Importing libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Mount google drive to fetch the dataset

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Read dataset

```
df = pd.read_csv('/content/drive/MyDrive/ProjectStage/Liver/Dataset/All_Combined.csv')
```

```
df.sample(5)
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Am
348	45.0	Male	2.400000	1.100000	168	33	
1994	53.0	m	4.664552	3.056918	744.1800508	70.07613446	
944	NaN	NaN	3.210000	0.860000	278.44	61.55	

979	NaN	NaN	0.320000	0.140000	91.18	26.76
1648	27.0	m	3.366542	2.044608	469.9292357	260.6680803

Features in dataset

```
df.shape
```

```
(2391, 11)
```

```
df.columns
```

```
Index(['Age', 'Gender', 'Total_Bilirubin', 'Direct_Bilirubin',
       'Alkaline_Phosphotase', 'Alamine_Aminotransferase',
       'Aspartate_Aminotransferase', 'Total_Protiens', 'Albumin',
       'Albumin_and_Globulin_Ratio', 'Dataset'],
      dtype='object')
```

Finding null values

```
# function to convert n and N to null values
def convert_to_nan(df):

  columns = df.columns
  for col in columns:
    df[col] = df[col].replace({'n': np.nan, 'N': np.nan})
  return df
```

```
df = convert_to_nan(df)
```

```
df.isna().sum()
```

	0
Age	500
Gender	500
Total_Bilirubin	0
Direct_Bilirubin	0
Alkaline_Phosphotase	22
Alamine_Aminotransferase	5
Aspartate_Aminotransferase	16
Total_Protiens	0
Albumin	0
Albumin_and_Globulin_Ratio	4
Dataset	0

dtype: int64

Removing null values

```
df.drop('Gender', axis=1, inplace=True)
df.drop('Age', axis=1, inplace=True)
```

```
df.isna().sum()
```

	0
Total_Bilirubin	0
Direct_Bilirubin	0
Alkaline_Phosphotase	22
Alamine_Aminotransferase	5
Aspartate_Aminotransferase	16
Total_Protiens	0
Albumin	0
Albumin_and_Globulin_Ratio	4
Dataset	0

dtype: int64

```
# Convert 'Alkaline_Phosphotase' column to numeric, coercing errors to NaN
df['Alkaline_Phosphotase'] = pd.to_numeric(df['Alkaline_Phosphotase'], errors='coerce')
df['Alamine_Aminotransferase'] = pd.to_numeric(df['Alamine_Aminotransferase'], errors='coerce')
df['Aspartate_Aminotransferase'] = pd.to_numeric(df['Aspartate_Aminotransferase'], errors='coerce')
df['Albumin_and_Globulin_Ratio'] = pd.to_numeric(df['Albumin_and_Globulin_Ratio'], errors='coerce')
df['Total_Protiens'] = pd.to_numeric(df['Total_Protiens'], errors='coerce')
df['Albumin'] = pd.to_numeric(df['Albumin'], errors='coerce')



# Now fill NaN values with the mean

df['Alkaline_Phosphotase'] = df['Alkaline_Phosphotase'].fillna(np.mean(df['Alkaline_Phosphotase']))
df['Alamine_Aminotransferase'] = df['Alamine_Aminotransferase'].fillna(np.mean(df['Alamine_Aminotransferase']))
df['Aspartate_Aminotransferase'] = df['Aspartate_Aminotransferase'].fillna(np.mean(df['Aspartate_Aminotransferase']))
df['Albumin_and_Globulin_Ratio'] = df['Albumin_and_Globulin_Ratio'].fillna(np.mean(df['Albumin_and_Globulin_Ratio']))
```

```
df['Total_Protiens'] = df['Total_Protiens'].fillna(np.mean(df['Total_Protiens']))
df['Albumin'] = df['Albumin'].fillna(np.mean(df['Albumin']))
```

```
# df.dropna(inplace=True)
```

```
df.dtypes
```

	0
Total_Bilirubin	float64
Direct_Bilirubin	float64
Alkaline_Phosphotase	float64
Alamine_Aminotransferase	float64
Aspartate_Aminotransferase	float64
Total_Protiens	float64
Albumin	float64
Albumin_and_Globulin_Ratio	float64
Dataset	int64

dtype: object

```
df.isna().sum()
```

	0
Total_Bilirubin	0
Direct_Bilirubin	0
Alkaline_Phosphotase	0

Alamine_Aminotransferase	0
Aspartate_Aminotransferase	0
Total_Protiens	0
Albumin	0
Albumin_and_Globulin_Ratio	0
Dataset	0

dtype: int64

```
df.shape
```

```
(2391, 9)
```

Dividing independent and dependent data in X and Y respectively

```
df.columns
```

```
Index(['Total_Bilirubin', 'Direct_Bilirubin', 'Alkaline_Phosphotase',
       'Alamine_Aminotransferase', 'Aspartate_Aminotransferase',
       'Total_Protiens', 'Albumin', 'Albumin_and_Globulin_Ratio', 'Dataset'],
      dtype='object')
```

```
X = df.iloc[:, :-1]
X.sample(5)
```

	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferas
1774	7.973344	3.036862	546.319440	181.562733	348.85248
342	2.600000	1.200000	410.000000	59.000000	57.00000

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
352	1.000000	0.300000	208.000000	17.000000	15.00000
1510	2.830050	2.859360	592.660559	216.284220	351.17047
1541	2.655904	1.388051	660.737060	136.504003	359.54097

```
X.columns
```

```
Index(['Total_Bilirubin', 'Direct_Bilirubin', 'Alkaline_Phosphotase',
       'Alamine_Aminotransferase', 'Aspartate_Aminotransferase',
       'Total_Protiens', 'Albumin', 'Albumin_and_Globulin_Ratio'],
      dtype='object')
```

```
y = df.iloc[:, -1]
y.sample(6)
```

	Dataset
37	0
2170	1
107	0
572	2
480	1
2147	1

dtype: int64

Unique values in target variable (dependent variable)

```
y.unique()
```

```
array([0, 1, 2])
```

```
y.isna().sum()
```

```
np.int64(0)
```

```
y = y.fillna(0)
```

```
y = y.astype(int)
```

```
y.dtype
```

```
dtype('int64')
```

```
y[:5]
```

	Dataset
0	0
1	0
2	0
3	0
4	0

dtype: int64

```
y=y.map({1:1, 2:0, 0:0})
```

```
y
```

	Dataset
0	0
1	0
2	0
3	0
4	0
...	...
2386	1
2387	1
2388	1
2389	1
2390	1

2391 rows × 1 columns

dtype: int64

```
y.value_counts()
```

	count
Dataset	
1	1668
0	723

dtype: int64

```
dtype: int64
```

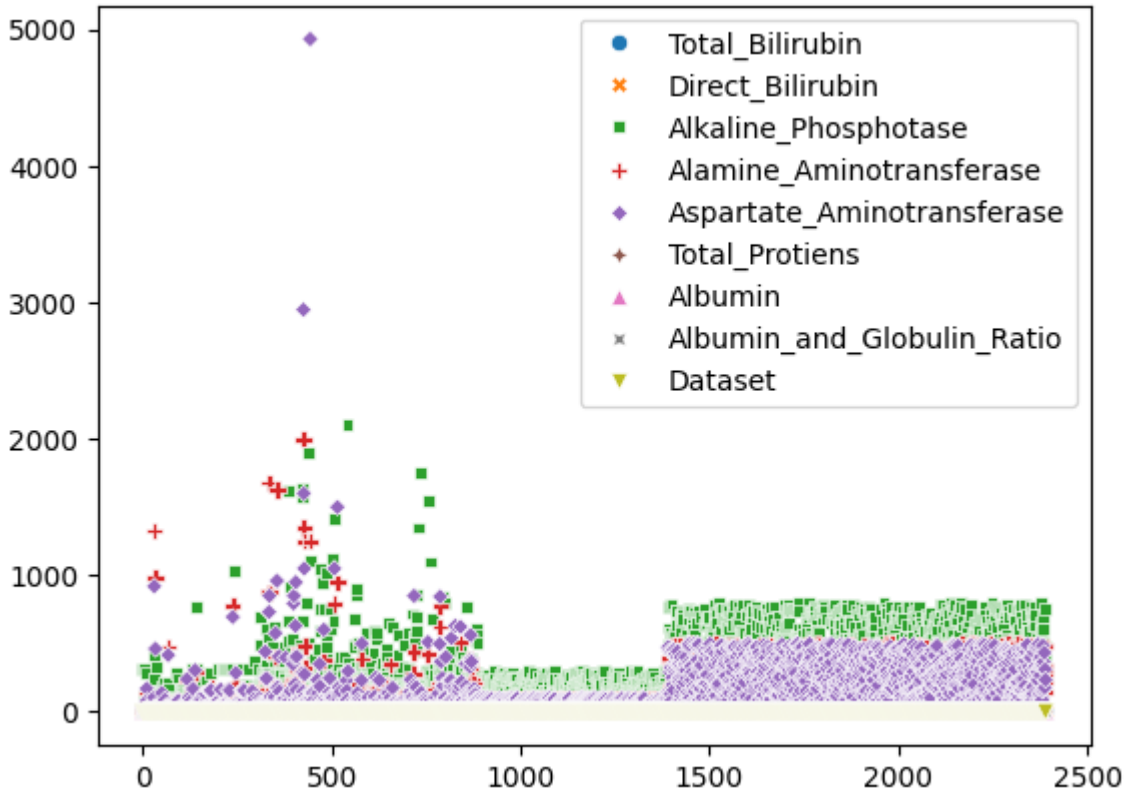
```
y.value_counts()[0], y.value_counts()[1]
```

```
(np.int64(723), np.int64(1668))
```

Finding outlier using scatter plot

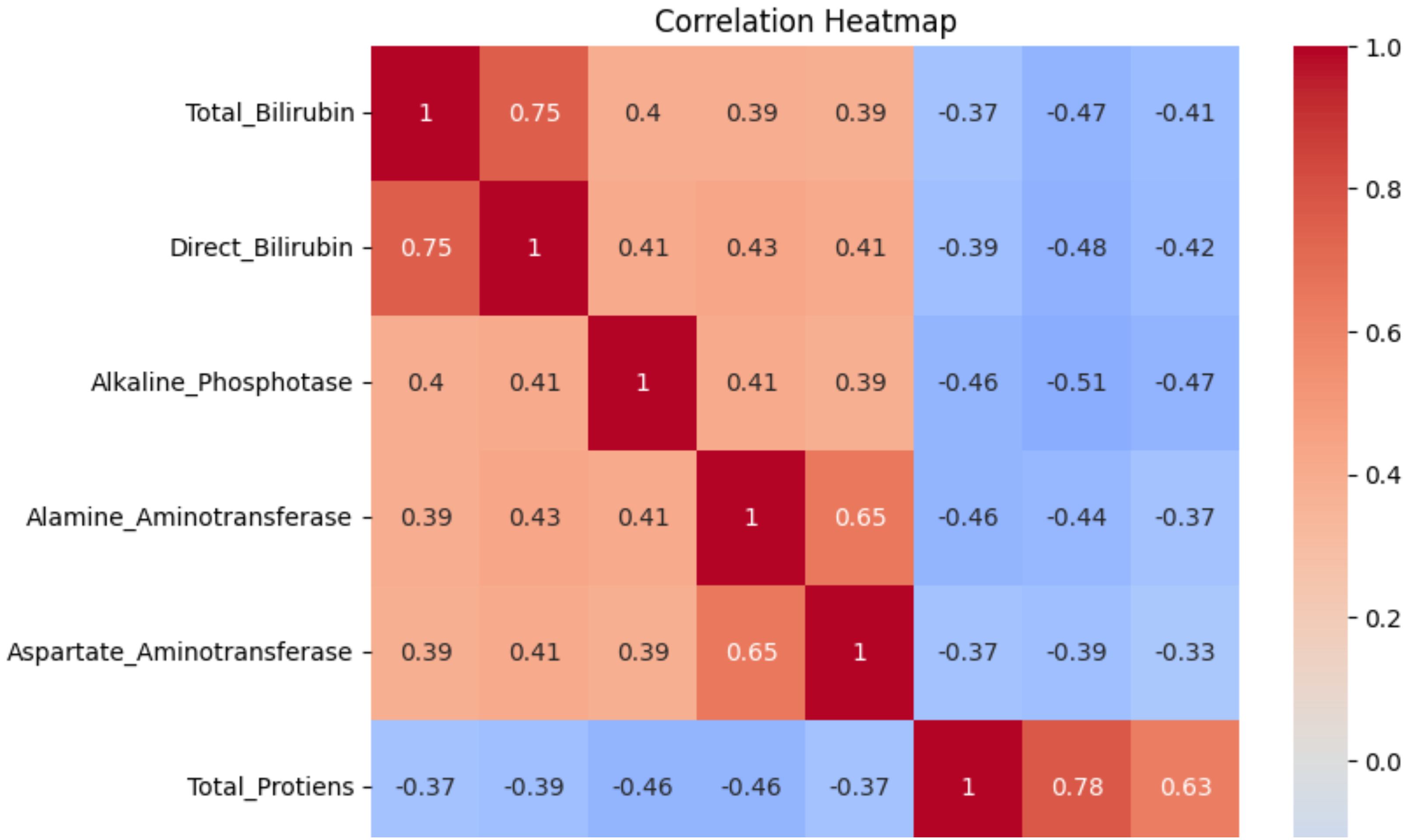
```
sns.scatterplot(data=df)
```

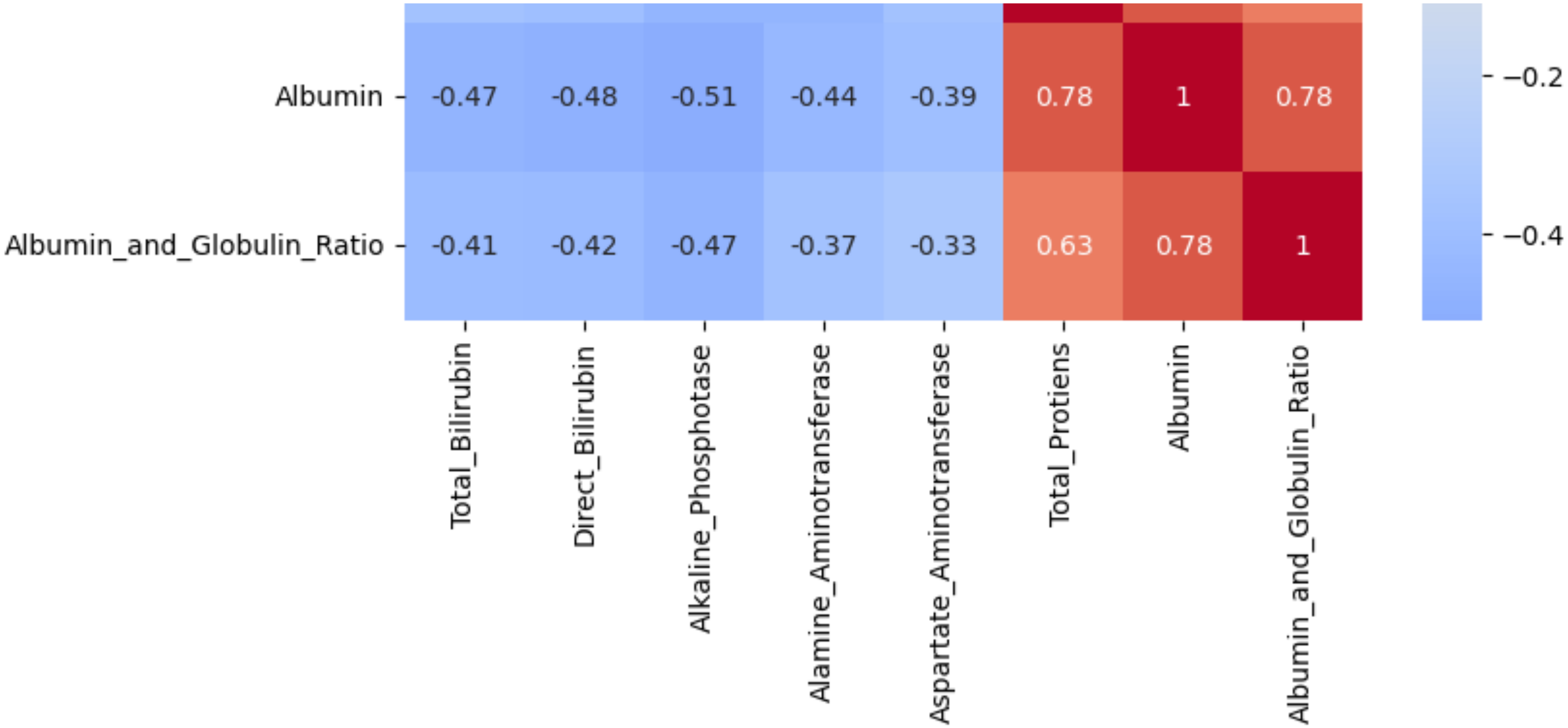
```
<Axes: >
```



Finding outlier using heatmap

```
corr_matrix = X.corr()

plt.figure(figsize=(8, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', center=0)
plt.title('Correlation Heatmap')
plt.show()
```



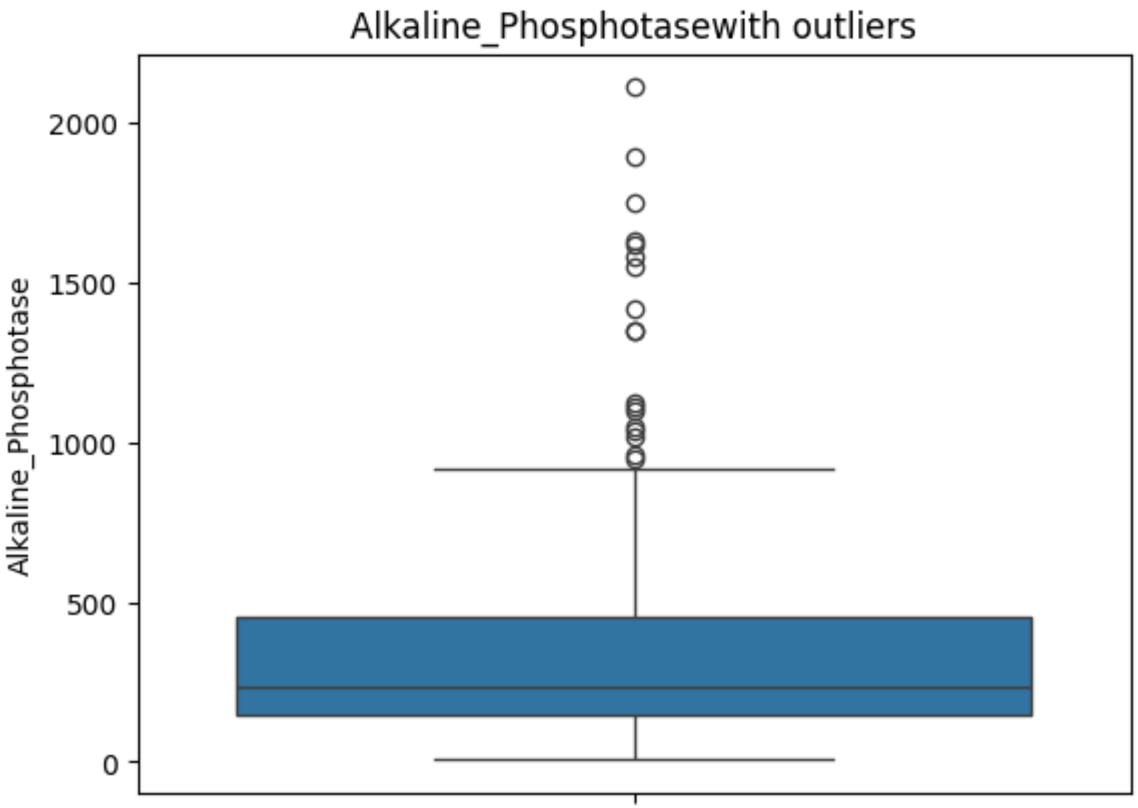


Double-click (or enter) to edit

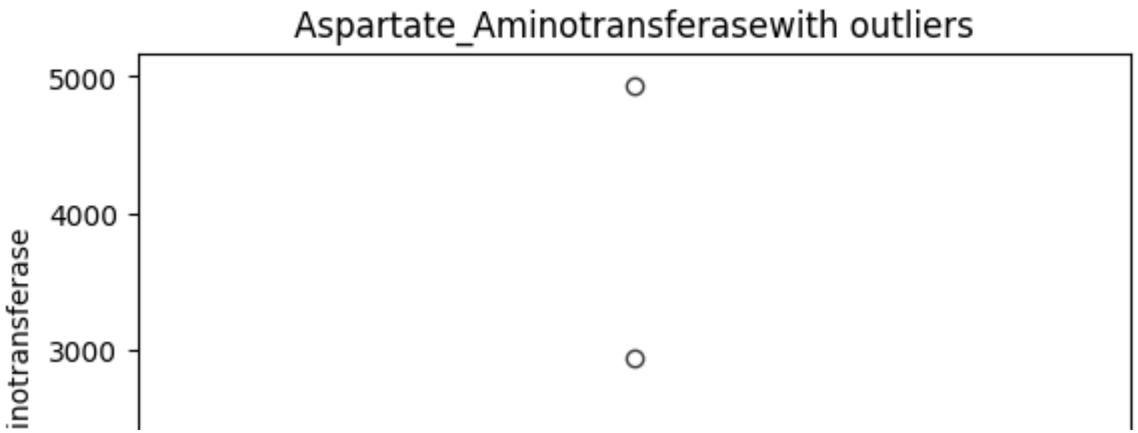
```
# function for box plot
def printBox(df, col, title):
  sns.boxplot(df[col])
  plt.title(col + title)
```

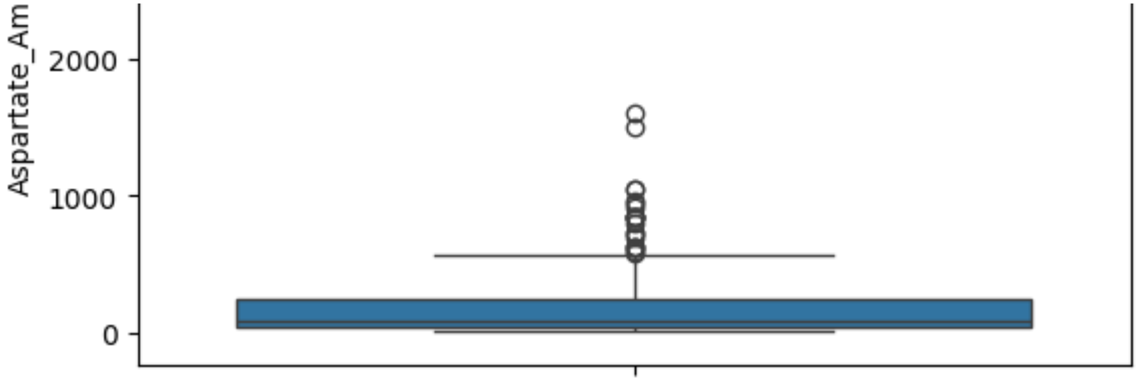
Finding outlier using box plot

```
printBox(df, 'Alkaline_Phosphotase', 'with outliers')
```




```
printBox(df, 'Aspartate_Aminotransferase', 'with outliers')
```





Removing outlier using IQR

```
# Function to remove outliers
def removeOutlier(df, col):
  Q1 = df[col].quantile(0.25)
  Q3 = df[col].quantile(0.75)
  IQR = Q3 - Q1

  lower = Q1 - 1.5 * IQR
  upper = Q3 + 1.5 * IQR

  filtered_col = df[col][(df[col] >= lower) & (df[col] <= upper)]
  # filtered_col = (df[col] >= lower) & (df[col] <= upper)

  return filtered_col
```

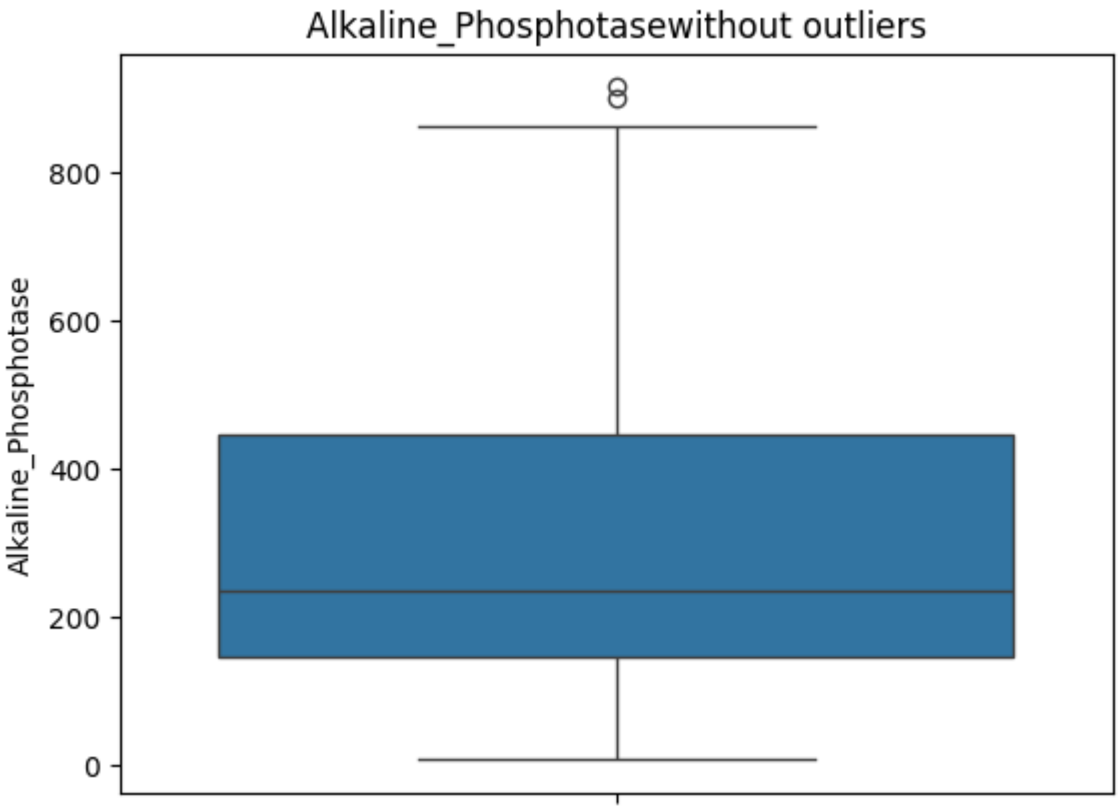
```
df['Alkaline_Phosphotase'] = removeOutlier(df, 'Alkaline_Phosphotase')
```

```
df['Aspartate_Aminotransferase'] = removeOutlier(df, 'Aspartate_Aminotransferase')
```

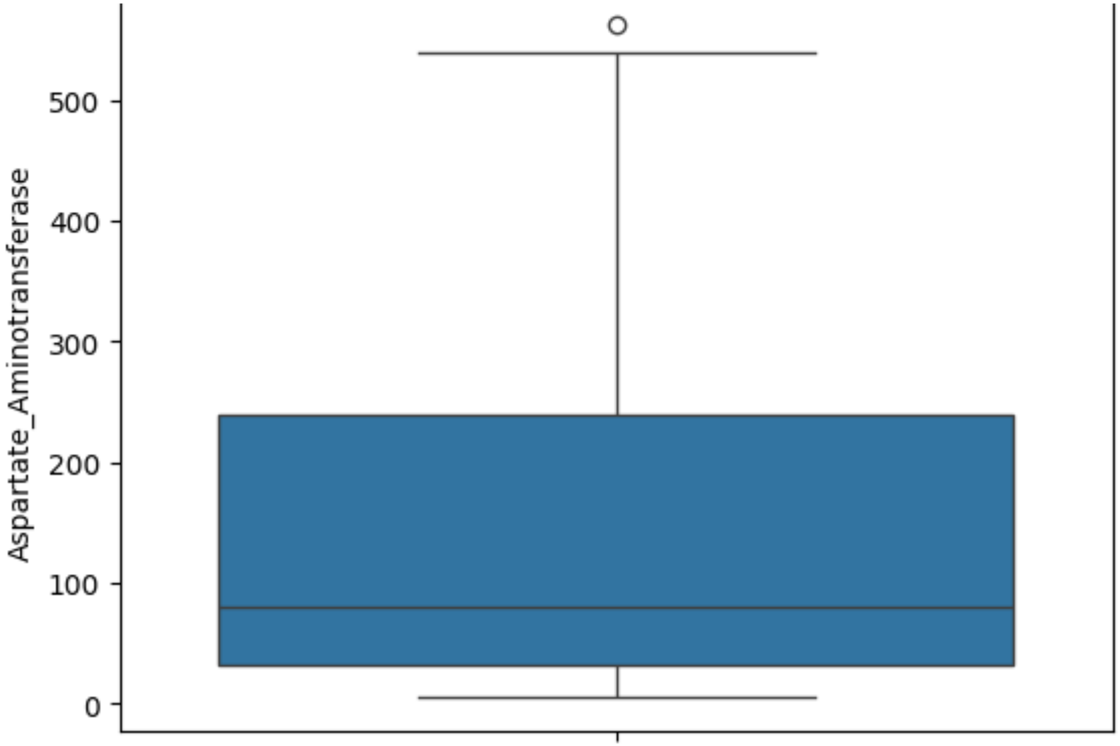
```
# df=df.copy()
```

Data without outliers

```
printBox(df, 'Alkaline_Phosphotase', 'without outliers')
```




```
printBox(df, 'Aspartate_Aminotransferase', 'without outliers')
```



Spliting dataset into training and testing data using train test split

```
from sklearn.model_selection import train_test_split
```

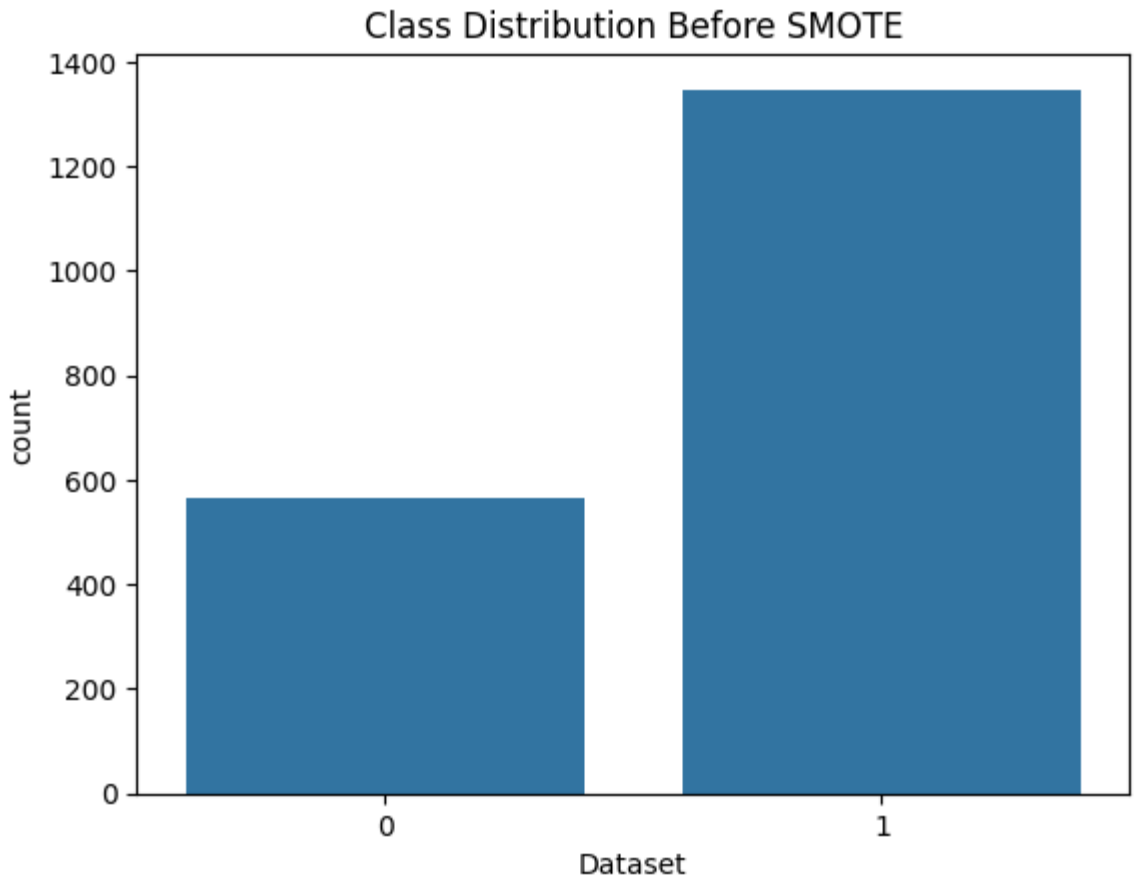
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size = 0.2)
```

Hanlde imbalanced data using SMOTE

```
# y_train.value_counts()[0], y_train.value_counts()[1]
```

```
# # ration = majority / minority
# imbalace_ratio = y_train.value_counts()[1] / y_train.value_counts()[0]
# imbalace_ratio
```

```
sns.countplot(x=y_train)
plt.title('Class Distribution Before SMOTE')
plt.show()
```




```
# !pip install imblearn
```

```
# from imblearn.over_sampling import SMOTE
# smote = SMOTE(random_state=42)

# X_train, y_train = smote.fit_resample(X_train, y_train)
```

```
y_train.value_counts()
```

	count
Dataset	
1	1347
0	565

dtype: int64

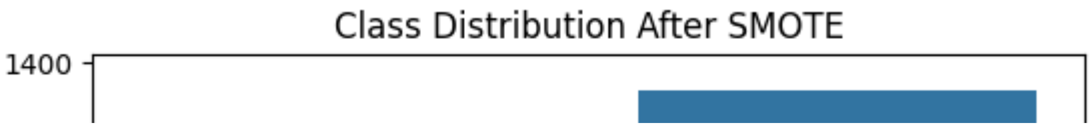
```
imbalace_ratio = y_train.value_counts()[1] / y_train.value_counts()[0]
imbalace_ratio
```

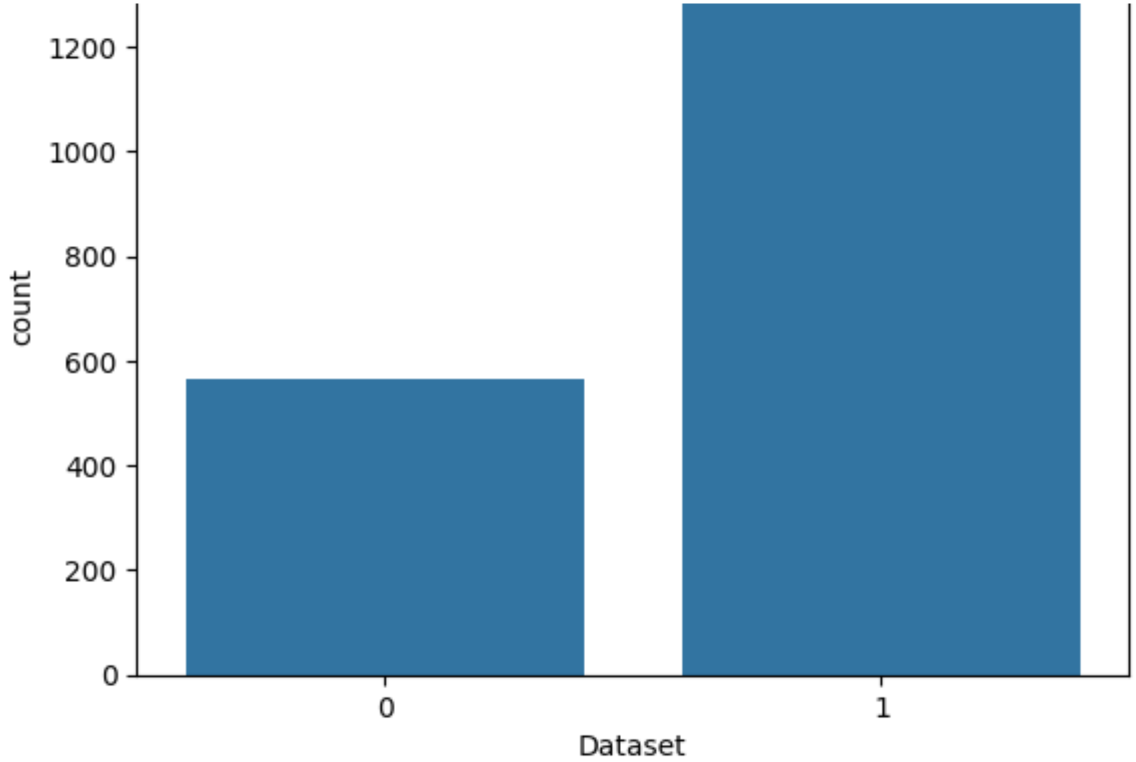
```
np.float64(2.384070796460177)
```

```
y_train.shape
```

```
(1912,)
```

```
sns.countplot(x=y_train)
plt.title('Class Distribution After SMOTE')
plt.show()
```





Standardization using standard scaler

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train_std = scaler.fit_transform(X_train)

X_test_std = scaler.transform(X_test)
```

Evaluation Metrics, Loss and ROC curve functions

```
# Function to plot ROC curve
def rocCurve(y_test, y_pred):
  from sklearn.metrics import roc_curve, auc
  fpr, tpr, thresholds = roc_curve(y_test, y_pred)
  roc_auc = auc(fpr, tpr)

  plt.figure()
  plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
  plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
  plt.xlim([0.0, 1.0])
  plt.ylim([0.0, 1.05])
  plt.xlabel('False Positive Rate')
  plt.ylabel('True Positive Rate')
  plt.title('Receiver Operating Characteristic (ROC)')
  plt.legend(loc="lower right")
  plt.show()


# Function to plot loss
def plot_loss(training_loss, validation_loss):
    epochs = range(1, len(training_loss) + 1)
    plt.plot(epochs, training_loss, 'r', label='Training Loss')
    plt.plot(epochs, validation_loss, 'b', label='Validation Loss')
    plt.title('Training and validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()



# Function for Evaluation Metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
def calculate_metrics(model, X_train_std, X_test_std, y_train, y_test):

  y_train_pred = model.predict(X_train_std)
  y_train_pred_labels = (y_train_pred > 0.5).astype(int)

  training_accuracy = accuracy_score(y_train, y_train_pred_labels)
  training_precision = precision_score(y_train, y_train_pred_labels)
  training_recall = recall_score(y_train, y_train_pred_labels)
  training_f1 = f1_score(y_train, y_train_pred_labels)

  print(f"Training Accuracy: {training_accuracy}")
  print(f"Training Precision: {training_precision}")
  print(f"Training Recall: {training_recall}")
  print(f"Training F1 Score: {training_f1}")

  y_pred = model.predict(X_test_std)
  y_pred_labels = (y_pred > 0.5).astype(int)
  accuracy = accuracy_score(y_test, y_pred_labels)
  precision = precision_score(y_test, y_pred_labels)
  recall = recall_score(y_test, y_pred_labels)
  f1 = f1_score(y_test, y_pred_labels)

  print(f"\nAccuracy: {accuracy}")
  print(f"Precision: {precision}")
  print(f"Recall: {recall}")
  print(f"F1 Score: {f1}")

  return y_pred
```

Machine Learning algorithms

KNN

KNN

```
from sklearn.neighbors import KNeighborsClassifier
```

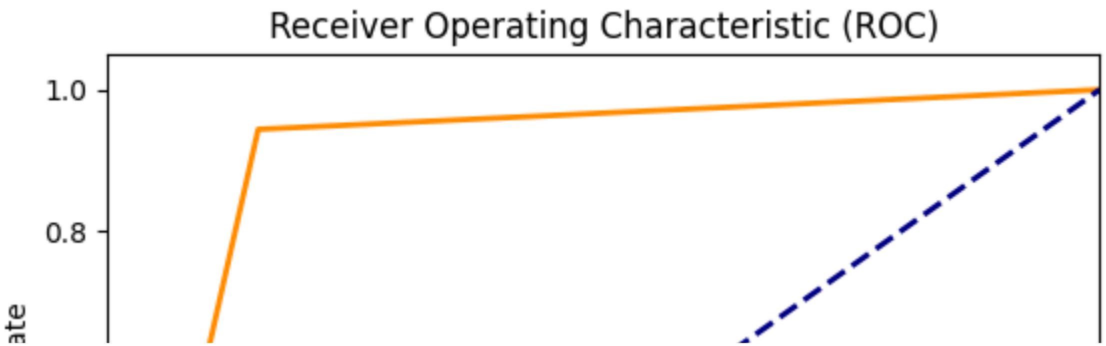
```
knn = KNeighborsClassifier(n_neighbors = 5)

knn_history = knn.fit(X_train_std, y_train)
```

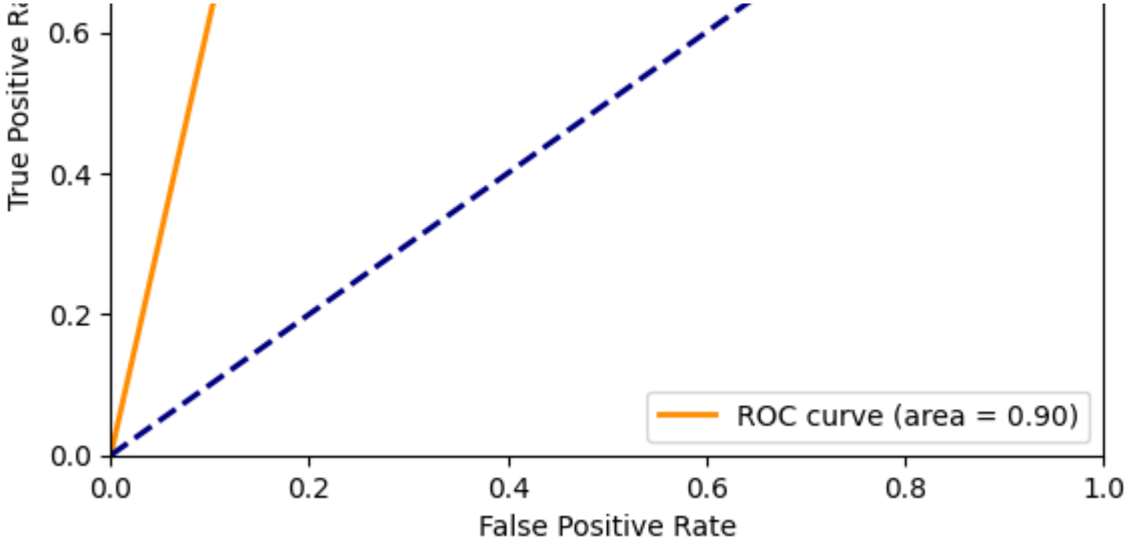
Result

```
knn_y_pred = calculate_metrics(knn, X_train_std, X_test_std, y_train, y_test)
```

```
Training Accuracy: 0.9435146443514645
Training Precision: 0.9538461538461539
Training Recall: 0.9665924276169265
Training F1 Score: 0.9601769911504425

Accuracy: 0.9123173277661796
Precision: 0.926605504587156
Recall: 0.9439252336448598
F1 Score: 0.9351851851851852
```

```
rocCurve(y_test, knn_y_pred)
```





SVM

```
from sklearn.svm import SVC

svm = SVC(kernel='linear')
svm.fit(X_train_std, y_train)
```

▾ SVC

```
SVC(kernel='linear')
```

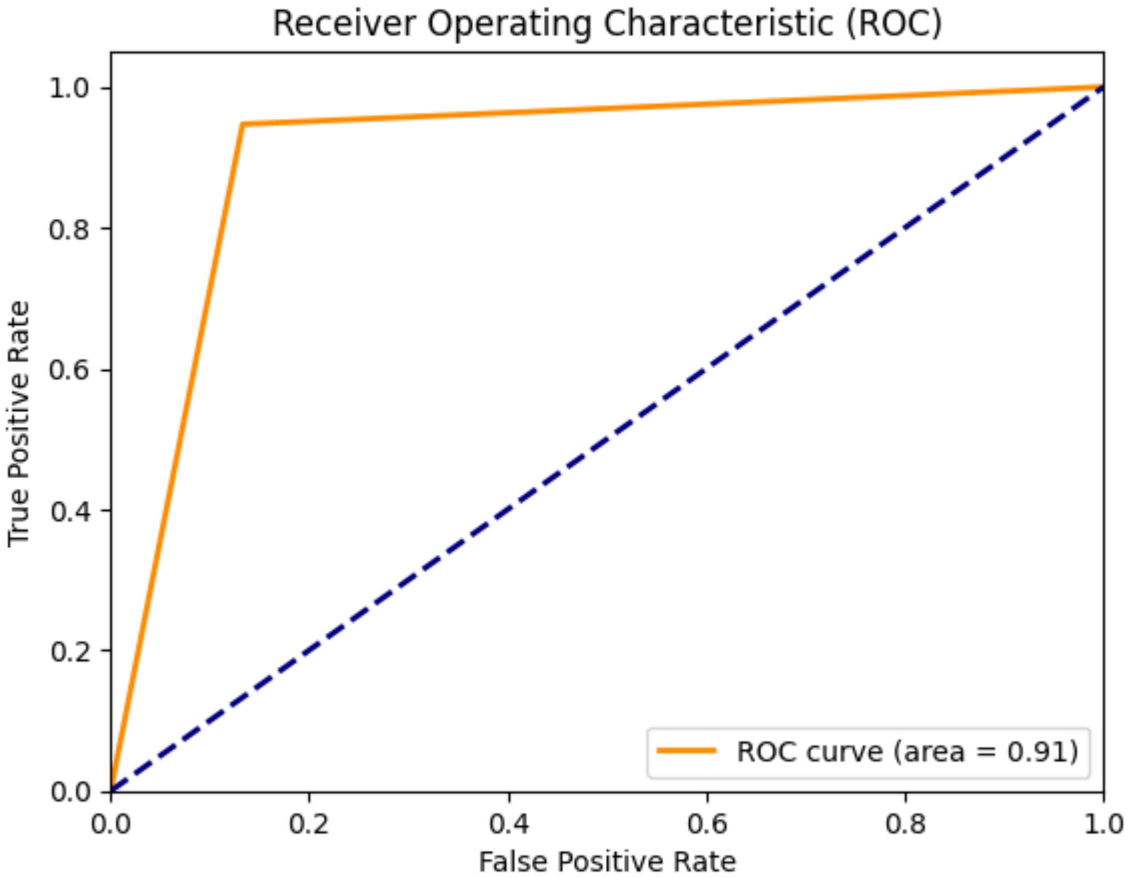
Result

```
svm_y_pred = calculate_metrics(svm, X_train_std, X_test_std, y_train, y_test)
```

```
Training Accuracy: 0.9189330543933054
Training Precision: 0.944113263785395
```

```
Training Precision: 0.944113263785395
Training Recall: 0.9406087602078693
Training F1 Score: 0.942357753811826

Accuracy: 0.9206680584551148
Precision: 0.9353846153846154
Recall: 0.9470404984423676
F1 Score: 0.9411764705882353
```

```
rocCurve(y_test, svm_y_pred)
```



Random Forest

```
from sklearn.ensemble import RandomForestClassifier

rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train_std, y_train)
```

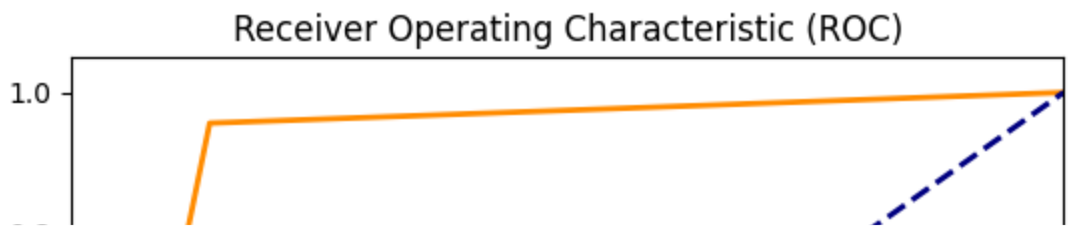
```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

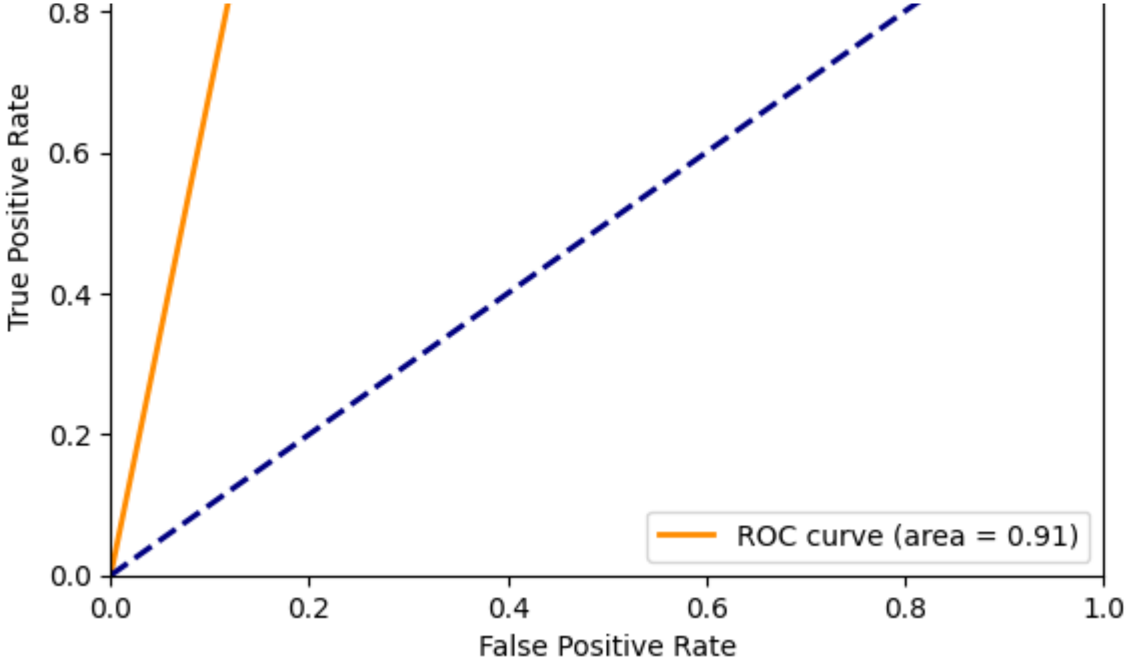
Result

```
rf_y_pred = calculate_metrics(rf_classifier, X_train_std, X_test_std, y_train, y_test)
```

```
Training Accuracy: 1.0
Training Precision: 1.0
Training Recall: 1.0
Training F1 Score: 1.0

Accuracy: 0.9248434237995825
Precision: 0.9331306990881459
Recall: 0.956386292834891
F1 Score: 0.9446153846153846
```

```
rocCurve(y_test, rf_y_pred)
```





Save RF classifier model

```
import joblib


joblib.dump(rf_classifier, 'liver_rf_model.pkl')
```

```
['liver_rf_model.pkl']
```

Creating and Training ANN model using keras

```
EPOCH = 50
```

```
EPOCH = 50
```

```
import tensorflow as tf
tf.random.set_seed(3)
from tensorflow import keras
```

```
tf.__version__
```

```
'2.18.0'
```

```
num_features = X_train_std.shape[1]
print(num_features)
```

```
8
```

```
ann_model = keras.Sequential([
    keras.layers.Dense(10, input_shape=(num_features, ), activation='relu'),
    keras.layers.Dense(20, activation='relu'),
    keras.layers.Dense(40, activation='relu'),
    keras.layers.Dense(80, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])

ann_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history = ann_model.fit(x=X_train_std, y=y_train, validation_data=(X_test_std, y_test), epochs=EPOCH)

ann_training_loss = history.history['loss']
ann_validation_loss = history.history['val_loss']
```

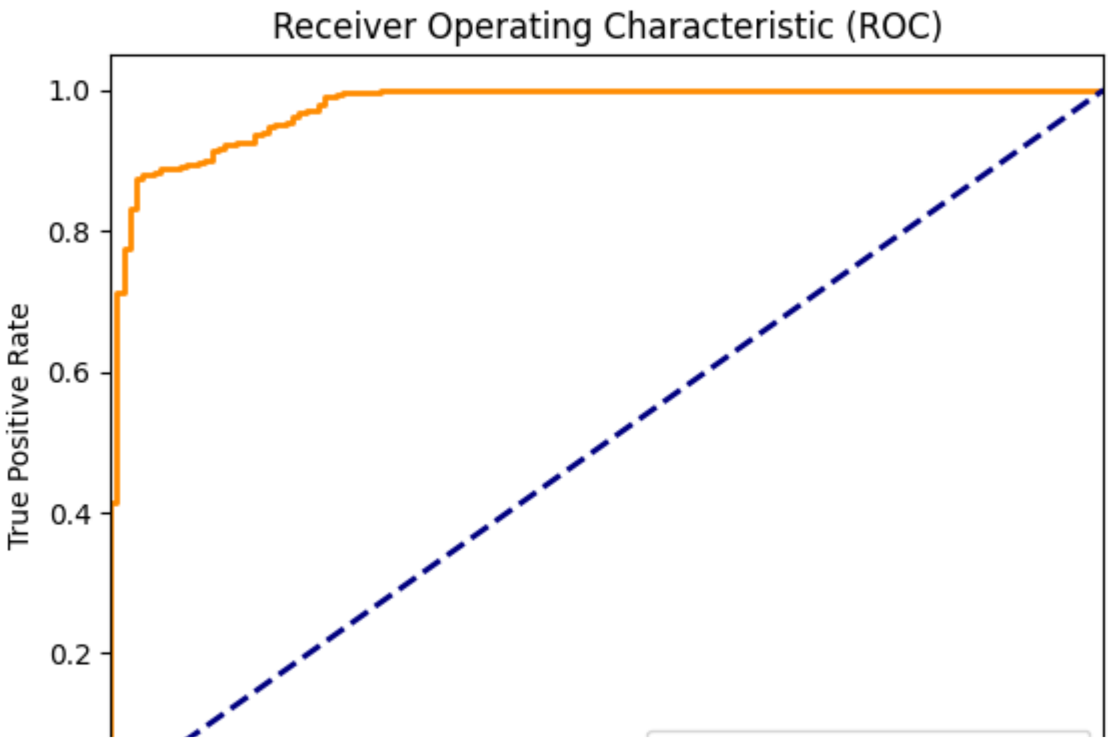
Show hidden output

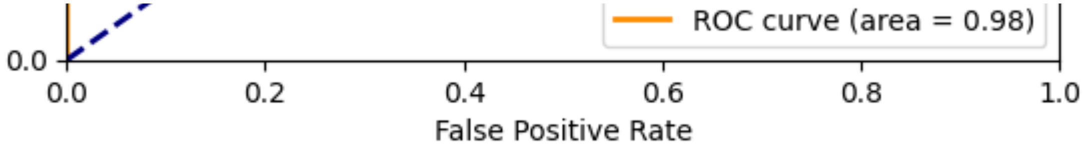
Result

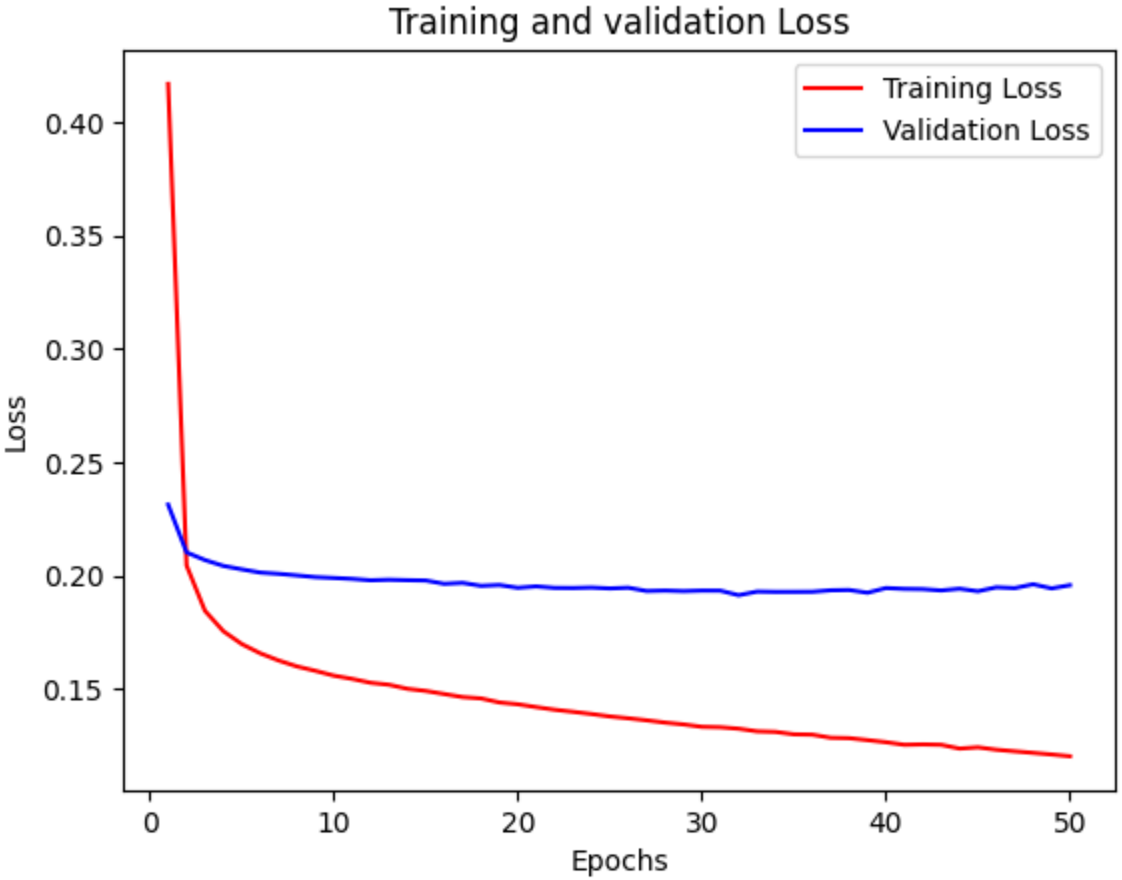
```
ann_y_pred = calculate_metrics(ann_model, X_train_std, X_test_std, y_train, y_test)
```

```
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
Training Accuracy: 0.9382845188284519
Training Precision: 0.9658832448824868
Training Recall: 0.9458054936896808
Training F1 Score: 0.9557389347336834
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step

Accuracy: 0.9018789144050104
Precision: 0.928125
Recall: 0.9252336448598131
F1 Score: 0.9266770670826833
```

```
rocCurve(y_test, ann_y_pred)
plot_loss(ann_training_loss, ann_validation_loss)
```







Added l2 regularization

```
from tensorflow import keras

l2_model = keras.Sequential([
```

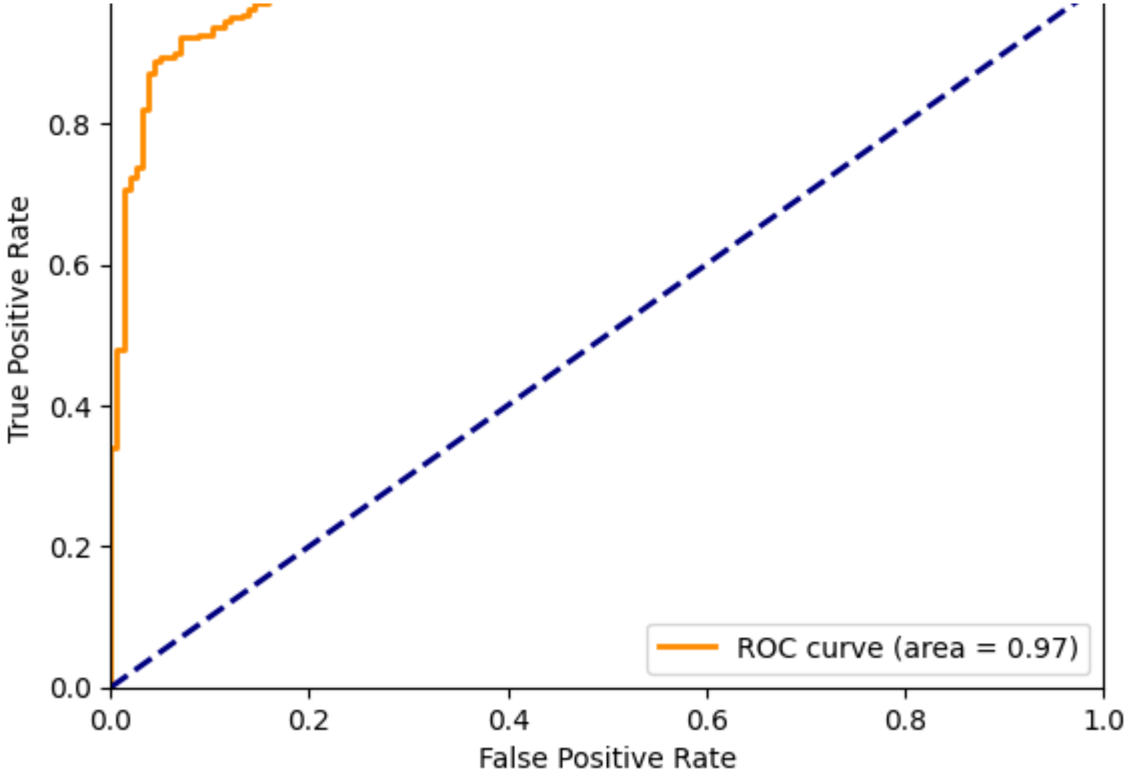
```
l2_model = keras.Sequential([
    keras.layers.Dense(10, input_shape=(num_features, ), activation='relu', kernel_regularizer=keras.regularizers.l2(
    keras.layers.Dense(20, activation='relu', kernel_regularizer=keras.regularizers.l2(0.02)),
    keras.layers.Dense(40, activation='relu', kernel_regularizer=keras.regularizers.l2(0.02)),
    keras.layers.Dense(80, activation='relu', kernel_regularizer=keras.regularizers.l2(0.02)),
    keras.layers.Dense(1, activation='sigmoid')
])


l2_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history = l2_model.fit(x=X_train_std, y=y_train, validation_split=0.1, epochs=EPOCH)

l2_training_loss = history.history['loss']
l2_validation_loss = history.history['val_loss']
```

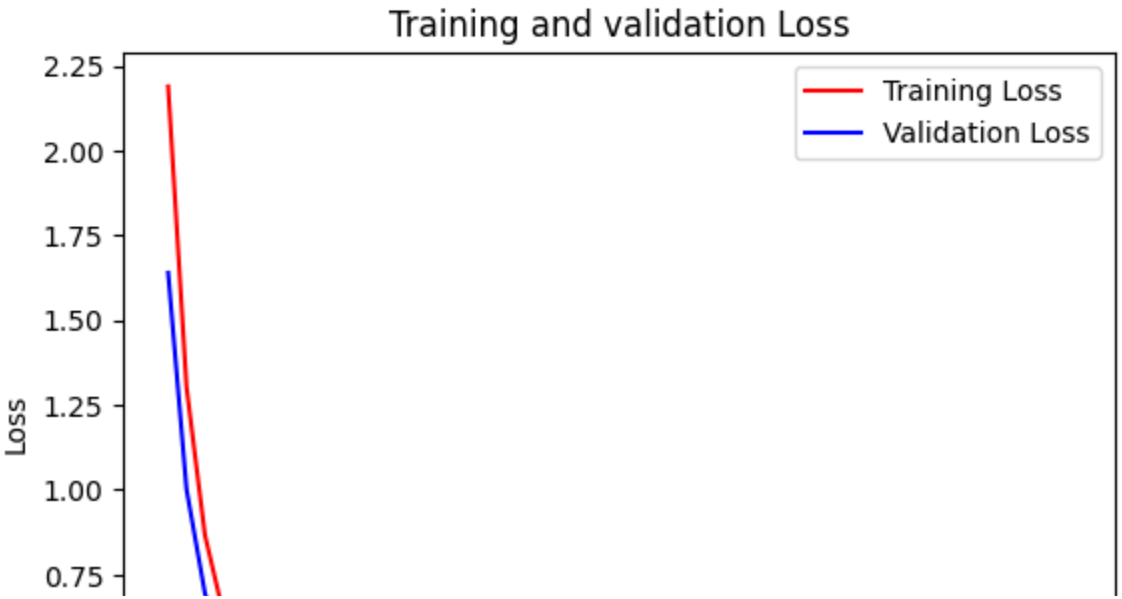
Show hidden output

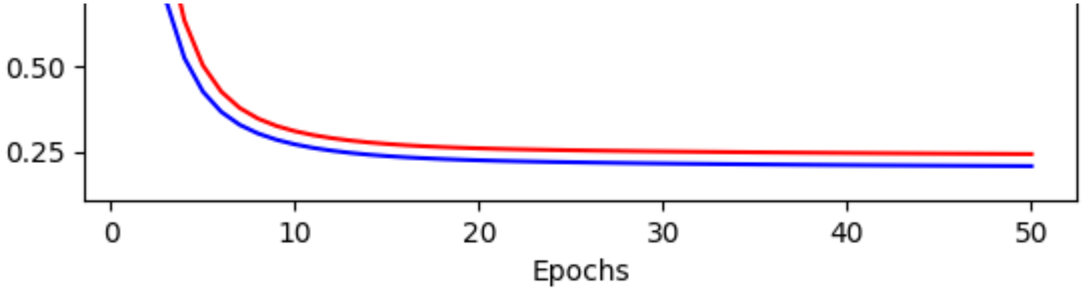
```
l2_y_pred = calculate_metrics(l2_model, X_train_std, X_test_std, y_train, y_test)
```

```
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
Training Accuracy: 0.922071129707113
Training Precision: 0.9450222882615156
Training Recall: 0.9443207126948775
Training F1 Score: 0.9446713702190865
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step

Accuracy: 0.9248434237995825
Precision: 0.9357798165137615
Recall: 0.9532710280373832
F1 Score: 0.9444444444444444
```

```
rocCurve(y_test, l2_y_pred)
plot_loss(l2_training_loss, l2_validation_loss)
```


0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
ROC curve (area = 0.97)


Training and validation Loss
2.25
2.00
1.75
1.50
1.25
1.00
0.75
Loss
Training Loss
Validation Loss



L1,L2 regularization

```
num_features
```

```
8
```

```
l1l2_model = keras.Sequential([

    keras.layers.Input(shape=(num_features,)),
    keras.layers.Dense(10, activation='relu'),
    keras.layers.Dense(20, activation='relu', kernel_regularizer=keras.regularizers.l1_l2(l1=0.01, l2=0.01)),
    keras.layers.Dense(40, activation='relu', kernel_regularizer=keras.regularizers.l1_l2(l1=0.01, l2=0.01)),
    keras.layers.Dense(80, activation='relu', kernel_regularizer=keras.regularizers.l1_l2(l1=0.01, l2=0.01)),
    keras.layers.Dense(1, activation='sigmoid')
])

l1l2_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history = l1l2_model.fit(x=X_train_std, y=y_train, validation_split=0.1, epochs=EPOCH)

l1l2_training_loss = history.history['loss']
l1l2_validation_loss = history.history['val_loss']
```

Show hidden output

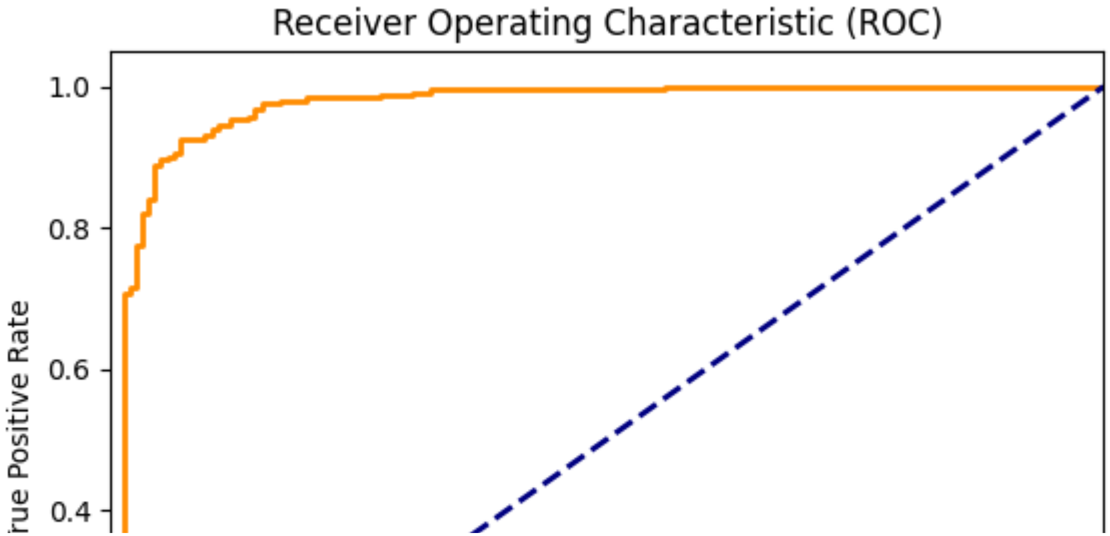
```
l1l2_model.input_shape
```

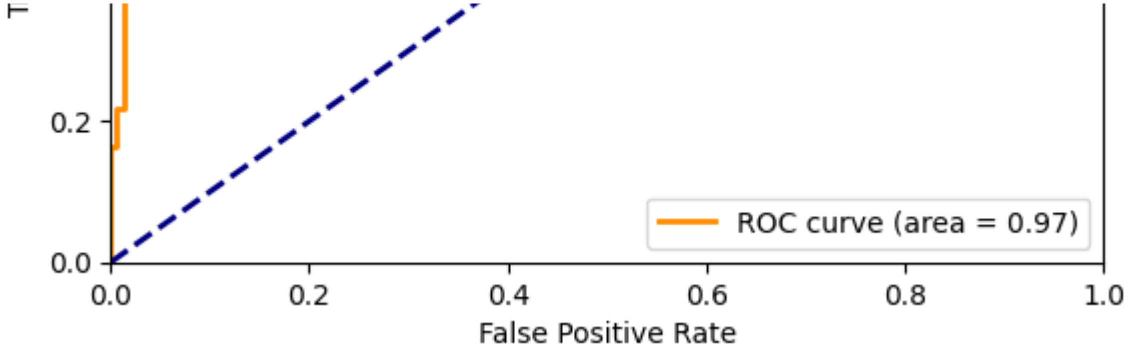
```
(None, 8)
```

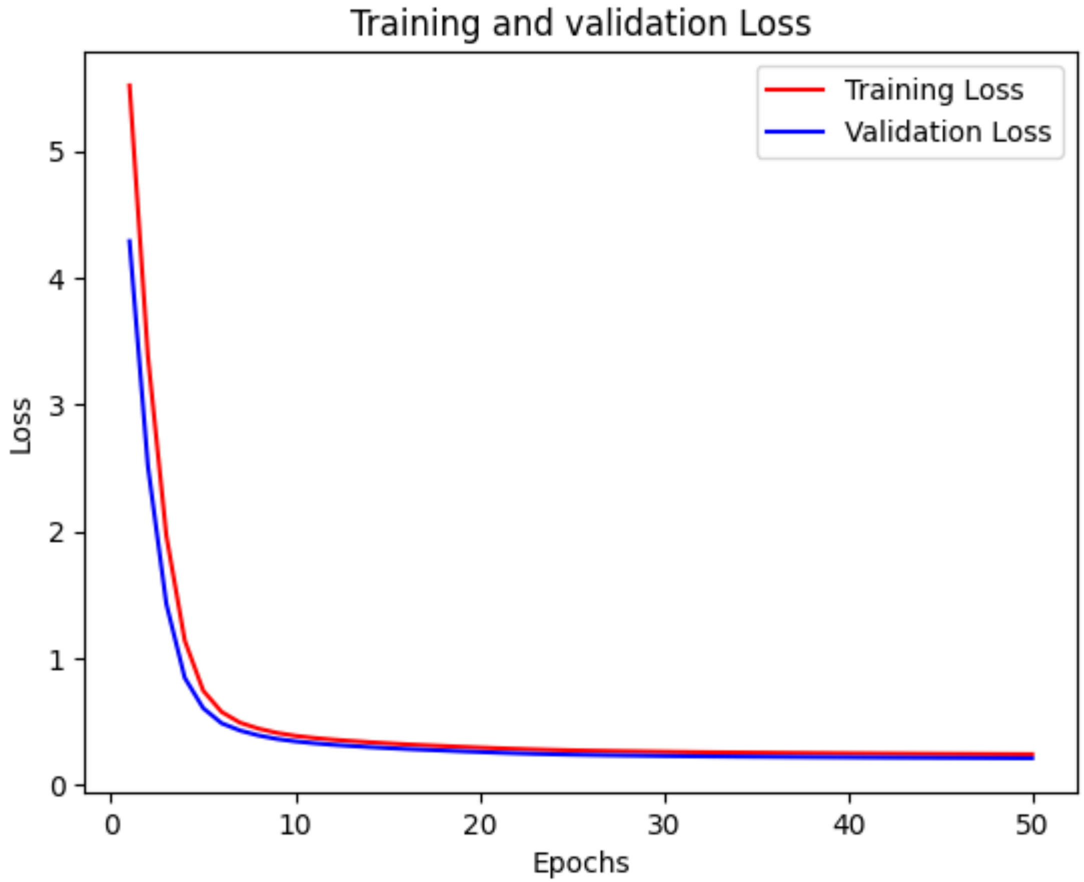
```
l1l2_y_pred = calculate_metrics(l1l2_model, X_train_std, X_test_std, y_train, y_test)
```

```
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
Training Accuracy: 0.9173640167364017
Training Precision: 0.9426656738644825
Training Recall: 0.9398663697104677
Training F1 Score: 0.9412639405204462
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step

Accuracy: 0.9248434237995825
Precision: 0.9411764705882353
Recall: 0.9470404984423676
F1 Score: 0.9440993788819876
```

```
rocCurve(y_test, l1l2_y_pred)
plot_loss(l1l2_training_loss, l1l2_validation_loss)
```







Added dropout layer

Added dropout layer

```
num_features
```

```
8
```

```
num_features = X_train_std.shape[1]

dropout_model = keras.Sequential([
    keras.layers.InputLayer(input_shape=(num_features,)),
    keras.layers.Dense(10, activation='relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(20, activation='relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(40, activation='relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(80, activation='relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(1, activation='sigmoid')
])

dropout_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history = dropout_model.fit(x=X_train_std, y=y_train, validation_split=0.1,epochs=EPOCH)

dropout_training_loss = history.history['loss']
dropout_validation_loss = history.history['val_loss']
```

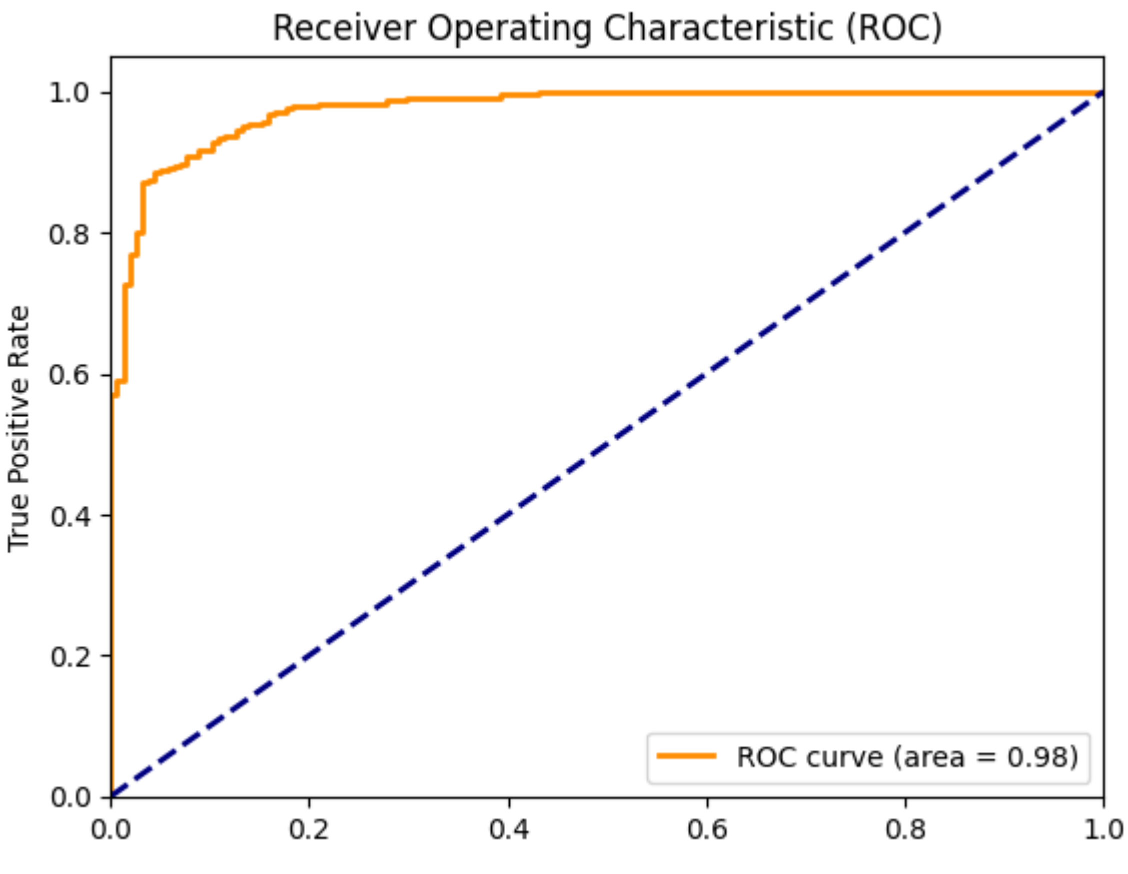
Show hidden output

Result

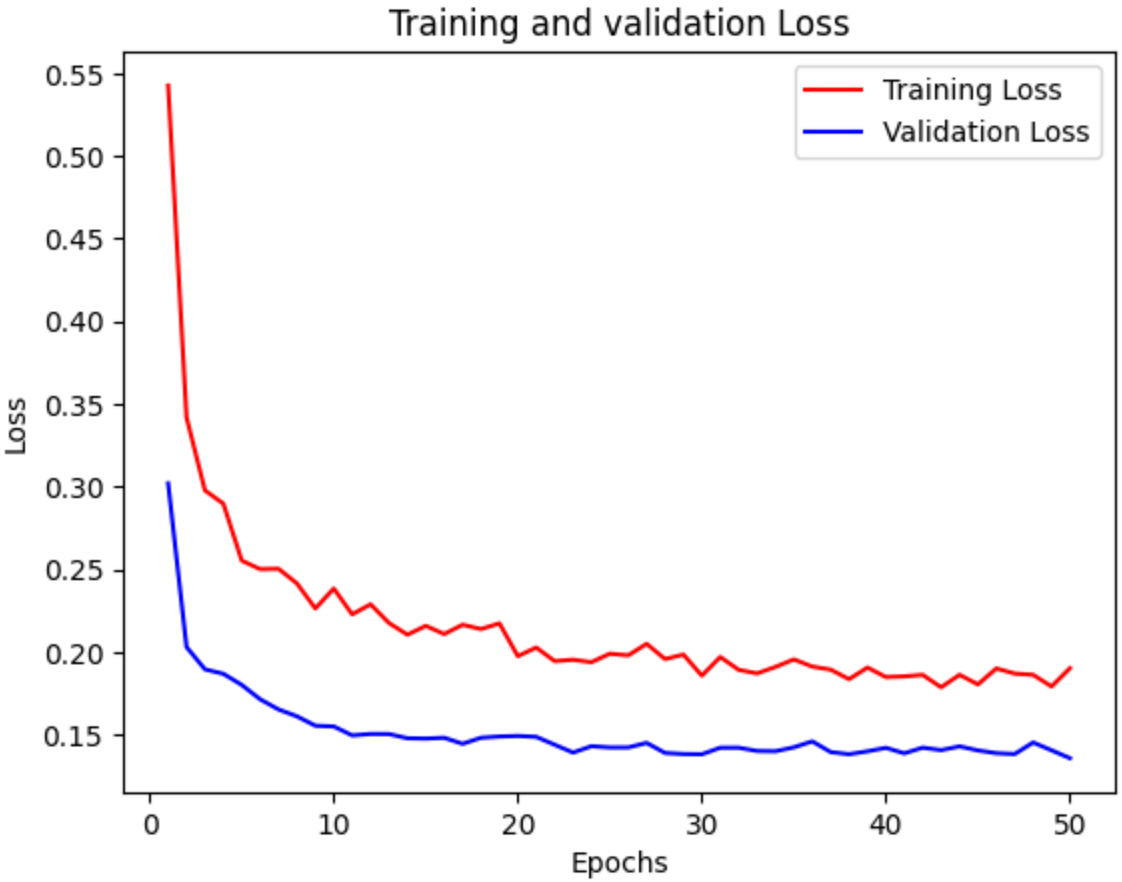
```
dropout_y_pred = calculate_metrics(dropout_model, X_train_std, X_test_std, y_train, y_test)
```

```
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
Training Accuracy: 0.9278242677824268
Training Precision: 0.9314775160599572
Training Recall: 0.9688195991091314
Training F1 Score: 0.9497816593886463
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step

Accuracy: 0.9227557411273486
Precision: 0.9251497005988024
Recall: 0.9626168224299065
F1 Score: 0.9435114503816794
```

```
rocCurve(y_test, dropout_y_pred)
plot_loss(dropout_training_loss, dropout_validation_loss)
```



False Positive Rate



Using CNN

```
X_train_cnn = X_train_std.reshape(X_train_std.shape[0], X_train_std.shape[1], 1)

X_test_cnn = X_test_std.reshape(X_test_std.shape[0], X_test_std.shape[1], 1)


X_train_cnn.shape
```

```
X_train_cnn.shape
```

```
(1912, 8, 1)
```

```
cnnModel = keras.Sequential([
    keras.layers.Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(X_train_cnn.shape[1], 1)),
    keras.layers.MaxPooling1D(pool_size=2),

    keras.layers.Conv1D(filters=64, kernel_size=2, activation='relu', padding='same'),
    keras.layers.MaxPooling1D(pool_size=2),

    keras.layers.Flatten(),
    keras.layers.Dense(1, activation='sigmoid')
])

cnnModel.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

history = cnnModel.fit(X_train_cnn, y_train, epochs=EPOCH, validation_split=0.1, validation_data=(X_test_cnn, y_test))

cnn_training_loss = history.history['loss']
cnn_validation_loss = history.history['val_loss']
```

Show hidden output

```
cnn_y_pred = calculate_metrics(cnnModel, X_train_cnn, X_test_cnn, y_train, y_test)
```

```
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step
Training Accuracy: 0.9278242677824268
Training Precision: 0.9534883720930233
Training Recall: 0.9435783221974758
Training F1 Score: 0.9485074626865672
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step

Accuracy: 0.9206680584551148
Precision: 0.9300911854103343
Recall: 0.9532710280373832
F1 Score: 0.9415384615384615
```

```
rocCurve(y_test, cnn_y_pred)
plot_loss(cnn_training_loss, cnn_validation_loss)
```

Show hidden output

Using multiple convolutional layer and pooling *layer*

```
cnnModel2 = keras.Sequential([
    keras.layers.Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(X_train_cnn.shape[1], 1), padding='
    keras.layers.MaxPooling1D(pool_size=2),

    keras.layers.Conv1D(filters=64, kernel_size=2, activation='relu', padding='same'),
    keras.layers.MaxPooling1D(pool_size=2),


    keras.layers.Conv1D(filters=128, kernel_size=2, activation='relu', padding='same'),
    keras.layers.MaxPooling1D(pool_size=1),

    keras.layers.Flatten(),
    keras.layers.Dense(1, activation='sigmoid')
])

cnnModel2.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

history = cnnModel2.fit(X_train_cnn, y_train, epochs=EPOCH, validation_split=0.1, validation_data=(X_test_cnn, y_test)

training_loss = history.history['loss']
validation_loss = history.history['val_loss']
```

```
Epoch 1/50
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
60/60 ━━━━━━━━━━━━━━━━━━━━ 4s 34ms/step - accuracy: 0.7834 - loss: 0.4668 - val_accuracy: 0.9207 - val_loss: 0.219
```

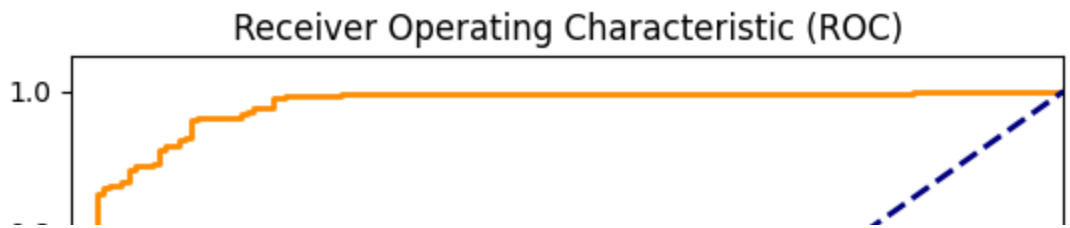
```
Epoch 2/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9219 - loss: 0.2074 - val_accuracy: 0.9207 - val_loss: 0.2109
Epoch 3/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9210 - loss: 0.1960 - val_accuracy: 0.9290 - val_loss: 0.2036
Epoch 4/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9222 - loss: 0.1862 - val_accuracy: 0.9269 - val_loss: 0.1966
Epoch 5/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9249 - loss: 0.1775 - val_accuracy: 0.9290 - val_loss: 0.1909
Epoch 6/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9255 - loss: 0.1706 - val_accuracy: 0.9311 - val_loss: 0.1867
Epoch 7/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9211 - loss: 0.1652 - val_accuracy: 0.9248 - val_loss: 0.1832
Epoch 8/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9161 - loss: 0.1610 - val_accuracy: 0.9228 - val_loss: 0.1812
Epoch 9/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9170 - loss: 0.1576 - val_accuracy: 0.9228 - val_loss: 0.1802
Epoch 10/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9166 - loss: 0.1550 - val_accuracy: 0.9248 - val_loss: 0.1797
Epoch 11/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9175 - loss: 0.1527 - val_accuracy: 0.9248 - val_loss: 0.1791
Epoch 12/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9173 - loss: 0.1513 - val_accuracy: 0.9207 - val_loss: 0.1793
Epoch 13/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9192 - loss: 0.1499 - val_accuracy: 0.9186 - val_loss: 0.1797
Epoch 14/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9209 - loss: 0.1486 - val_accuracy: 0.9186 - val_loss: 0.1798
Epoch 15/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9220 - loss: 0.1477 - val_accuracy: 0.9186 - val_loss: 0.1804
Epoch 16/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9248 - loss: 0.1468 - val_accuracy: 0.9186 - val_loss: 0.1806
Epoch 17/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9242 - loss: 0.1462 - val_accuracy: 0.9165 - val_loss: 0.1815
Epoch 18/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9250 - loss: 0.1454 - val_accuracy: 0.9186 - val_loss: 0.1814
Epoch 19/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9221 - loss: 0.1448 - val_accuracy: 0.9207 - val_loss: 0.1817
Epoch 20/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9228 - loss: 0.1440 - val_accuracy: 0.9165 - val_loss: 0.1822
Epoch 21/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9210 - loss: 0.1436 - val_accuracy: 0.9186 - val_loss: 0.1819
```

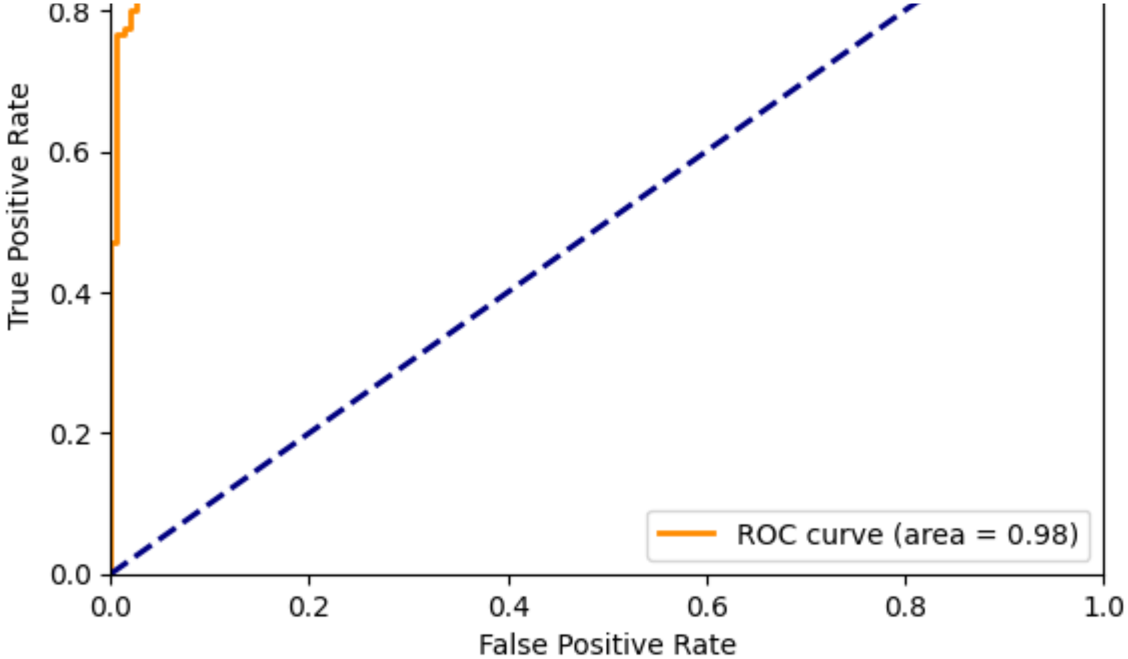
```
Epoch 22/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9217 - loss: 0.1431 - val_accuracy: 0.9186 - val_loss: 0.1822
Epoch 23/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9242 - loss: 0.1424 - val_accuracy: 0.9228 - val_loss: 0.1824
Epoch 24/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9226 - loss: 0.1418 - val_accuracy: 0.9228 - val_loss: 0.1822
Epoch 25/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9226 - loss: 0.1412 - val_accuracy: 0.9228 - val_loss: 0.1825
Epoch 26/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9228 - loss: 0.1408 - val_accuracy: 0.9228 - val_loss: 0.1834
Epoch 27/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9249 - loss: 0.1405 - val_accuracy: 0.9228 - val_loss: 0.1837
Epoch 28/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9249 - loss: 0.1401 - val_accuracy: 0.9228 - val_loss: 0.1838
```

```
cnn2_y_pred = calculate_metrics(cnnModel2, X_train_std, X_test_std, y_train, y_test)
```

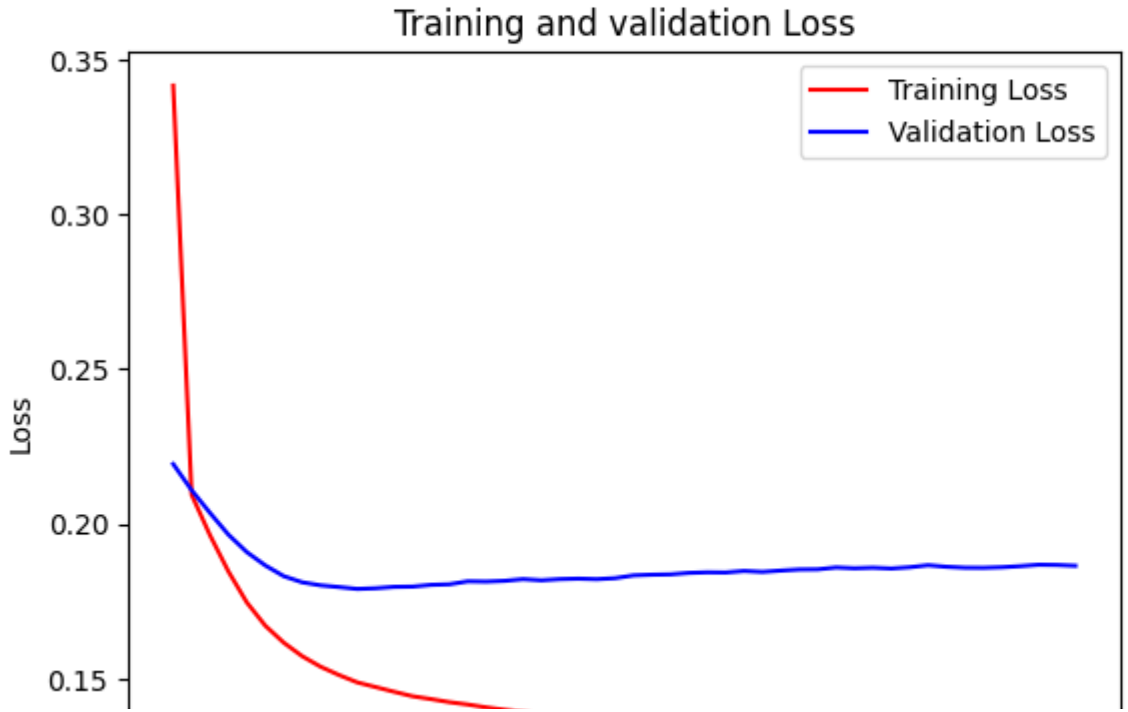
```
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step
Training Accuracy: 0.9388075313807531
Training Precision: 0.9617117117117117
Training Recall: 0.9510022271714922
Training F1 Score: 0.9563269876819709
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step

Accuracy: 0.9311064718162839
Precision: 0.9390243902439024
Recall: 0.9595015576323987
F1 Score: 0.9491525423728814
```

```
rocCurve(y_test, cnn2_y_pred)
plot_loss(training_loss, validation_loss)
```

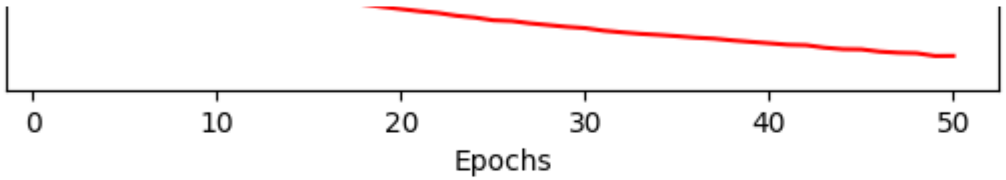




True Positive Rate
False Positive Rate
ROC curve (area = 0.98)

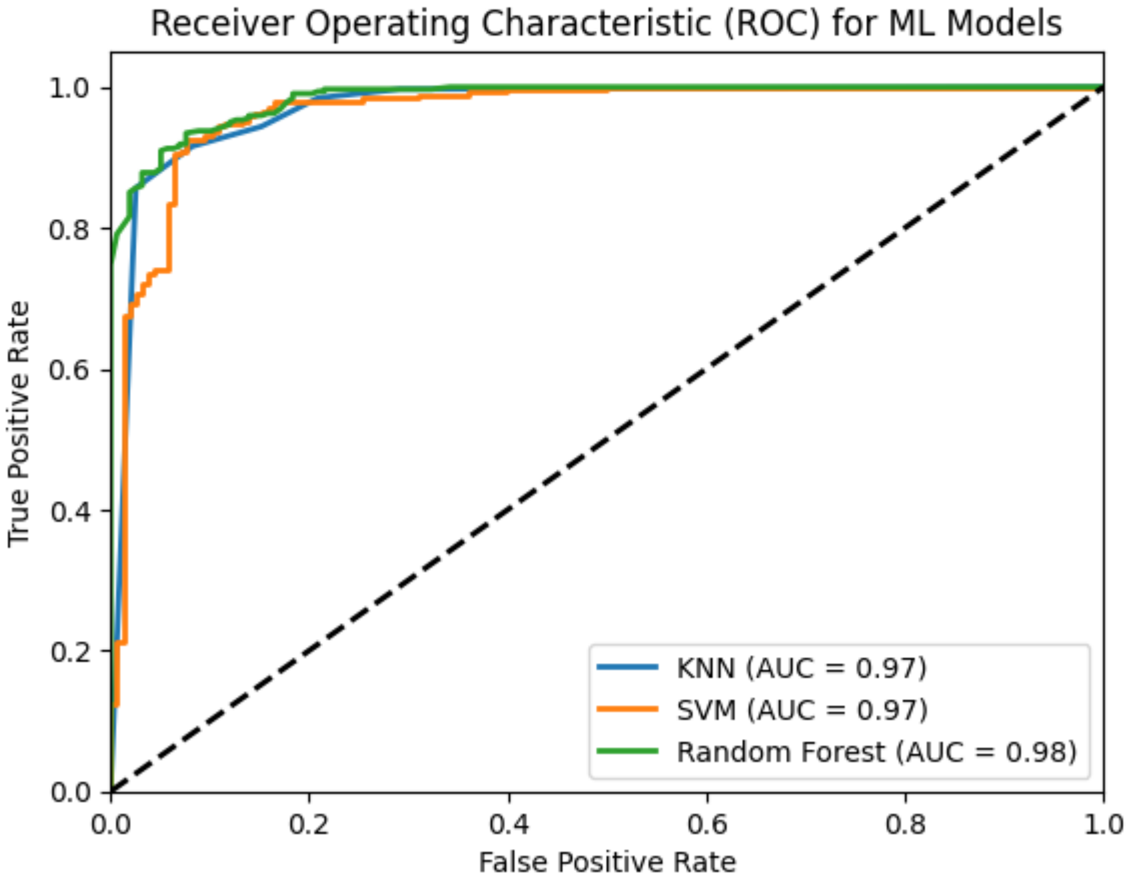


Training and validation Loss
Loss
Training Loss
Validation Loss



Plot ROC curve for all the models

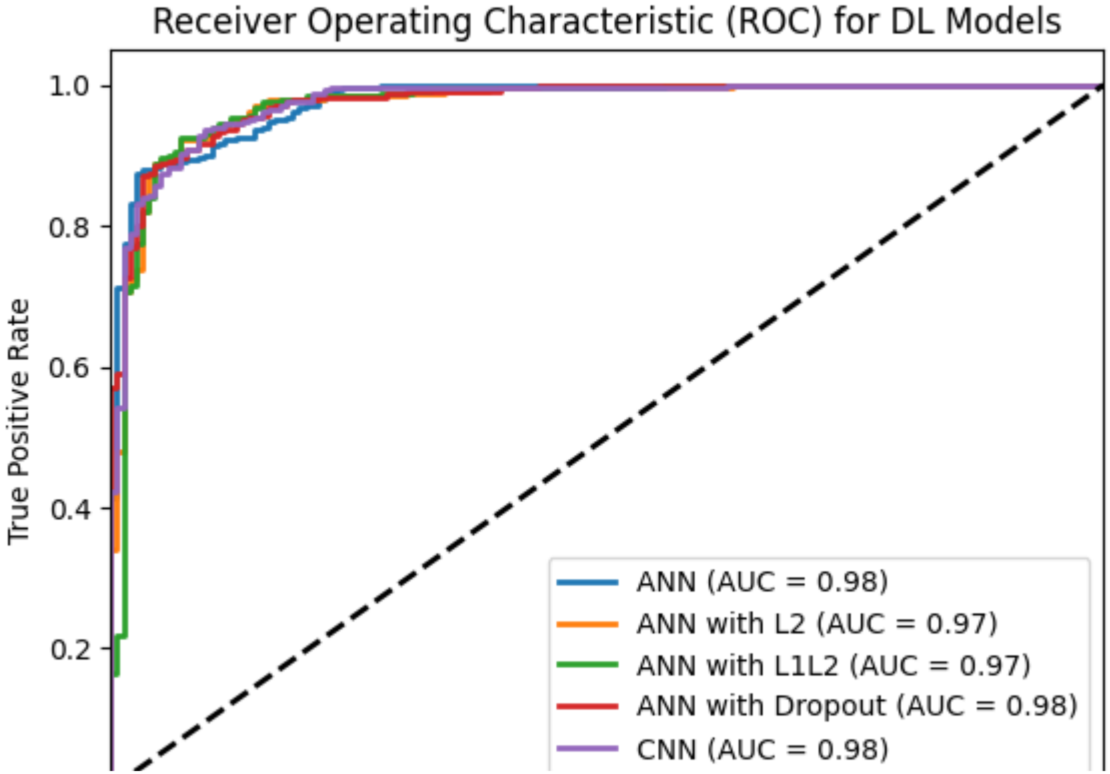
```
# plot ROC in single figure
from sklearn.metrics import roc_curve, auc
def plot_roc_curve(y_test, y_pred_prob, model_name):
    fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, lw=2, label=f'{model_name} (AUC = {roc_auc:.2f})')


# ML
knn_y_pred_prob = knn.predict_proba(X_test_std)[:, 1]
svm_y_pred_prob = svm.decision_function(X_test_std)
rf_y_pred_prob = rf_classifier.predict_proba(X_test_std)[:, 1]

plot_roc_curve(y_test, knn_y_pred_prob, 'KNN')
plot_roc_curve(y_test, svm_y_pred_prob, 'SVM')
plot_roc_curve(y_test, rf_y_pred_prob, 'Random Forest')

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) for ML Models')
plt.legend(loc="lower right")
plt.show()
```

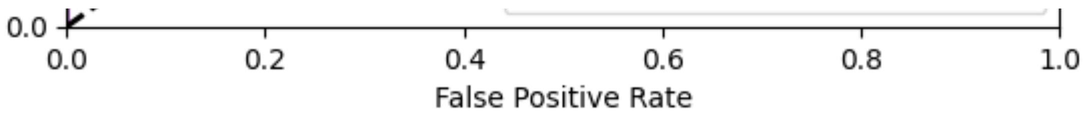



```
# DL
ann_y_pred_prob = ann_model.predict(X_test_std).ravel()
l2_y_pred_prob = l2_model.predict(X_test_std).ravel()
l1l2_y_pred_prob = l1l2_model.predict(X_test_std).ravel()
dropout_y_pred_prob = dropout_model.predict(X_test_std).ravel()
cnn_y_pred_prob = cnnModel.predict(X_test_cnn).ravel()

plot_roc_curve(y_test, ann_y_pred_prob, 'ANN')
plot_roc_curve(y_test, l2_y_pred_prob, 'ANN with L2')
plot_roc_curve(y_test, l1l2_y_pred_prob, 'ANN with L1L2')
plot_roc_curve(y_test, dropout_y_pred_prob, 'ANN with Dropout')
plot_roc_curve(y_test, cnn_y_pred_prob, 'CNN')
```

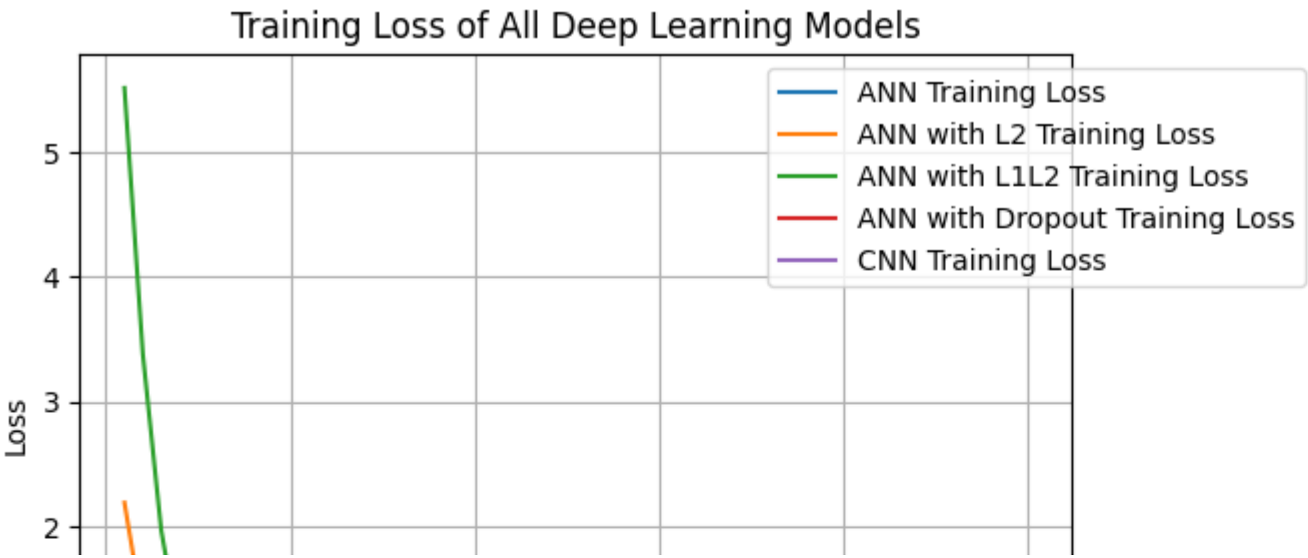
```
plot_roc_curve(y_test, cnn_y_pred_prob, 'CNN')

# plt.figure(figsize=(10, 8))
plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) for DL Models')
plt.legend(loc="lower right")
plt.show()
```

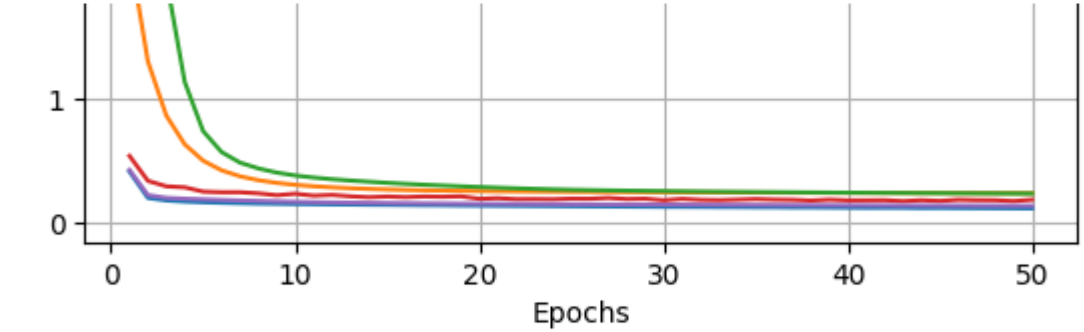
```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```



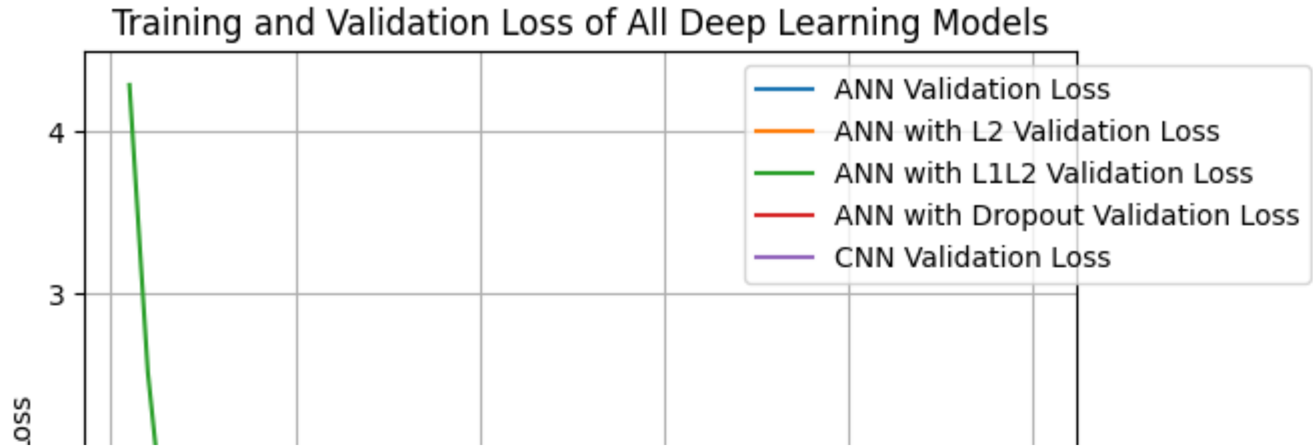


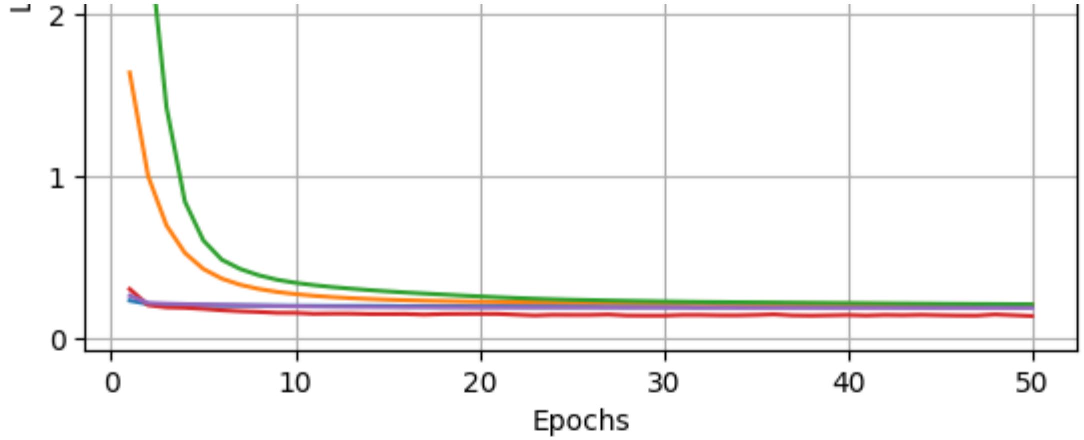

```
epochs = range(1, EPOCH + 1)

plt.plot(epochs, ann_training_loss, label='ANN Training Loss')
plt.plot(epochs, l2_training_loss, label='ANN with L2 Training Loss')
plt.plot(epochs, l1l2_training_loss, label='ANN with L1L2 Training Loss')
plt.plot(epochs, dropout_training_loss, label='ANN with Dropout Training Loss')
plt.plot(epochs, cnn_training_loss, label='CNN Training Loss')


plt.title('Training Loss of All Deep Learning Models')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc='upper right', bbox_to_anchor=(1.25, 1))
plt.grid(True)
plt.show()
```






```
plt.plot(epochs, ann_validation_loss, label='ANN Validation Loss')
plt.plot(epochs, l2_validation_loss, label='ANN with L2 Validation Loss')
plt.plot(epochs, l1l2_validation_loss, label='ANN with L1L2 Validation Loss')
plt.plot(epochs, dropout_validation_loss, label='ANN with Dropout Validation Loss')
plt.plot(epochs, cnn_validation_loss, label='CNN Validation Loss')


plt.title('Training and Validation Loss of All Deep Learning Models')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc='upper right', bbox_to_anchor=(1.25, 1))
plt.grid(True)
plt.show()
```

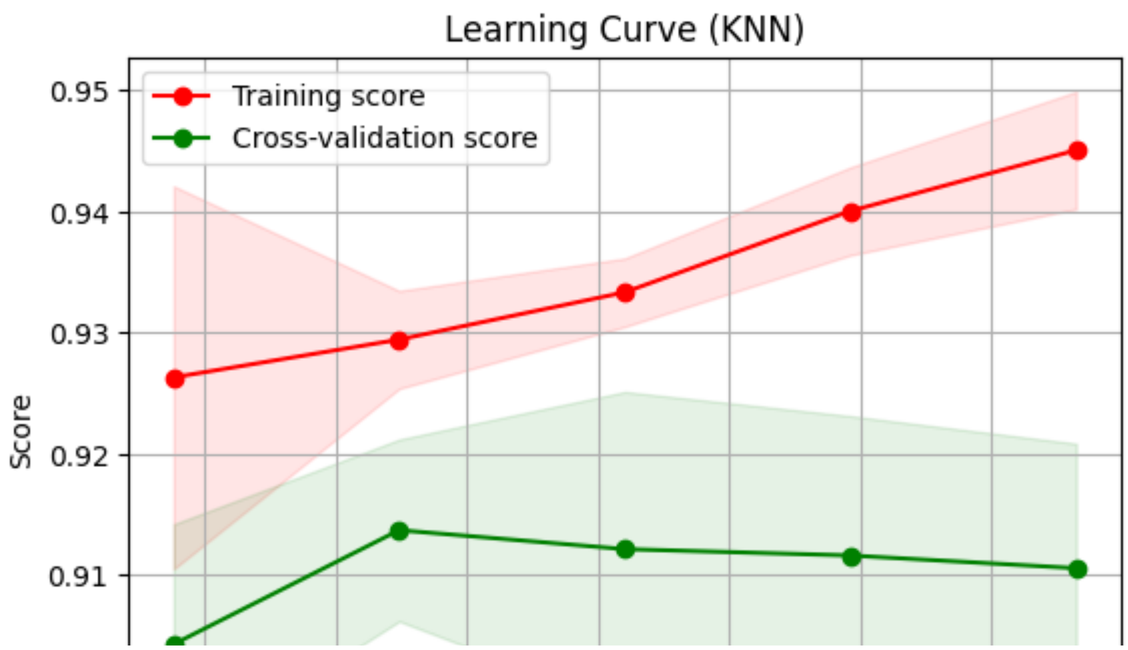


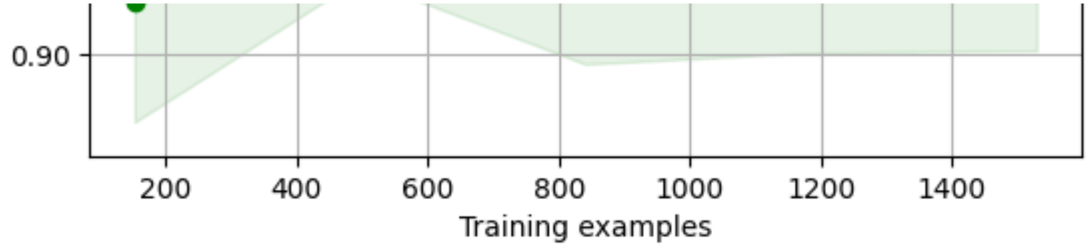


Plot Leraning Curve fot ML models

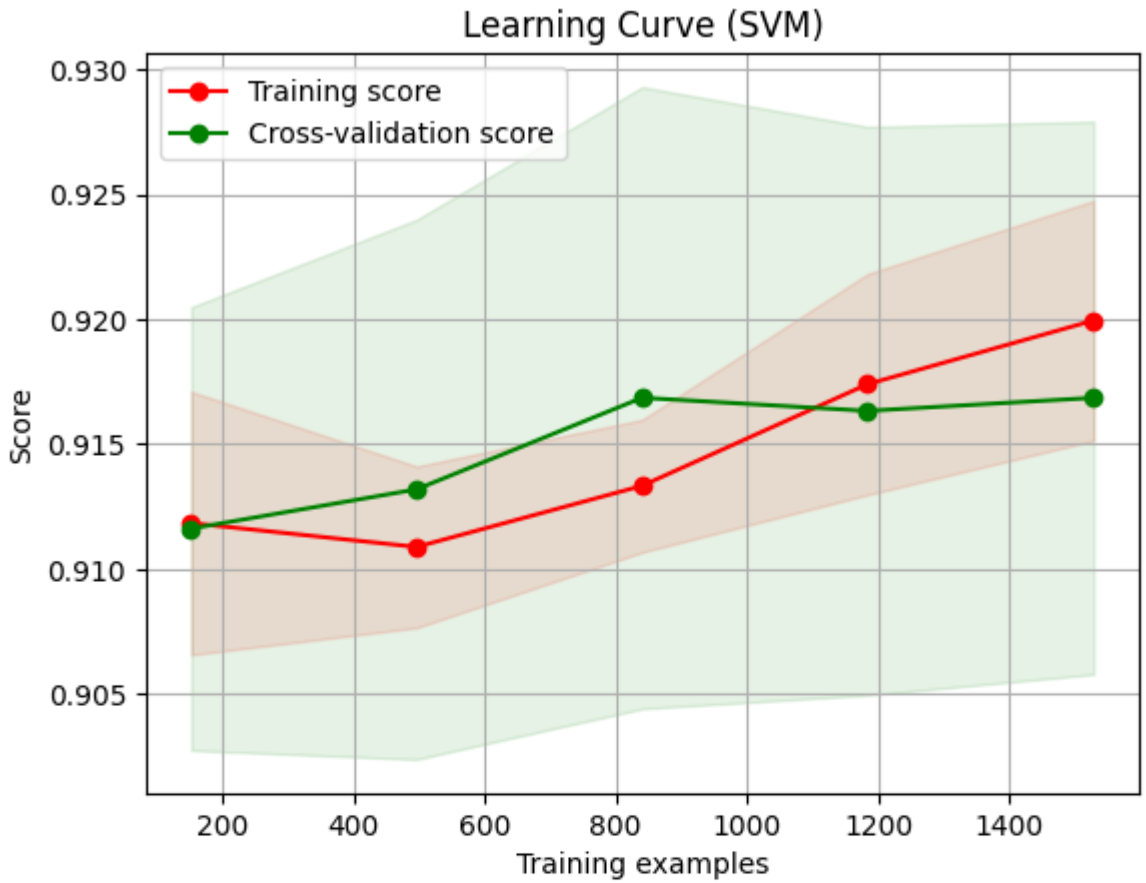
```
from sklearn.model_selection import learning_curve

def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):

    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv,
        n_jobs=n_jobs,
        train_sizes=train_sizes)

    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
```

```
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt


plot_learning_curve(knn, "Learning Curve (KNN)", X_train_std, y_train, cv=5)
plt.show()
```

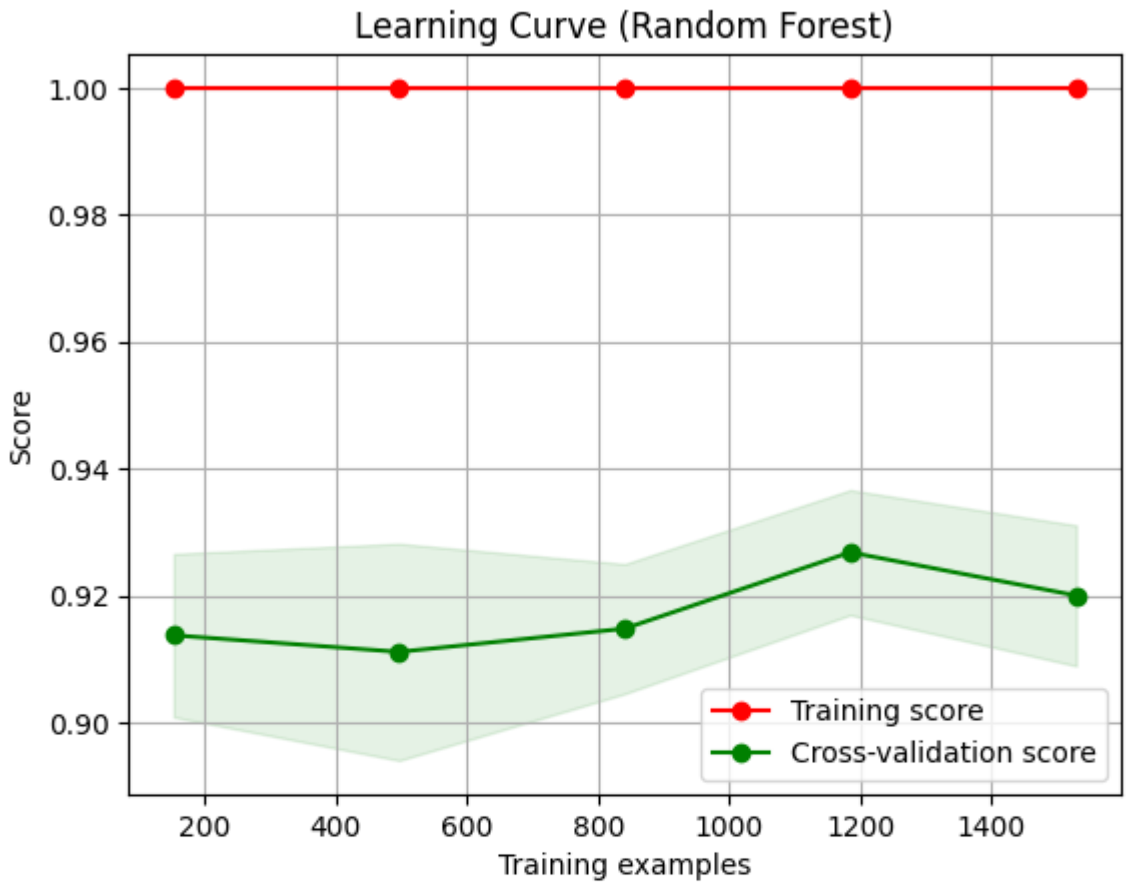





```
plot_learning_curve(svm, "Learning Curve (SVM)", X_train_std, y_train, cv=5)
plt.show()
```



```
plot_learning_curve(rf_classifier, "Learning Curve (Random Forest)", X_train_std, y_train, cv=5)
plt.show()
```



Start coding or generate with AI.