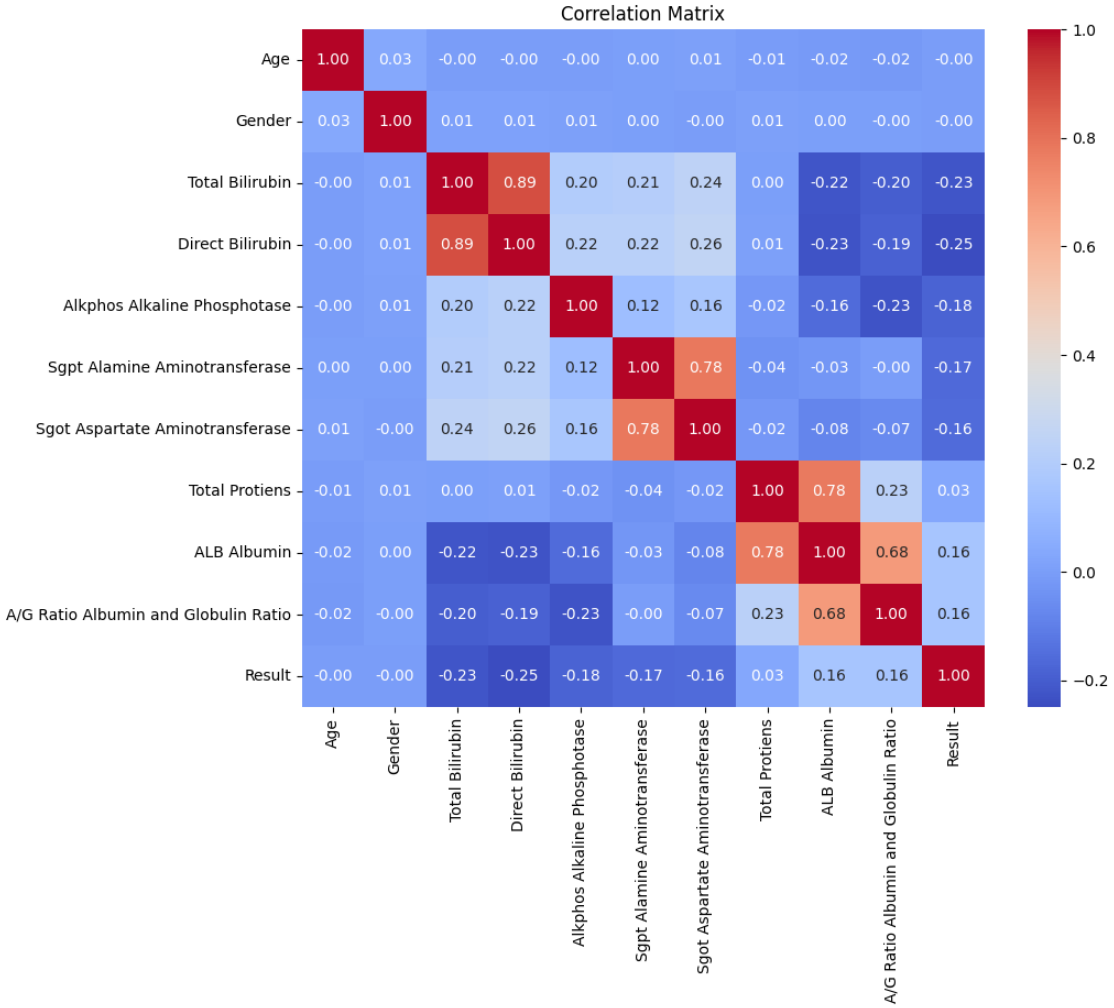
```
dtype: int64
```

[71]:
```
# check data types
print(df.dtypes)

# convert categorical x
df['Gender'] = df['Gender'].map({'Male': 1, 'Female': 0})
```

```
Age                                  float64
Gender                                object
Total Bilirubin                      float64
Direct Bilirubin                     float64
Alkphos Alkaline Phosphotase         float64
Sgpt Alamine Aminotransferase        float64
Sgot Aspartate Aminotransferase      float64
Total Protiens                       float64
ALB Albumin                          float64
A/G Ratio Albumin and Globulin Ratio float64
Result                                 int64
dtype: object
```

2.2 Correlation

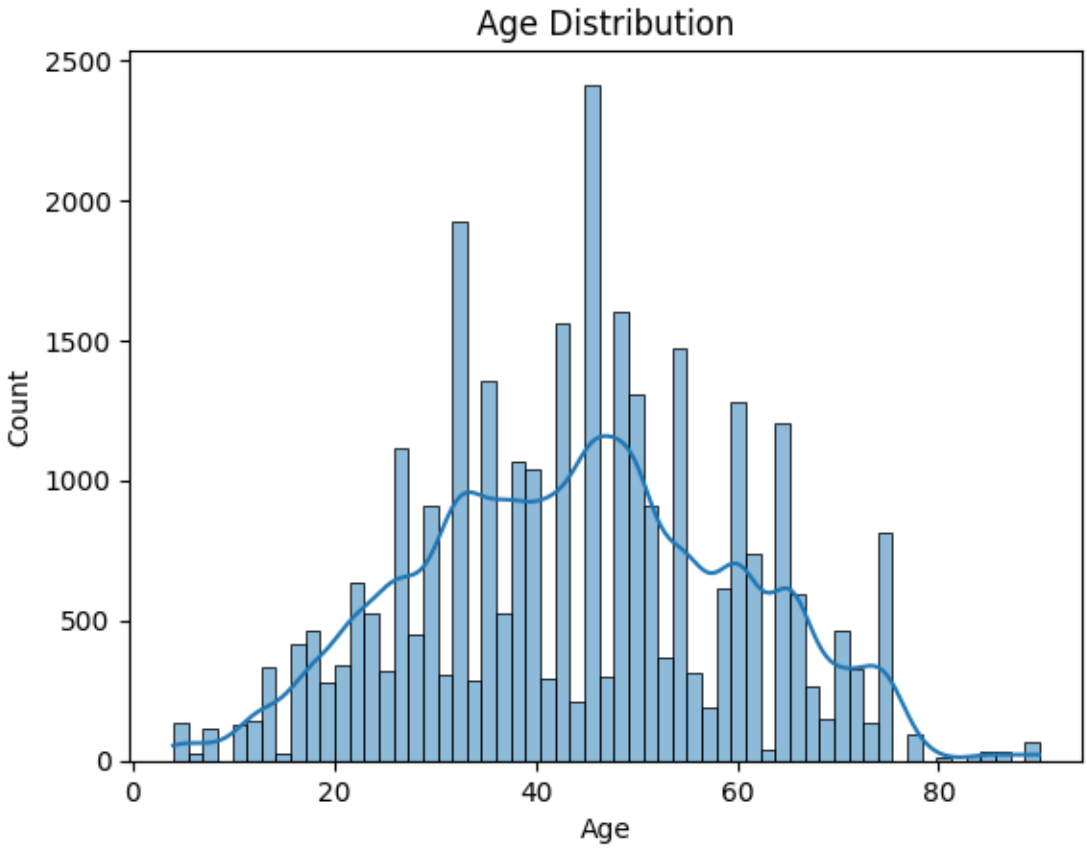
[72]:
```
# correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```

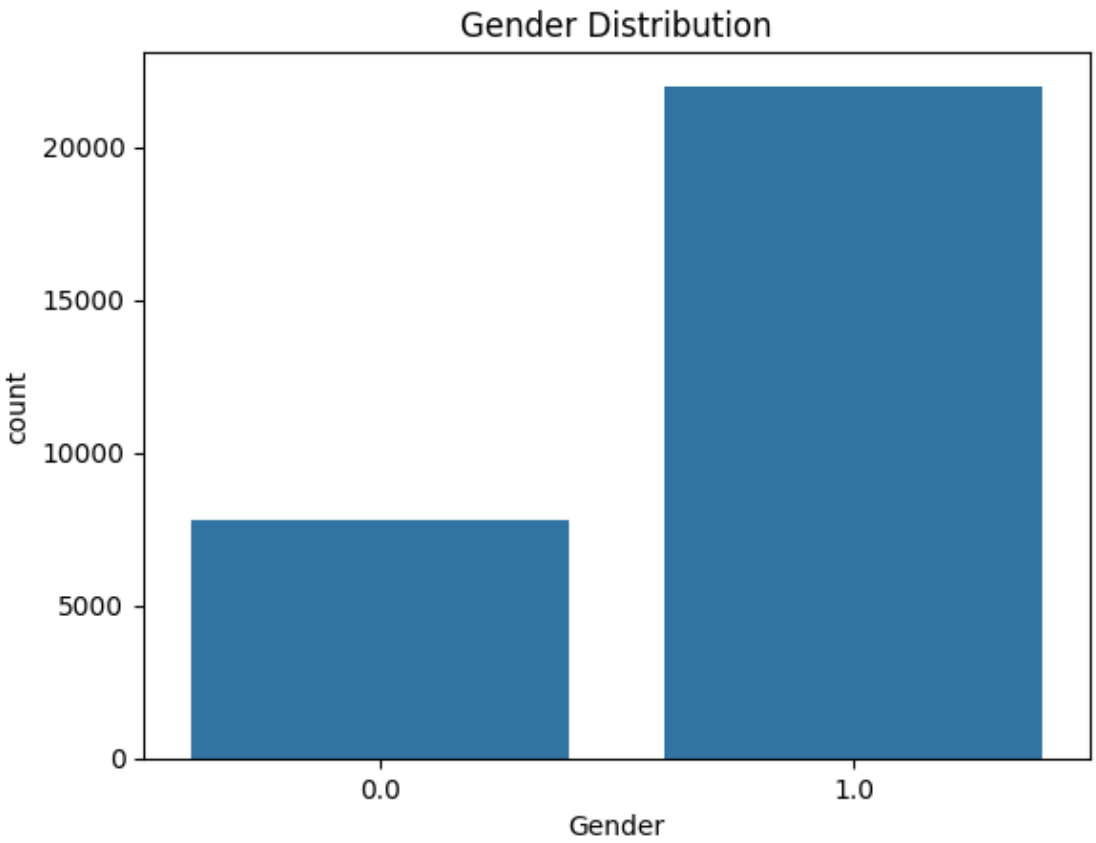


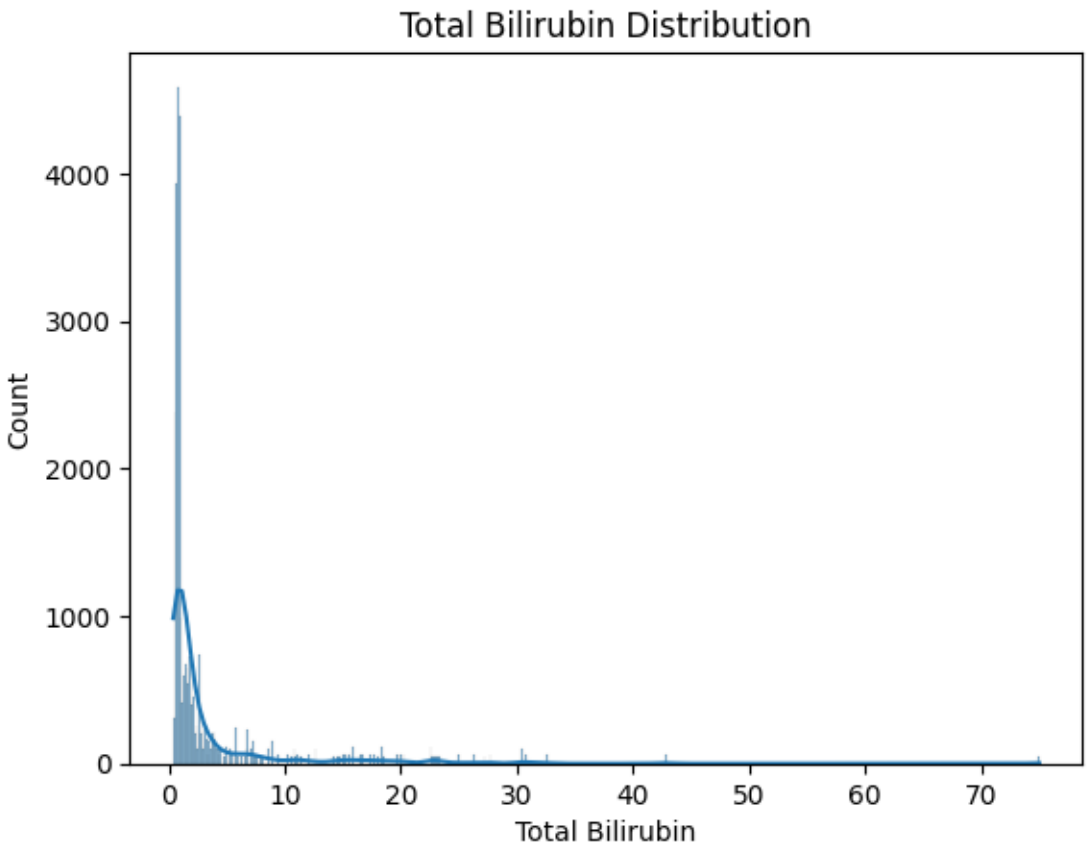
2.3 Visualization

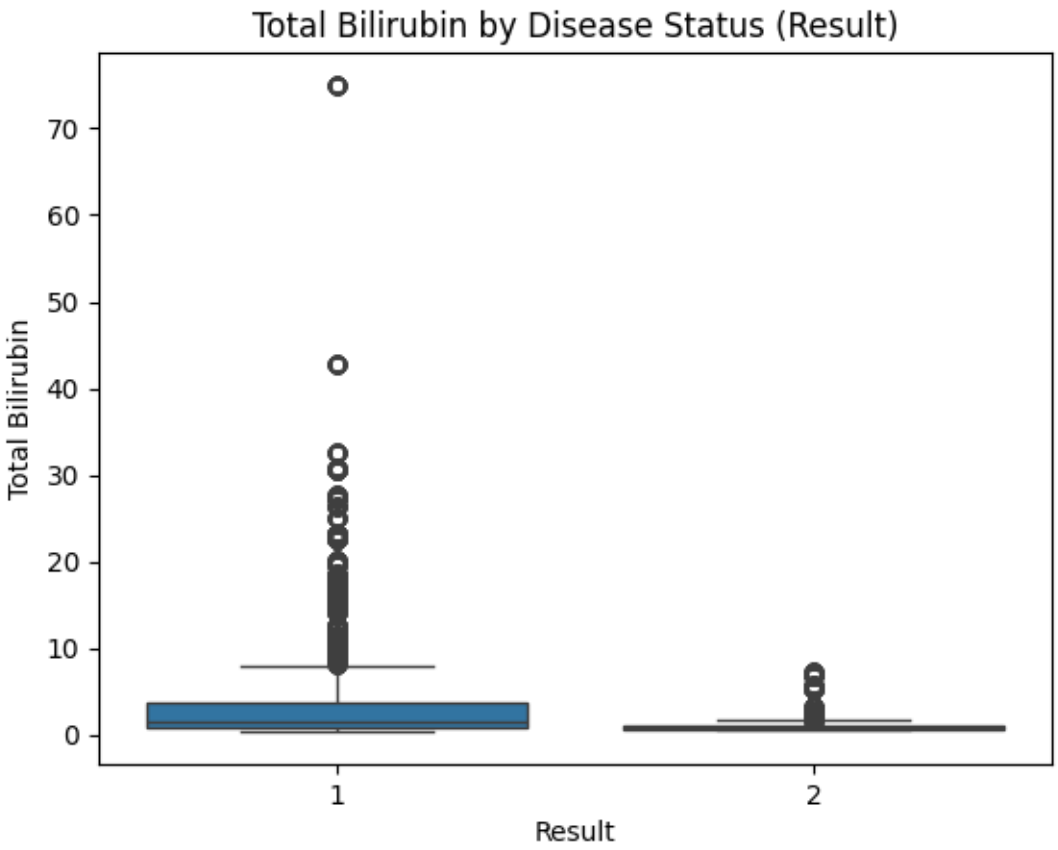
[73]:
```
# visualization
sns.histplot(df['Age'], kde=True)
plt.title('Age Distribution')
plt.show()

sns.countplot(x='Gender', data=df)
plt.title('Gender Distribution')
plt.show()

# total Bilirubin levels
sns.histplot(df['Total Bilirubin'], kde=True)
plt.title('Total Bilirubin Distribution')
plt.show()
sns.boxplot(x='Result', y='Total Bilirubin', data=df)
```

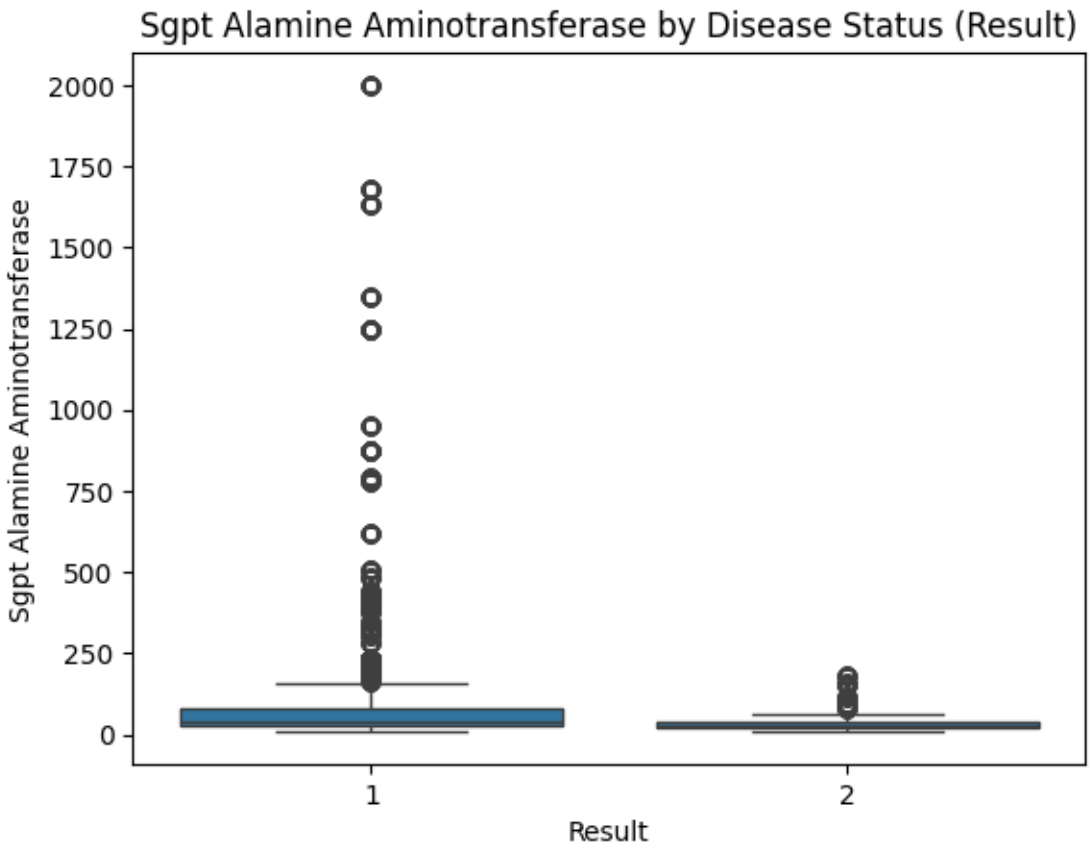
```
plt.title('Total Bilirubin by Disease Status (Result)')
plt.show()

# highly correlated variables
sns.boxplot(x='Result', y='Sgpt Alamine Aminotransferase', data=df)
plt.title('Sgpt Alamine Aminotransferase by Disease Status (Result)')
plt.show()

sns.boxplot(x='Gender', y='Alkphos Alkaline Phosphotase', data=df)
plt.title('Alkaline Phosphotase by Gender')
plt.show()
```

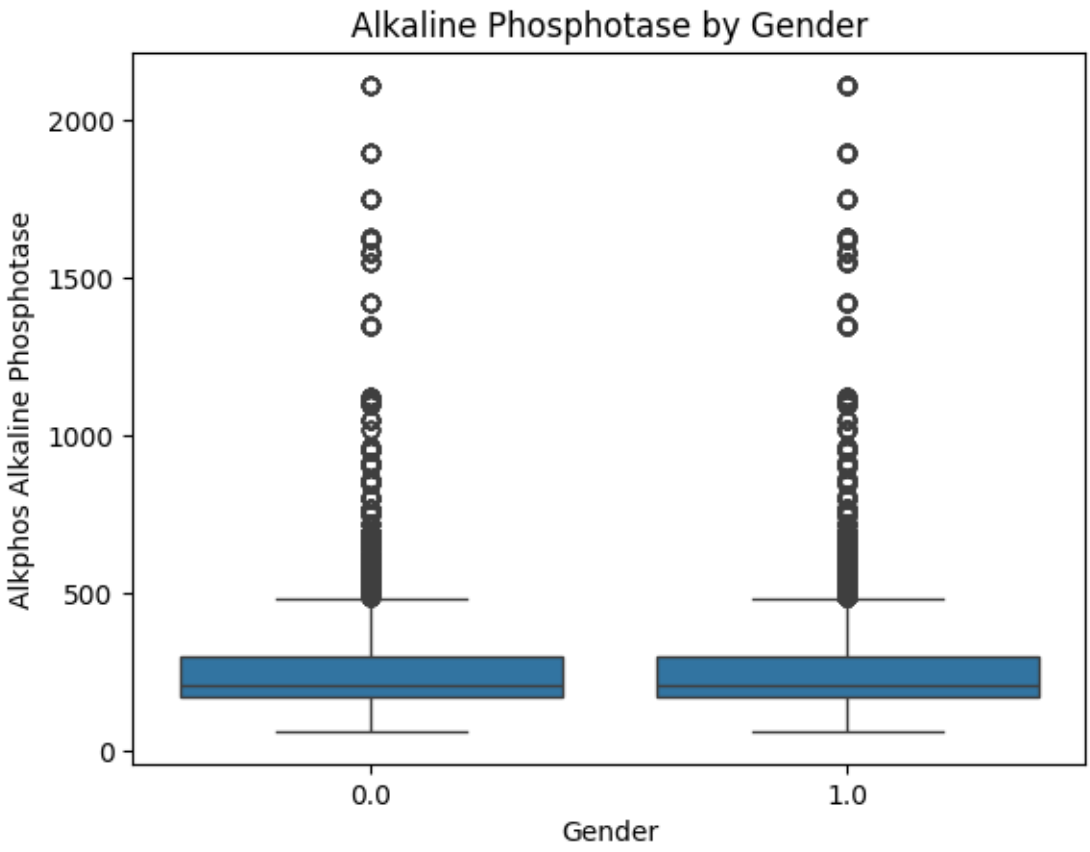



Gender Distribution
20000
15000
10000
5000
0
count
0.0
1.0
Gender

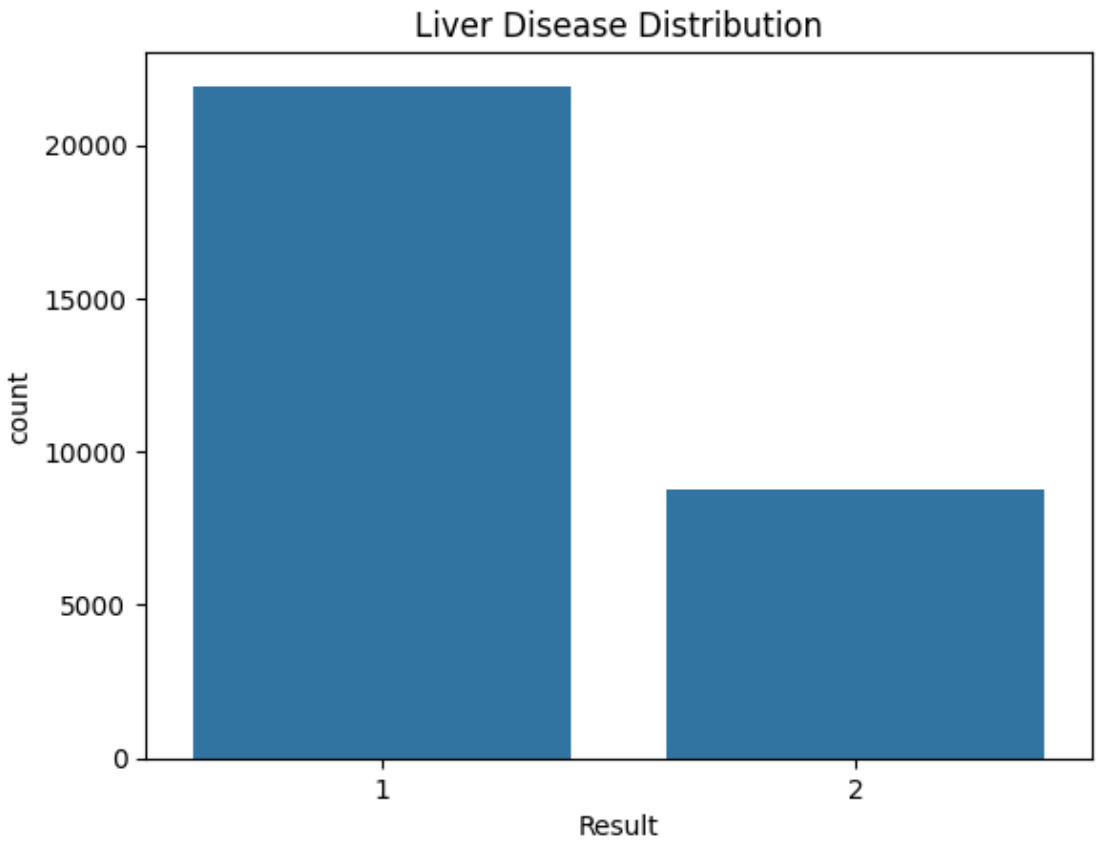

Total Bilirubin Distribution
Count
4000
3000
2000
1000
0
0
10
20
30
40
50
60
70
Total Bilirubin


Total Bilirubin by Disease Status (Result)
Total Bilirubin
70
60
50
40
30
20
10
0
1
2
Result

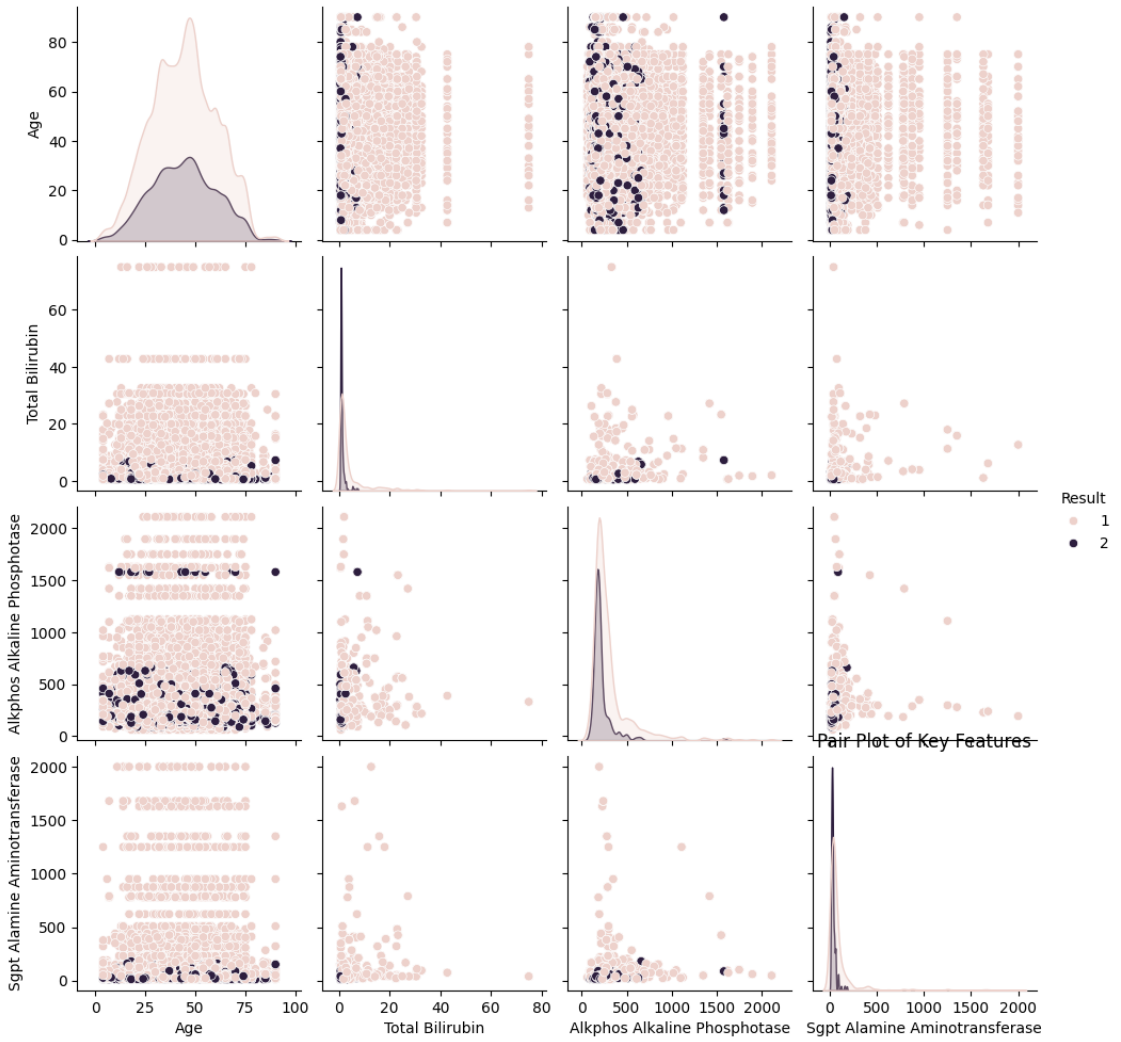

Sgpt Alamine Aminotransferase by Disease Status (Result)
Sgpt Alamine Aminotransferase
2000
1750
1500
1250
1000
750
500
250
0
1
2
Result



[74]:
```
# y
sns.countplot(x='Result', data=df)
plt.title('Liver Disease Distribution')
plt.show()
```



[75]:
```
# key features
sns.pairplot(df[['Age', 'Total Bilirubin', 'Alkphos Alkaline Phosphotase',␣
 ↪'Sgpt Alamine Aminotransferase', 'Result']], hue='Result')
plt.title('Pair Plot of Key Features')
plt.show()
```



2.4 Missing value

[76]:
```
missing_values = df.isnull().sum()
print("Missing Values per Column:\n", missing_values)
```

```
Missing Values per Column:
 Age                              2
Gender                          902
Total Bilirubin                 648
Direct Bilirubin                561
Alkphos Alkaline Phosphotase    796
Sgpt Alamine Aminotransferase   538
Sgot Aspartate Aminotransferase 462
Total Protiens                  463
ALB Albumin                     494
```

```
A/G Ratio Albumin and Globulin Ratio    559
Result                                    0
dtype: int64
```

2.5 Outliers

[77]:
```
# Detect outliers using IQR method
# Convert boolean columns to numeric (int) before calculating quantiles
numeric_df = df.select_dtypes(include=np.number)  # Select only numeric columns

Q1 = numeric_df.quantile(0.05)
Q3 = numeric_df.quantile(0.95)
IQR = Q3 - Q1

# Define lower and upper bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Check for outliers (using numeric columns only)
outliers_iqr = ((numeric_df < lower_bound) | (numeric_df > upper_bound)).sum()
print("\nOutliers detected using IQR method:\n", outliers_iqr)
```

```
Outliers detected using IQR method:
 Age                                    0
Gender                                  0
Total Bilirubin                       105
Direct Bilirubin                        0
Alkphos Alkaline Phosphotase          339
Sgpt Alamine Aminotransferase         633
Sgot Aspartate Aminotransferase       293
Total Protiens                          0
ALB Albumin                             0
A/G Ratio Albumin and Globulin Ratio    0
Result                                  0
dtype: int64
```

3 Data Preparation/Feature Engineering

3.1 Clean outliers

[79]:
```
# Cap outliers at the 5th and 95th percentile for selected features
for col in ['Total Bilirubin', 'Alkphos Alkaline Phosphotase', 'Sgpt Alamine␣
 ↪Aminotransferase',
            'Sgot Aspartate Aminotransferase']:
    lower = df[col].quantile(0.05)
    upper = df[col].quantile(0.95)
    df[col] = df[col].clip(lower, upper)
```

3.2 Clean missing values

[ ]:
```
# Handle missing values
df['Gender'] = df['Gender'].fillna(df['Gender'].mode()[0])

num_cols = ['Total Bilirubin', 'Direct Bilirubin', 'Alkphos Alkaline␣
 ↪Phosphotase',
            'Sgpt Alamine Aminotransferase', 'Sgot Aspartate Aminotransferase',
            'Total Protiens', 'ALB Albumin', 'A/G Ratio Albumin and Globulin␣
 ↪Ratio']

for col in num_cols:
    df[col] = df[col].fillna(df[col].median())
```

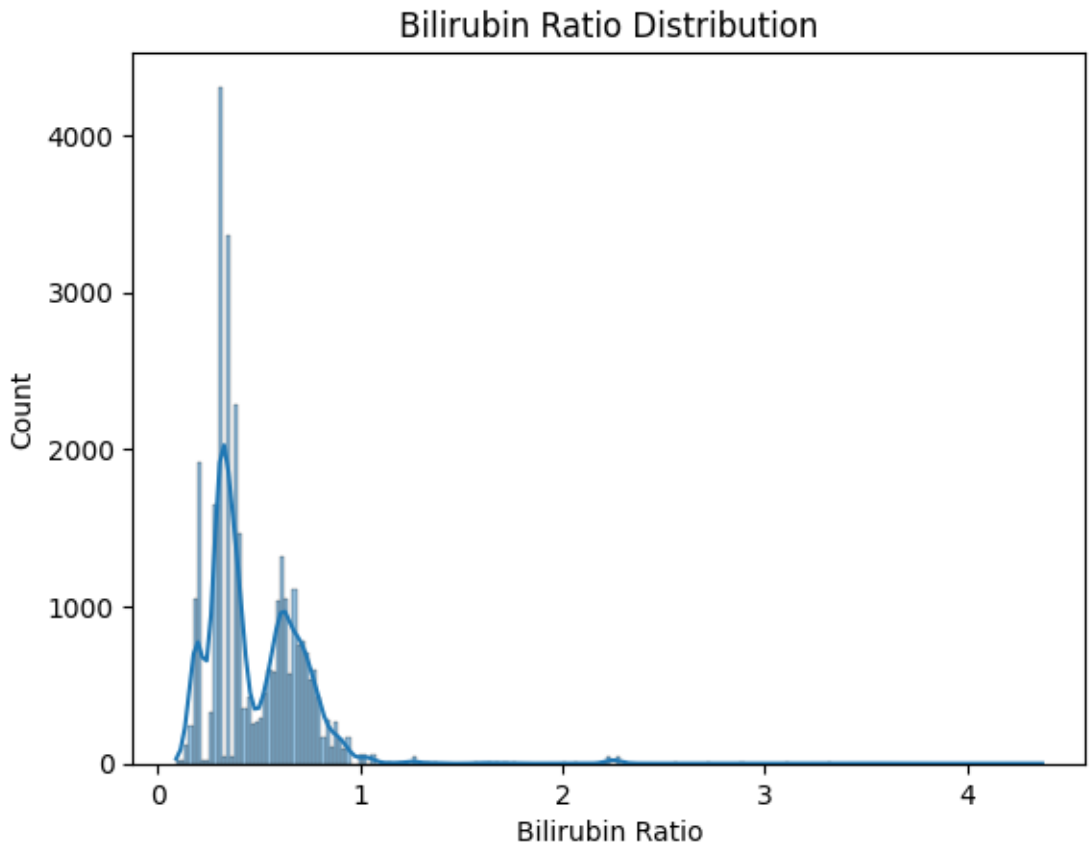
3.3 Log transformation

[ ]:
```
# Log transformation (avoid skewness)
log_transform_cols = ['Total Bilirubin', 'Direct Bilirubin', 'Sgpt Alamine␣
 ↪Aminotransferase', 'Sgot Aspartate Aminotransferase']
for col in log_transform_cols:
    df[col] = np.log1p(df[col])  # np.log1p(x) = log(1 + x) to avoid log(0)␣
 ↪error
```

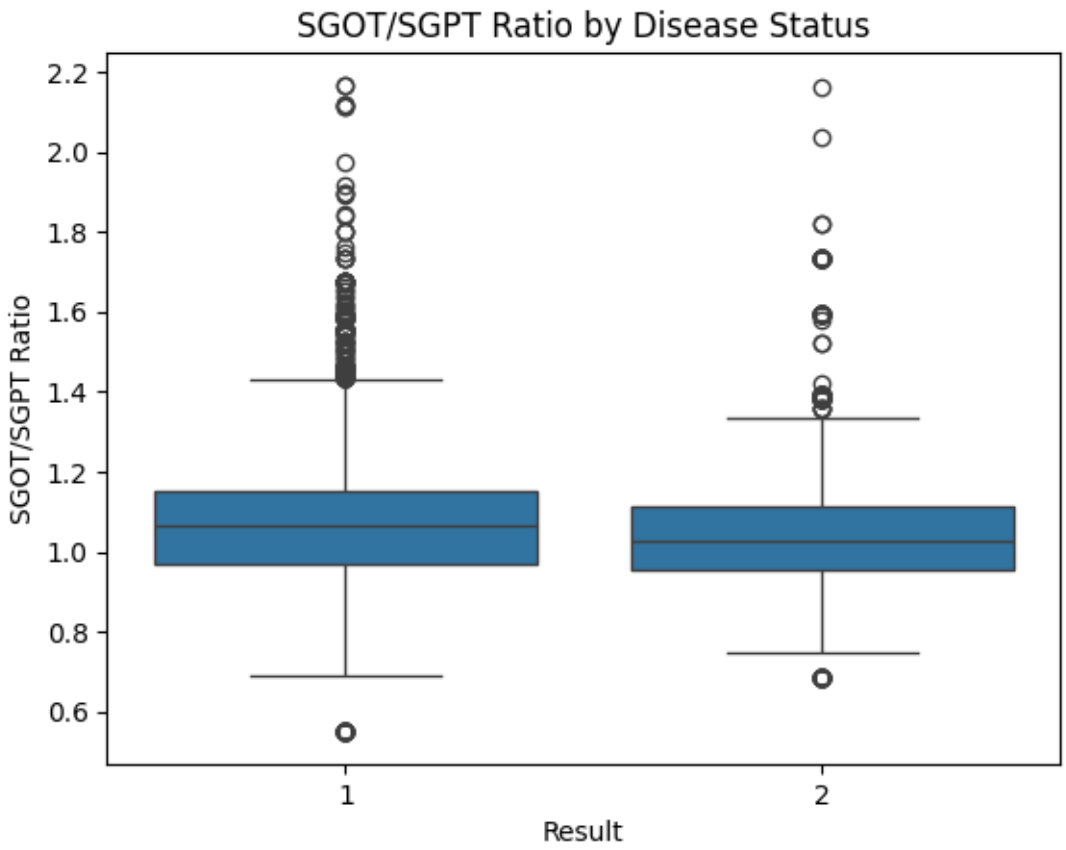
3.4 Feature Engineering

[80]:
```
# Create new features
df['Bilirubin Ratio'] = df['Direct Bilirubin'] / df['Total Bilirubin']
df['SGOT/SGPT Ratio'] = df['Sgot Aspartate Aminotransferase'] / df['Sgpt␣
 ↪Alamine Aminotransferase']
df['Protien Ratio'] = df['ALB Albumin'] / df['Total Protiens']

# Group by age
bins = [0, 18, 40, 60, 100]
labels = ['Child', 'Young', 'Middle-aged', 'Old']
df['Age Group'] = pd.cut(df['Age'], bins=bins, labels=labels)

# One-hot encode categorical variable
df = pd.get_dummies(df, columns=['Age Group'], drop_first=True)

# Remove highly correlated features to avoid multicollinearity
df = df.drop(columns=['Direct Bilirubin'])  # Highly correlated with Total␣
 ↪Bilirubin
```

[81]:
```
sns.histplot(df['Bilirubin Ratio'], kde=True)
plt.title('Bilirubin Ratio Distribution')
plt.show()
```



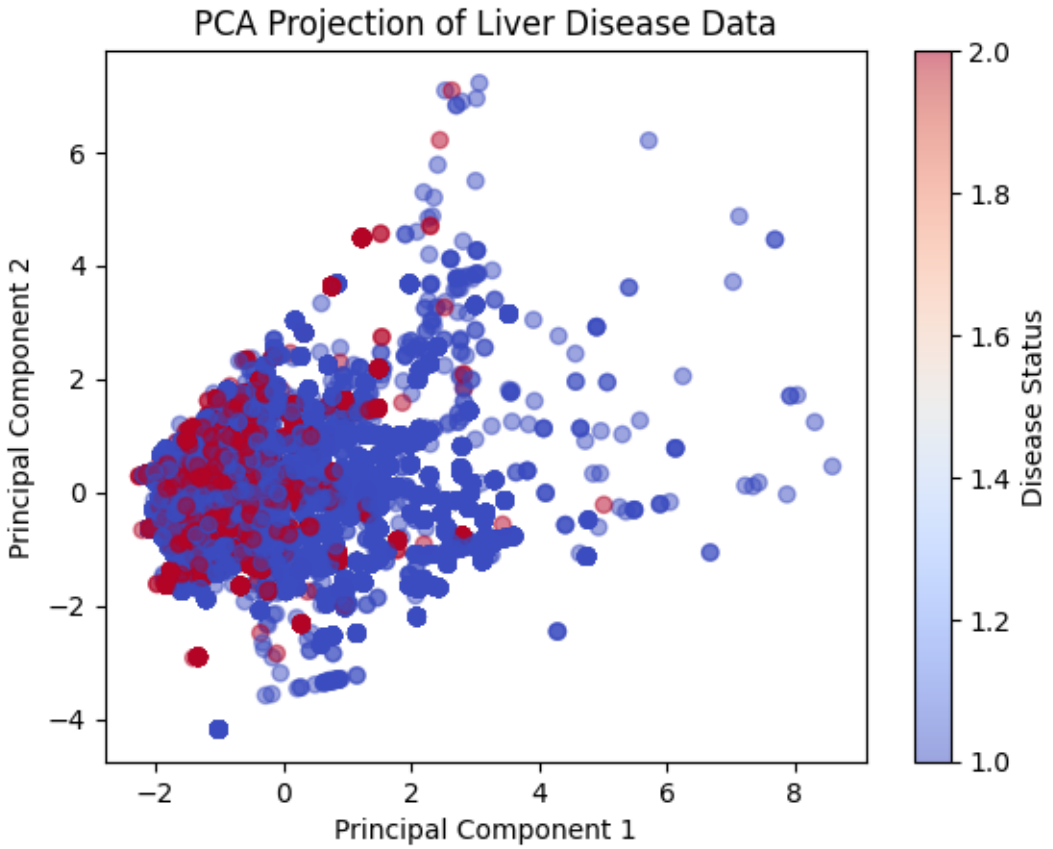
[82]:
```
sns.boxplot(x='Result', y='SGOT/SGPT Ratio', data=df)
plt.title('SGOT/SGPT Ratio by Disease Status')
plt.show()
```



3.5 PCA

[83]:
```
# choose features for PCA
features = ['SGOT/SGPT Ratio', 'Bilirubin Ratio', 'Sgpt Alamine␣
 ↪Aminotransferase', 'Sgot Aspartate Aminotransferase']
X = df[features]

# standard
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# visualization for PCA
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df['Result'], cmap='coolwarm', alpha=0.
 ↪5)
plt.xlabel('Principal Component 1')
```

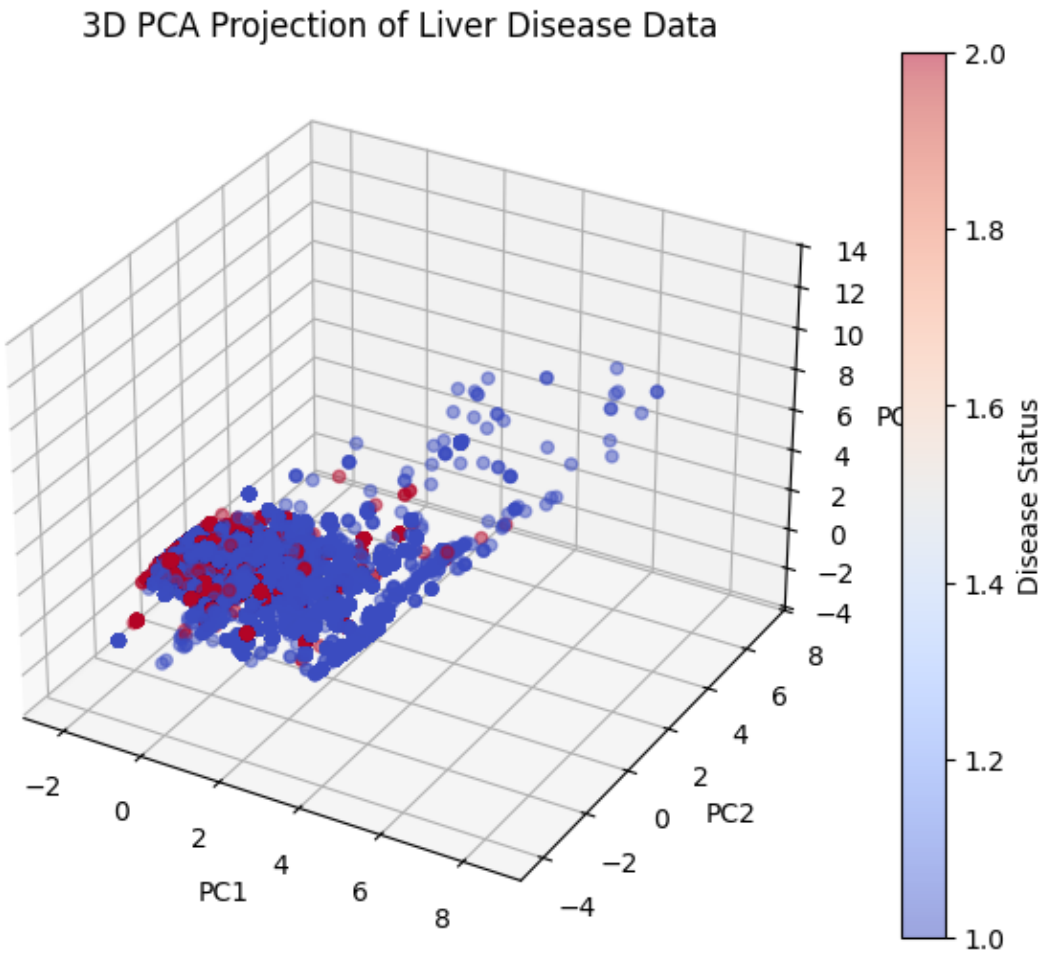
```
plt.ylabel('Principal Component 2')
plt.title('PCA Projection of Liver Disease Data')
plt.colorbar(label='Disease Status')
plt.show()

# variance ratio of PCA principal components
print(f"Explained variance ratio: {pca.explained_variance_ratio_}")
```




```
Explained variance ratio: [0.53403442 0.28552017]
```

[84]:
```
pca = PCA(n_components=3)
X_pca = pca.fit_transform(X_scaled)

fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(X_pca[:,0], X_pca[:,1], X_pca[:,2], c=df['Result'],␣
 ↪cmap='coolwarm', alpha=0.5)
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.set_zlabel('PC3')
```

```
plt.title('3D PCA Projection of Liver Disease Data')
plt.colorbar(scatter, label='Disease Status')
plt.show()
```



3.5.1 Scaled

[163]:
```
scaler = StandardScaler()
df_scaled = df.copy()
df_scaled[df.columns] = scaler.fit_transform(df)
```

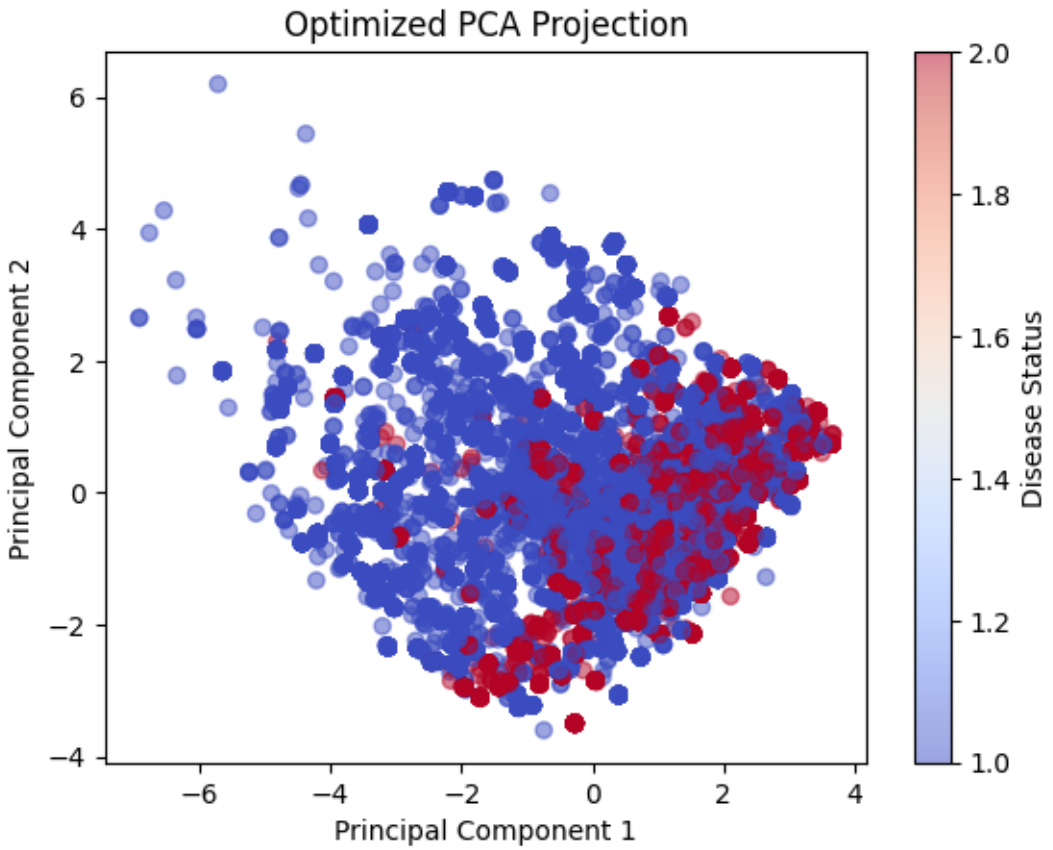
3.5.2 Under random forest

[164]:
```
X = df.drop(columns=['Result'])
y = df['Result']
X = X.fillna(X.median())
model = RandomForestClassifier(n_estimators=100, random_state=42)
selector = RFE(model, n_features_to_select=10)  # 10 features
```

```
X_selected = selector.fit_transform(X, y)

selected_features = X.columns[selector.support_]
print("Selected Features:", selected_features)
```

```
Selected Features: Index(['Total Bilirubin', 'Alkphos Alkaline Phosphotase',
       'Sgpt Alamine Aminotransferase', 'Sgot Aspartate Aminotransferase',
       'Total Protiens', 'ALB Albumin', 'A/G Ratio Albumin and Globulin Ratio',
       'Bilirubin Ratio', 'SGOT/SGPT Ratio', 'Protien Ratio'],
      dtype='object')
```

[165]:
```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(df_scaled[selected_features])

plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='coolwarm', alpha=0.5)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Optimized PCA Projection')
plt.colorbar(label='Disease Status')
plt.show()
```



4 Splitting

[205]:
```
# Prepare data for modeling
X = df[['Total Bilirubin', 'Alkphos Alkaline Phosphotase',
        'Sgpt Alamine Aminotransferase', 'Sgot Aspartate Aminotransferase',
        'Total Protiens', 'ALB Albumin', 'A/G Ratio Albumin and Globulin Ratio',
        'Bilirubin Ratio', 'SGOT/SGPT Ratio', 'Protien Ratio']]
df['Result'] = df['Result'].replace({1: 1, 2: 0})
y = df['Result']

# Split data into training (80%) and testing (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

5 Model

5.1 Rondom Forest

[206]:
```
# Training Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
```

[206]:
```
RandomForestClassifier(random_state=42)
```

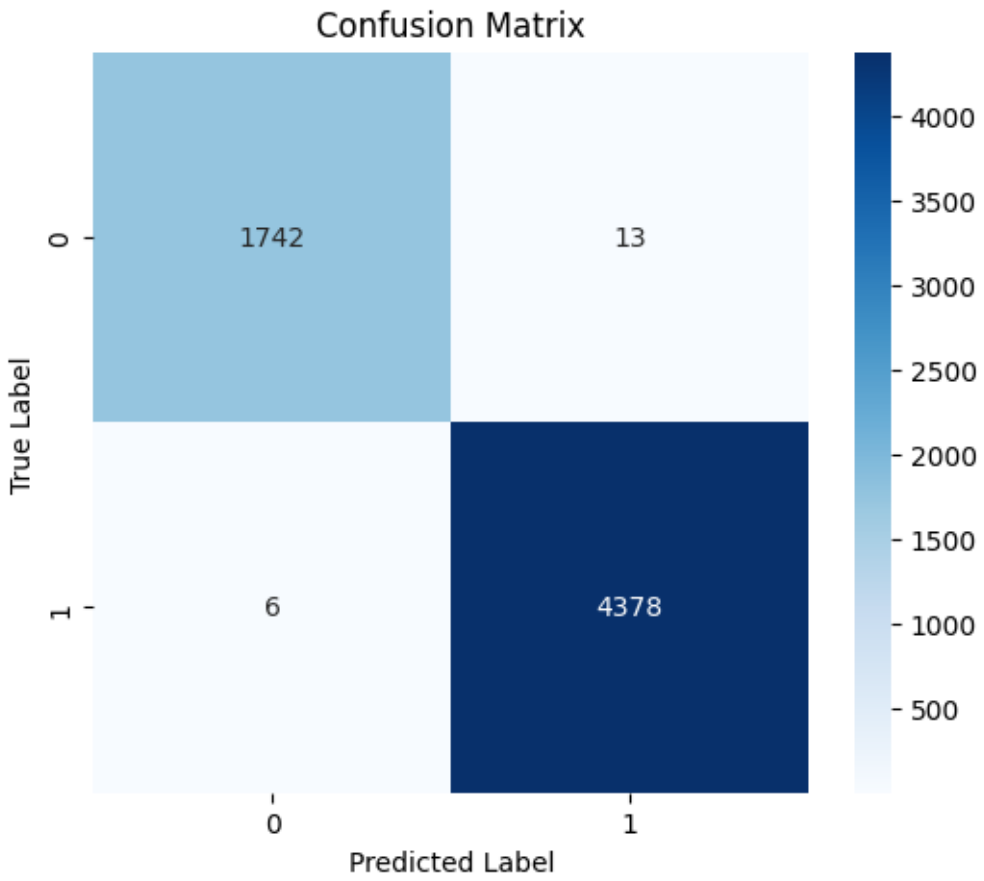
[207]:
```
train_accuracy = rf_model.score(X_train, y_train)
print(f"Training Accuracy: {train_accuracy:.4f}")
```

```
Training Accuracy: 0.9999
```

[208]:
```
y_pred = rf_model.predict(X_test)

# Evaluation
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Print Results
print("Test Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```

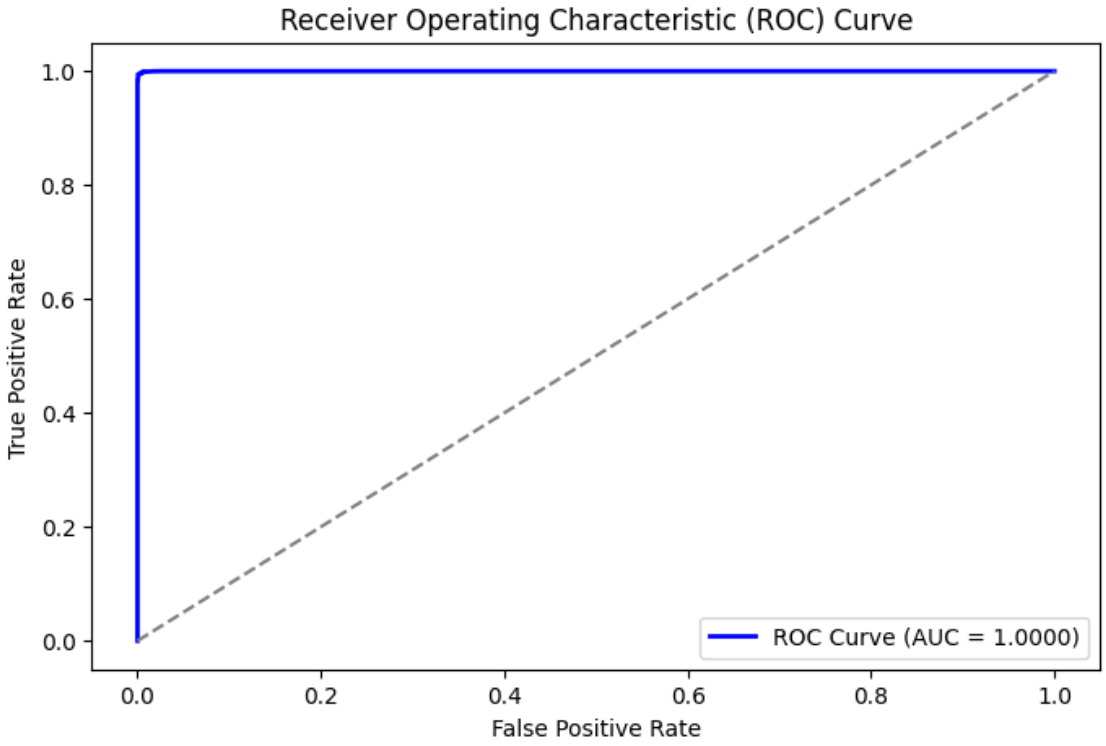
```
Test Accuracy: 0.9969050333930608
Confusion Matrix:
 [[1742   13]
 [   6 4378]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      1755
           1       1.00      1.00      1.00      4384

    accuracy                           1.00      6139
   macro avg       1.00      1.00      1.00      6139
weighted avg       1.00      1.00      1.00      6139
```

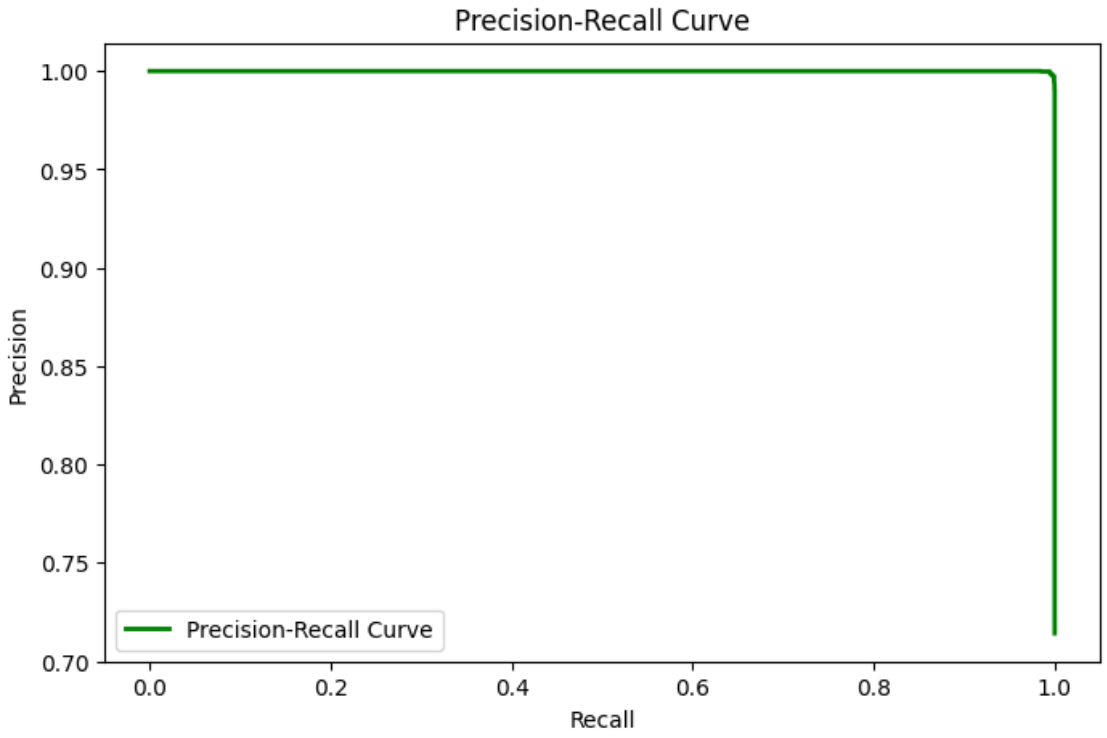
[209]:
```
# Plot Confusion Matrix
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",␣
 ↪xticklabels=set(y_test), yticklabels=set(y_test))
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```



[210]:
```
# Predictions
y_pred = rf_model.predict(X_test)
y_probs = rf_model.predict_proba(X_test)[:, 1]  # Get probability for positive␣
↪class

# Compute ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_probs)
roc_auc = auc(fpr, tpr)

# Compute Precision-Recall Curve
precision, recall, _ = precision_recall_curve(y_test, y_probs)

# Plot ROC Curve
plt.figure(figsize=(8, 5))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC Curve (AUC = {roc_auc:.4f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristic (ROC) Curve")
```

```
plt.legend()
plt.show()

# Plot Precision-Recall Curve
plt.figure(figsize=(8, 5))
plt.plot(recall, precision, color='green', lw=2, label="Precision-Recall Curve")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve")
plt.legend()
plt.show()
```





5.2 Cross validation

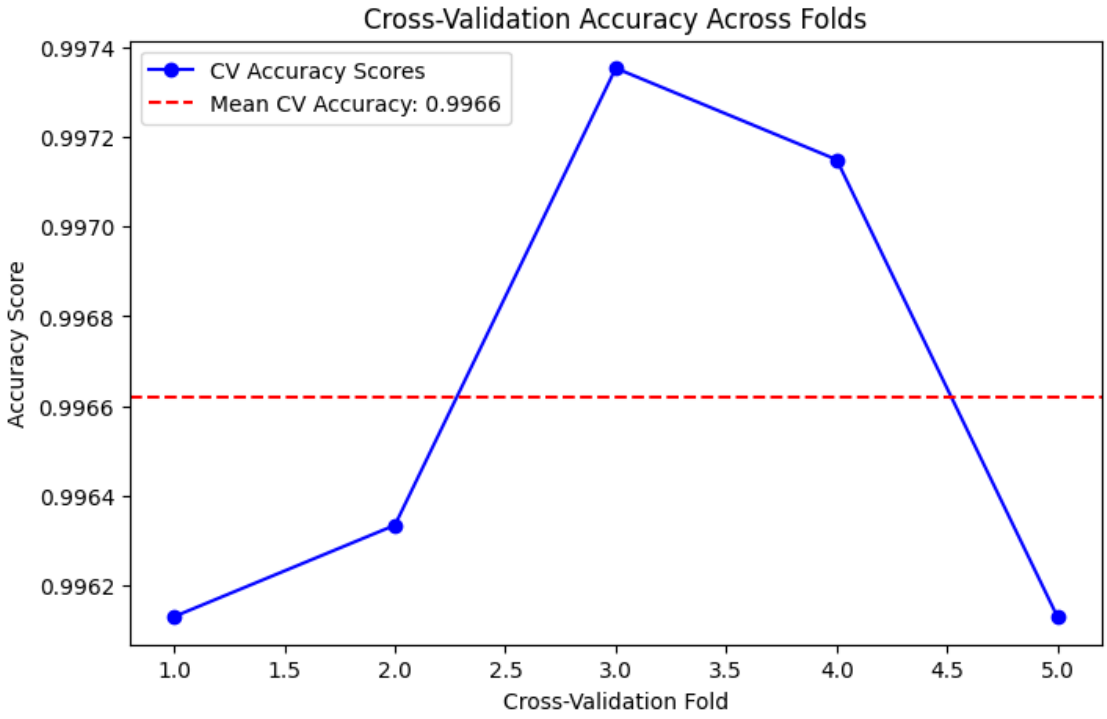
[211]:
```
# Perform Cross-Validation (5-fold)
cv_scores = cross_val_score(rf_model, X_train, y_train, cv=5,␣
 ↪scoring="accuracy")
```

[212]:
```
print(cv_scores)
print(f"Cross-Validation Accuracy: {cv_scores.mean():.4f} ± {cv_scores.std():.
 ↪4f}")
```

```
[0.99613113 0.99633476 0.99735234 0.99714868 0.99613035]
Cross-Validation Accuracy: 0.9966 ± 0.0005
```

[213]:
```
# Plot Cross-Validation Accuracy as a Line Chart
plt.figure(figsize=(8, 5))
plt.plot(range(1, len(cv_scores) + 1), cv_scores, marker='o', linestyle='-',␣
 ↪color='blue', label="CV Accuracy Scores")
plt.axhline(np.mean(cv_scores), linestyle="--", color="red", label=f"Mean CV␣
 ↪Accuracy: {np.mean(cv_scores):.4f}")
plt.xlabel("Cross-Validation Fold")
plt.ylabel("Accuracy Score")
plt.title("Cross-Validation Accuracy Across Folds")
plt.legend()
```

```
plt.show()
```



5.3 Under PCA

[214]:
```
# Splitting dataset
X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(X_pca, y,␣
 ↪test_size=0.2, random_state=42)

# Training Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_pca, y_train_pca)

# Predictions
y_pred_pca = rf_model.predict(X_test_pca)

# Evaluation
accuracy_pca = accuracy_score(y_test_pca, y_pred_pca)
conf_matrix_pca = confusion_matrix(y_test_pca, y_pred_pca)
class_report_pca = classification_report(y_test_pca, y_pred_pca)

# Print Results
print("Accuracy:", accuracy_pca)
print("Confusion Matrix:\n", conf_matrix_pca)
```

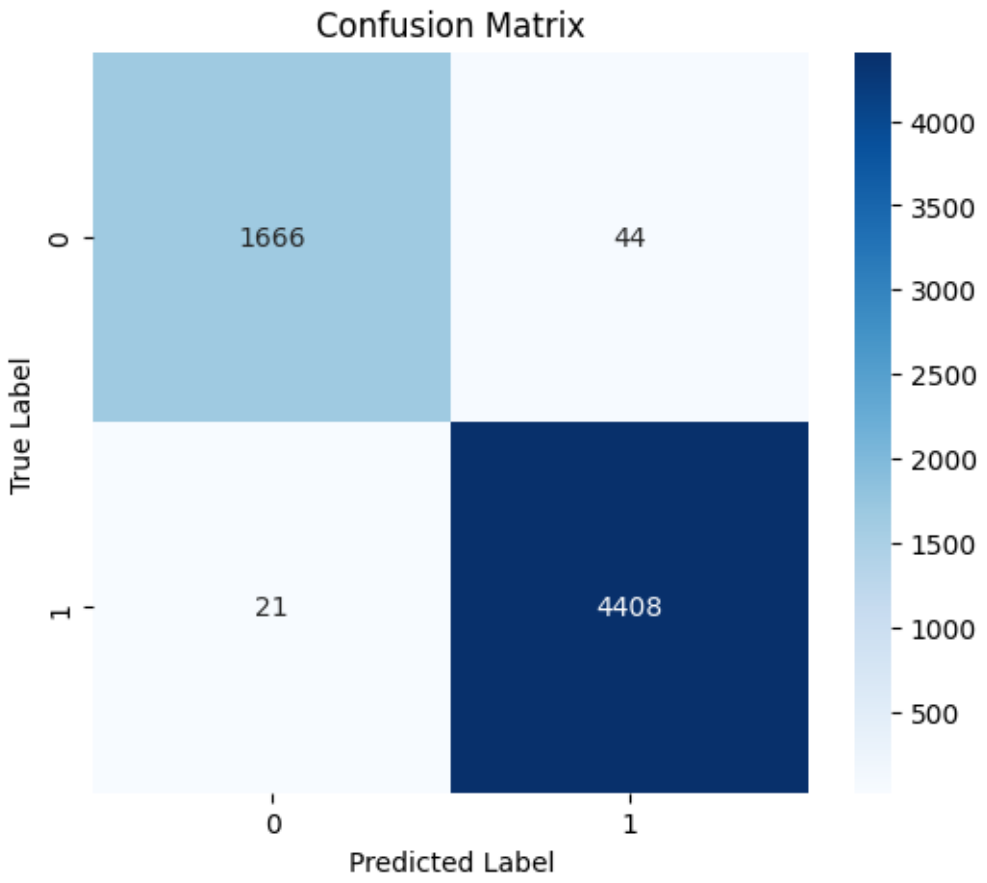
```
print("Classification Report:\n", class_report_pca)
```

```
Accuracy: 0.9894119563446816
Confusion Matrix:
 [[1666   44]
 [  21 4408]]
Classification Report:
               precision    recall  f1-score   support

           0       0.99      0.97      0.98      1710
           1       0.99      1.00      0.99      4429

    accuracy                           0.99      6139
   macro avg       0.99      0.98      0.99      6139
weighted avg       0.99      0.99      0.99      6139
```

[215]:
```
train_accuracy_pca = rf_model.score(X_train_pca, y_train_pca)
print(f"Training Accuracy: {train_accuracy_pca:.4f}")
```

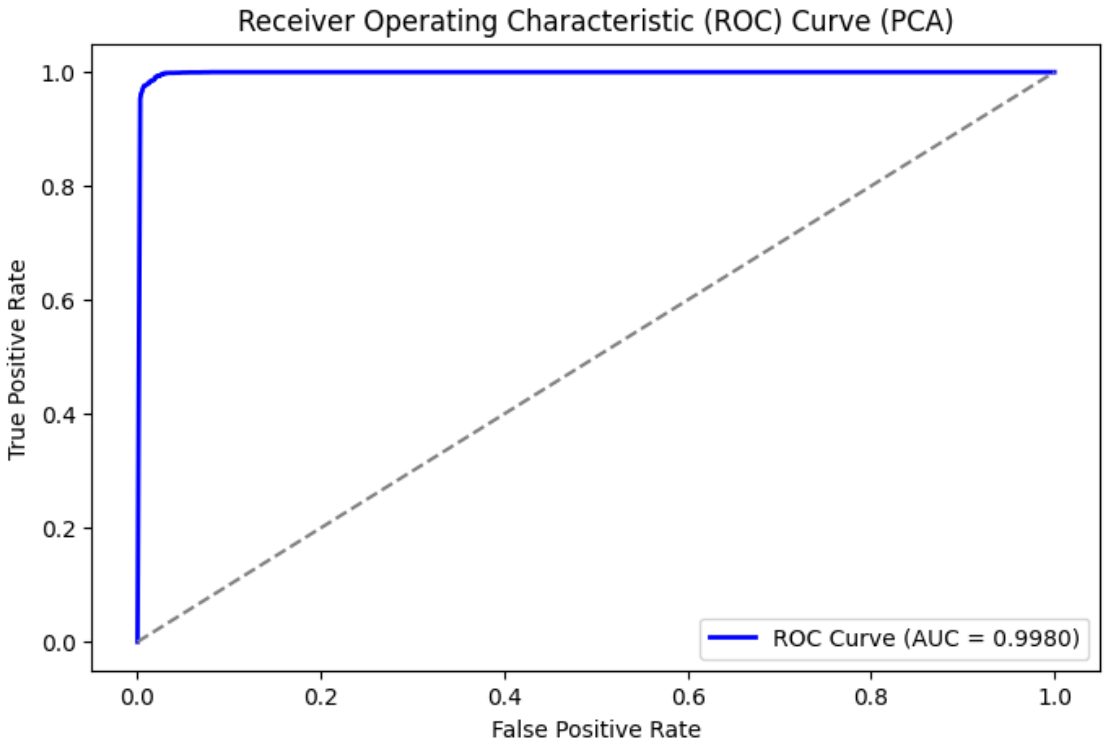
```
Training Accuracy: 0.9999
```

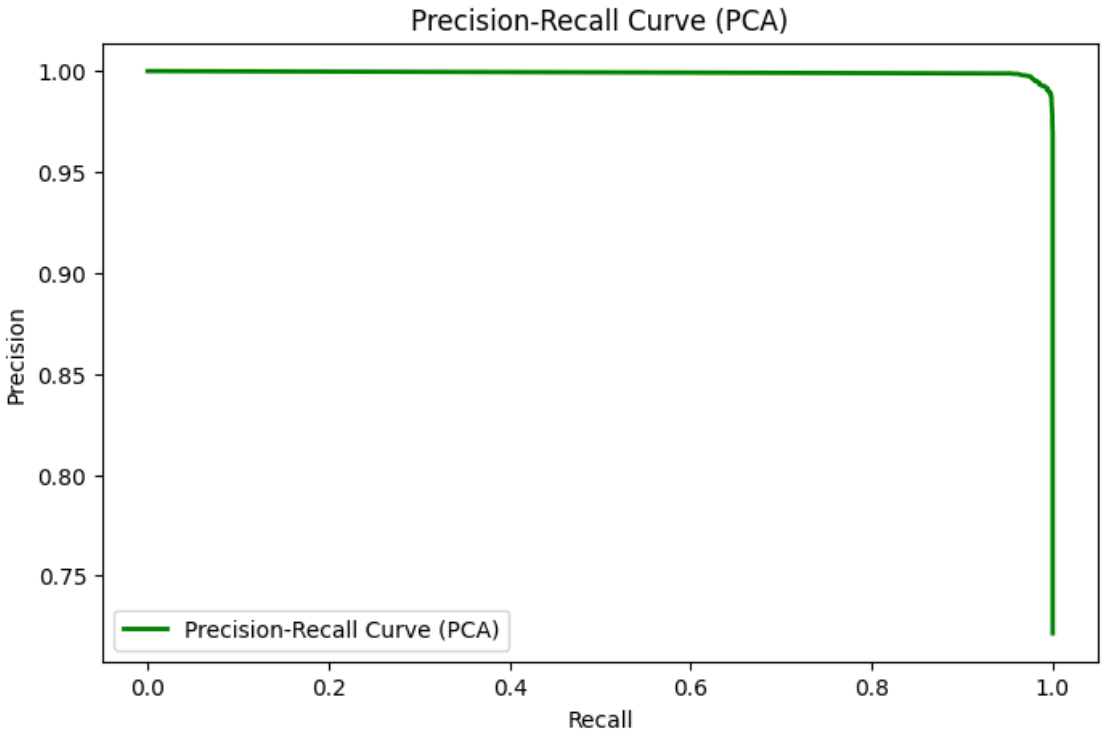
[216]:
```
# Plot Confusion Matrix
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix_pca, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```



[217]:
```
# Predictions
y_pred_pca = rf_model.predict(X_test_pca)
y_probs_pca = rf_model.predict_proba(X_test_pca)[:, 1]  # Get probability for␣
 ↪positive class

# Compute ROC Curve
fpr_pca, tpr_pca, _ = roc_curve(y_test_pca, y_probs_pca)
roc_auc_pca = auc(fpr_pca, tpr_pca)

# Compute Precision-Recall Curve
precision_pca, recall_pca, _ = precision_recall_curve(y_test_pca, y_probs_pca)

# Plot ROC Curve
plt.figure(figsize=(8, 5))
plt.plot(fpr_pca, tpr_pca, color='blue', lw=2, label=f'ROC Curve (AUC =␣
 ↪{roc_auc_pca:.4f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

```
plt.title("Receiver Operating Characteristic (ROC) Curve (PCA)")
plt.legend()
plt.show()

# Plot Precision-Recall Curve
plt.figure(figsize=(8, 5))
plt.plot(recall_pca, precision_pca, color='green', lw=2,␣
 ↪label="Precision-Recall Curve (PCA)")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve (PCA)")
plt.legend()
plt.show()
```




Precision-Recall Curve (PCA)
1.00
0.95
0.90
0.85
0.80
0.75
Precision
Precision-Recall Curve (PCA)
0.0
0.2
0.4
0.6
0.8
1.0
Recall