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

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

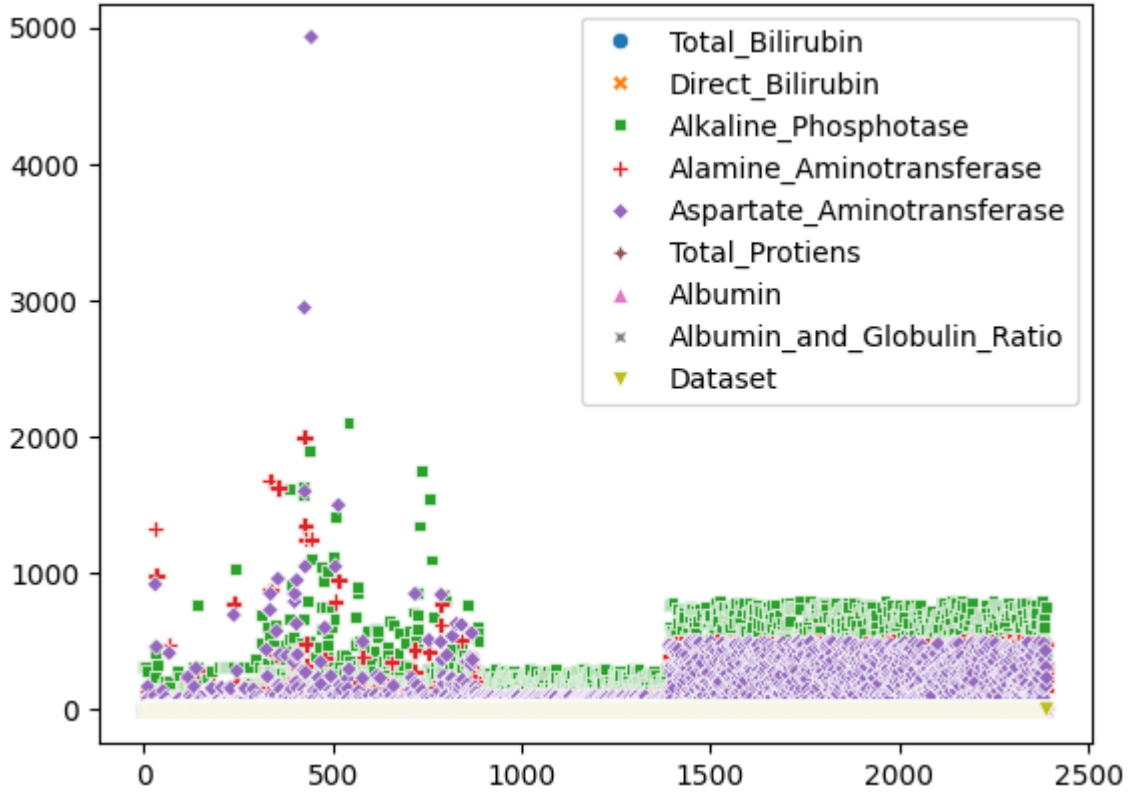
```
y.value_counts()[0], y.value_counts()[1]
```

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

Finding outlier using scatter plot

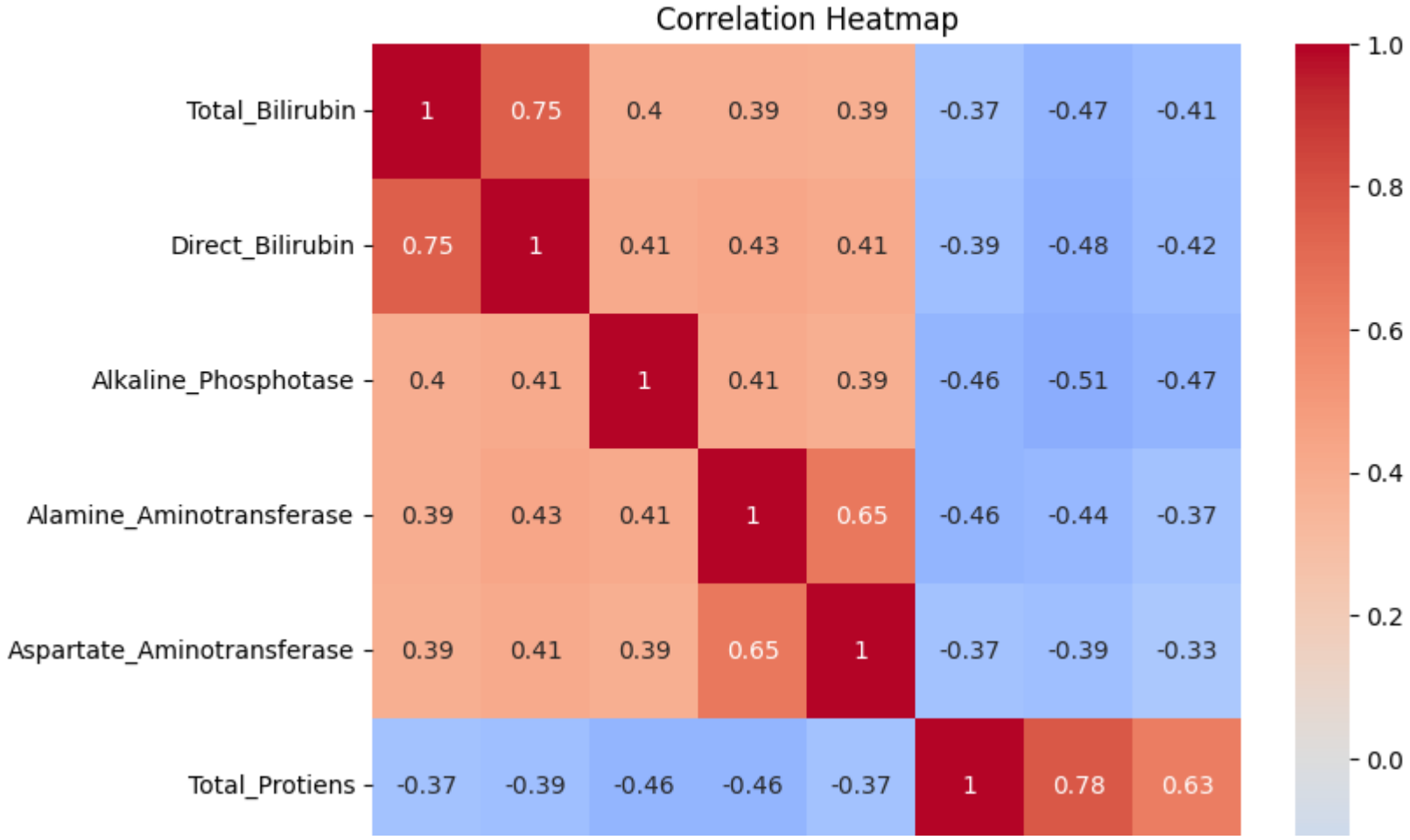
```
sns.scatterplot(data=df)
```

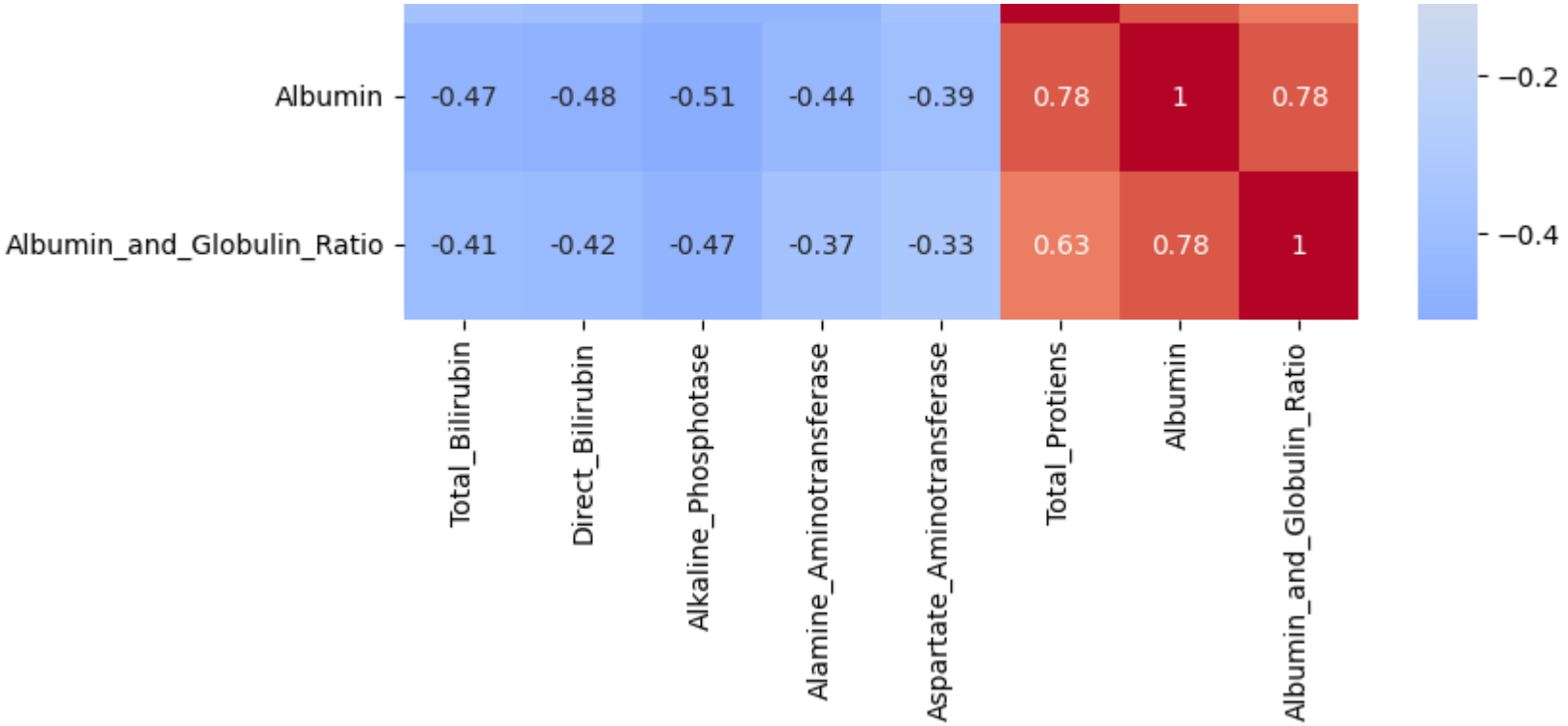
```
<Axes: >
```



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()
```



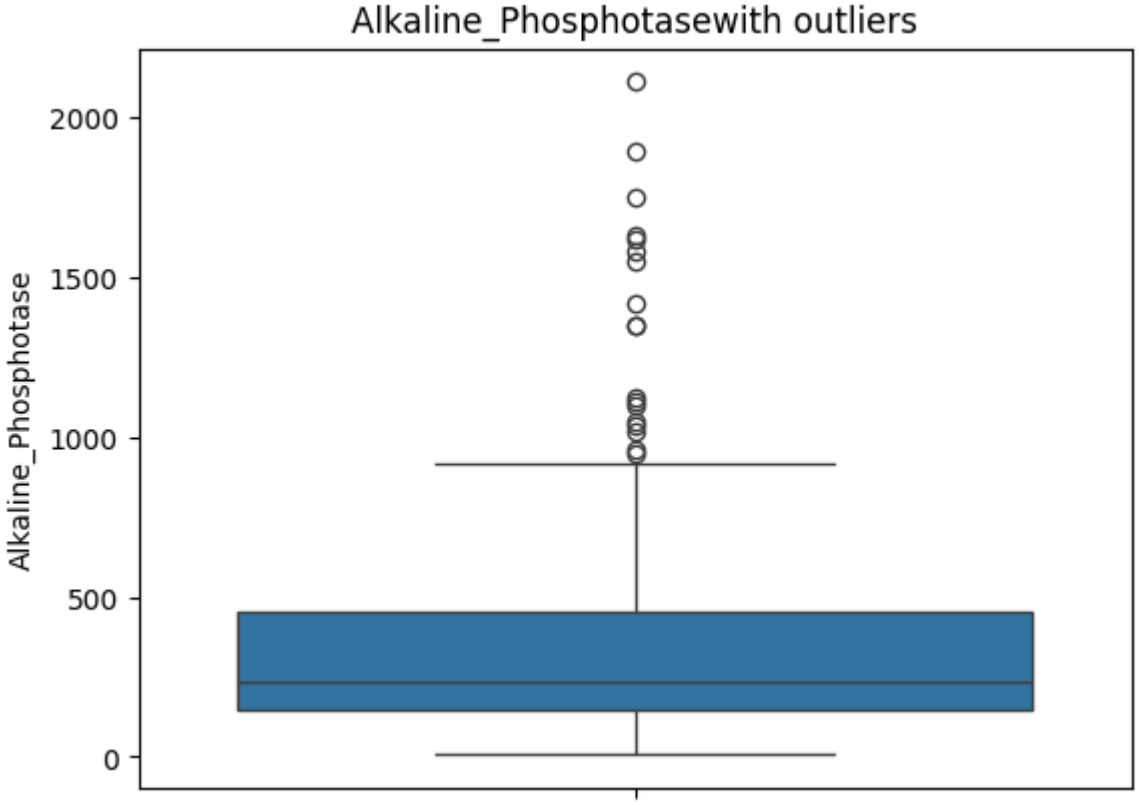


Double-click (or enter) to edit

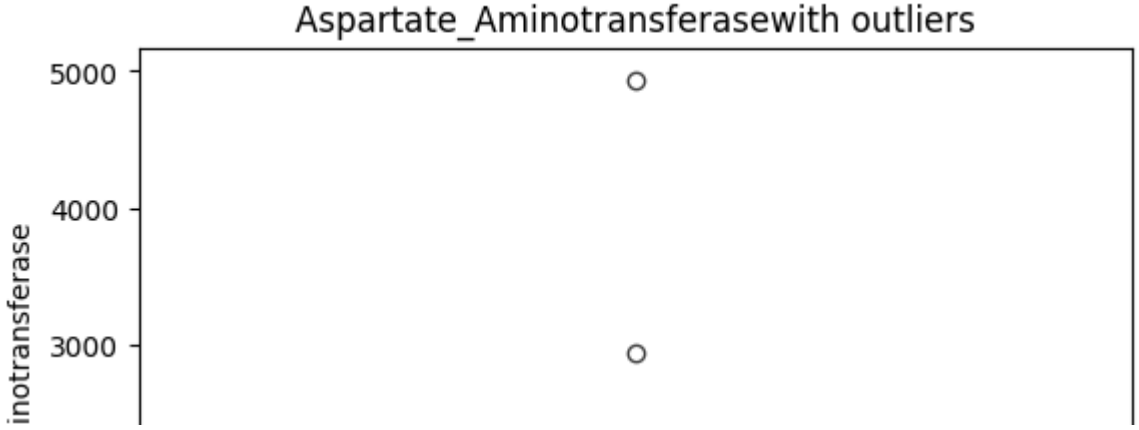
```
# function for box plot
def printBox(df, col, title):
  sns.boxplot(df[col])
  plt.title(col + title)
```

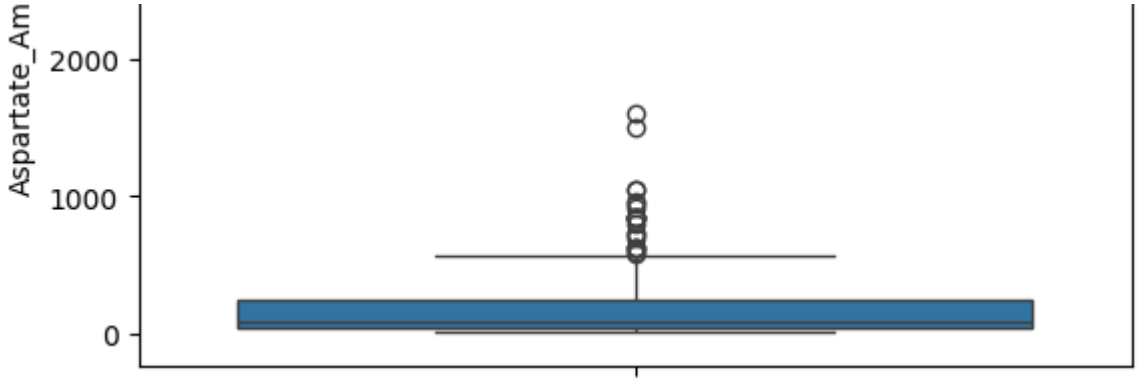
Finding outlier using box plot

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




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





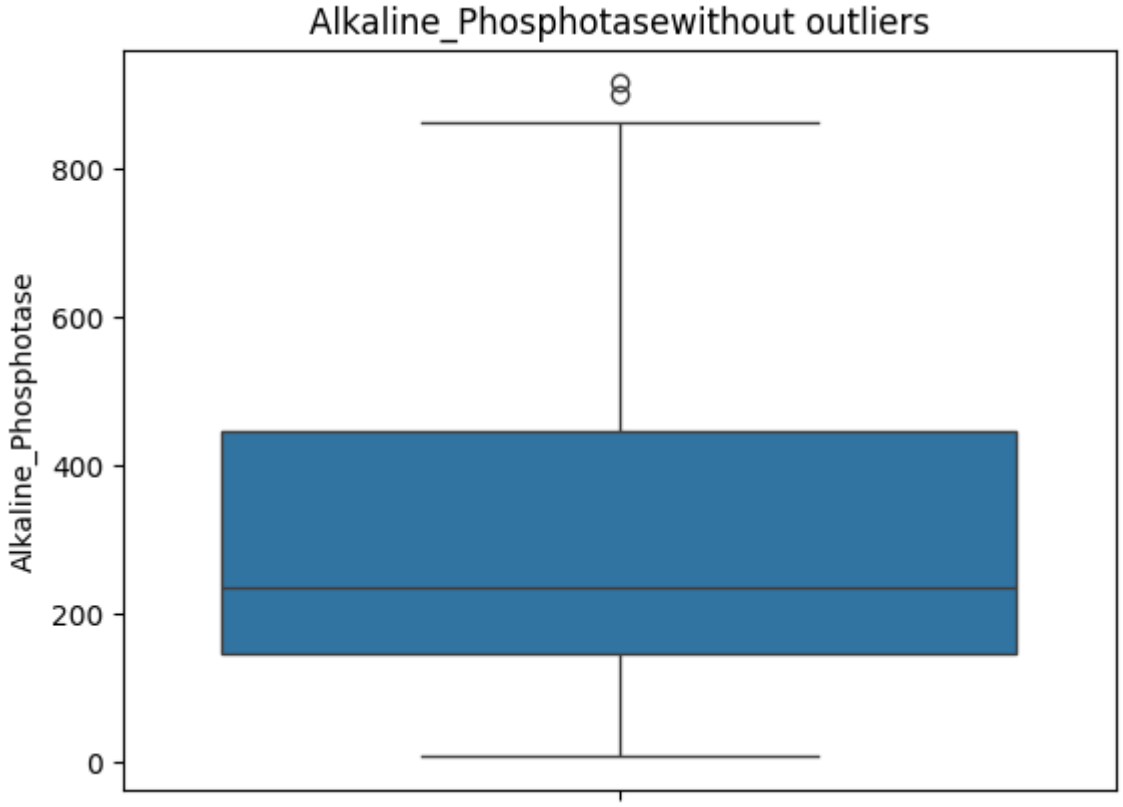
Removing outlier using IQR

```
# 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')
```

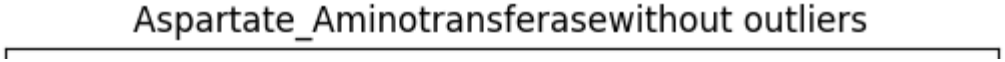
```
# df=df.copy()
```

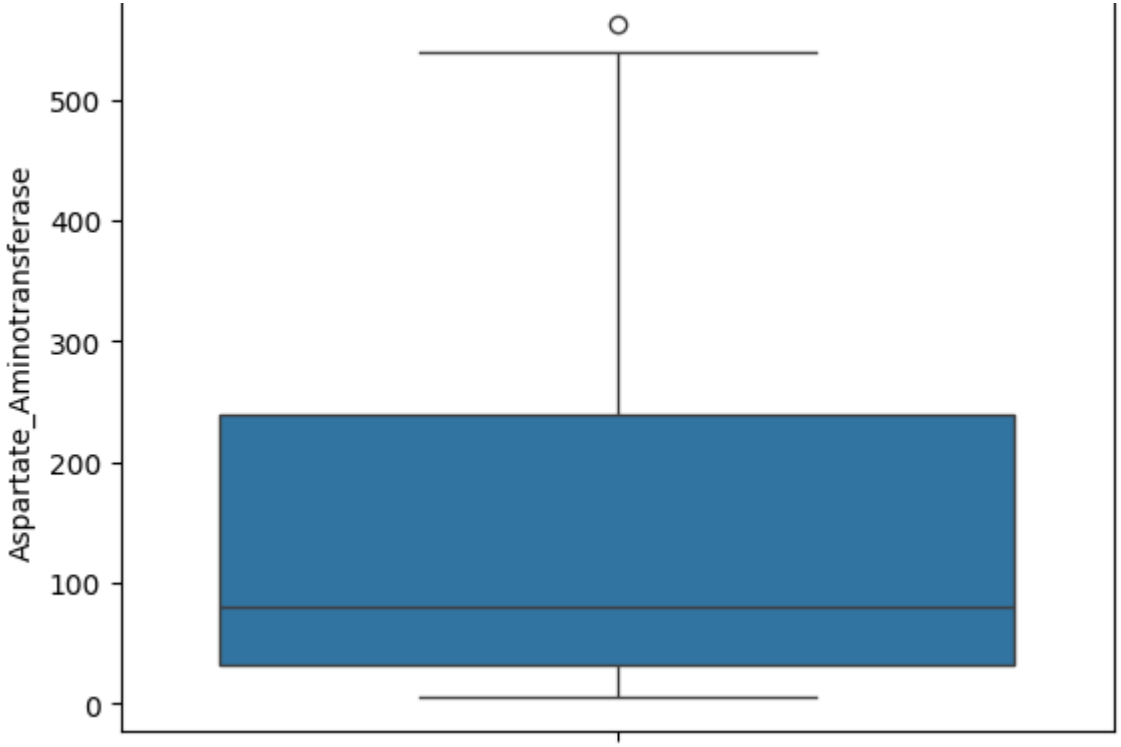
Data without outliers

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




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





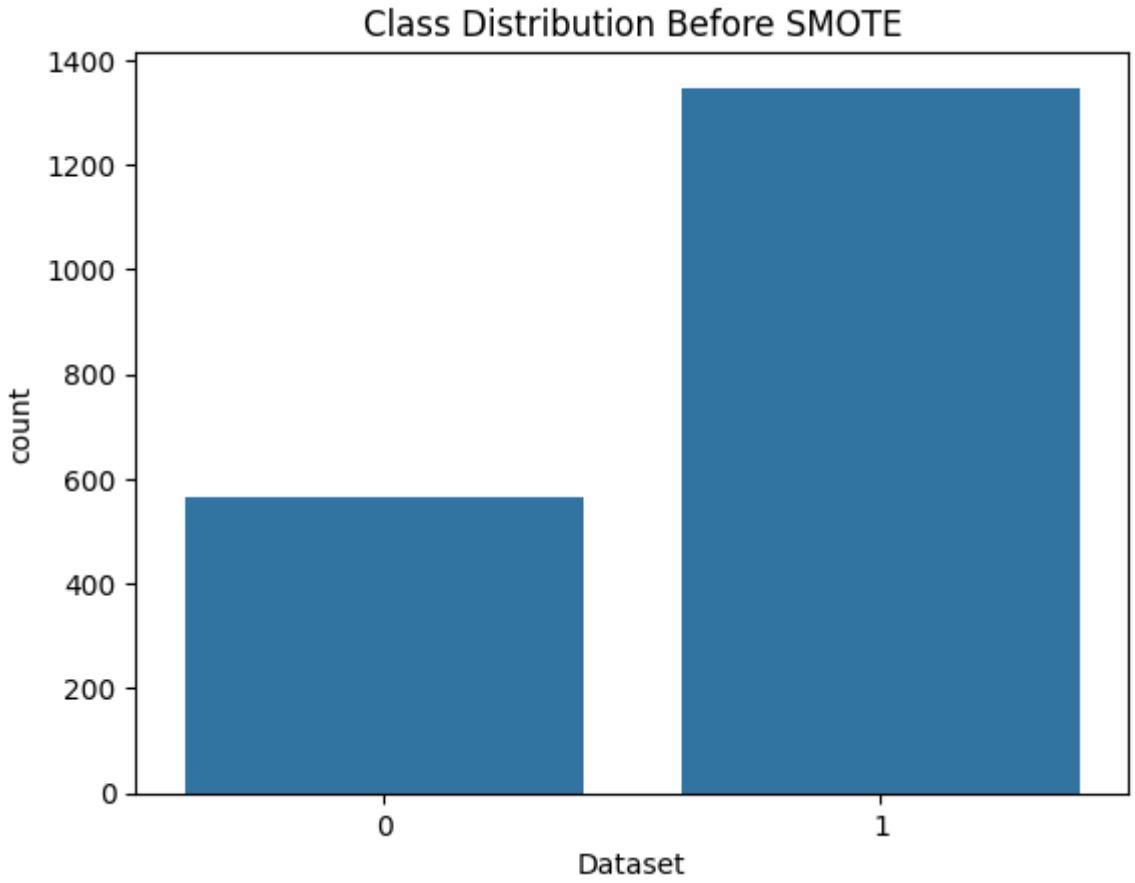
Spliting dataset into training and testing data using train test split

```
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()
```

Dataset	count
1	1347
0	565

dtype: int64

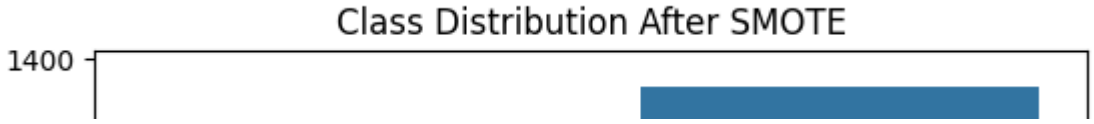
```
imbalace_ratio = y_train.value_counts()[1] / y_train.value_counts()[0]
imbalace_ratio
```

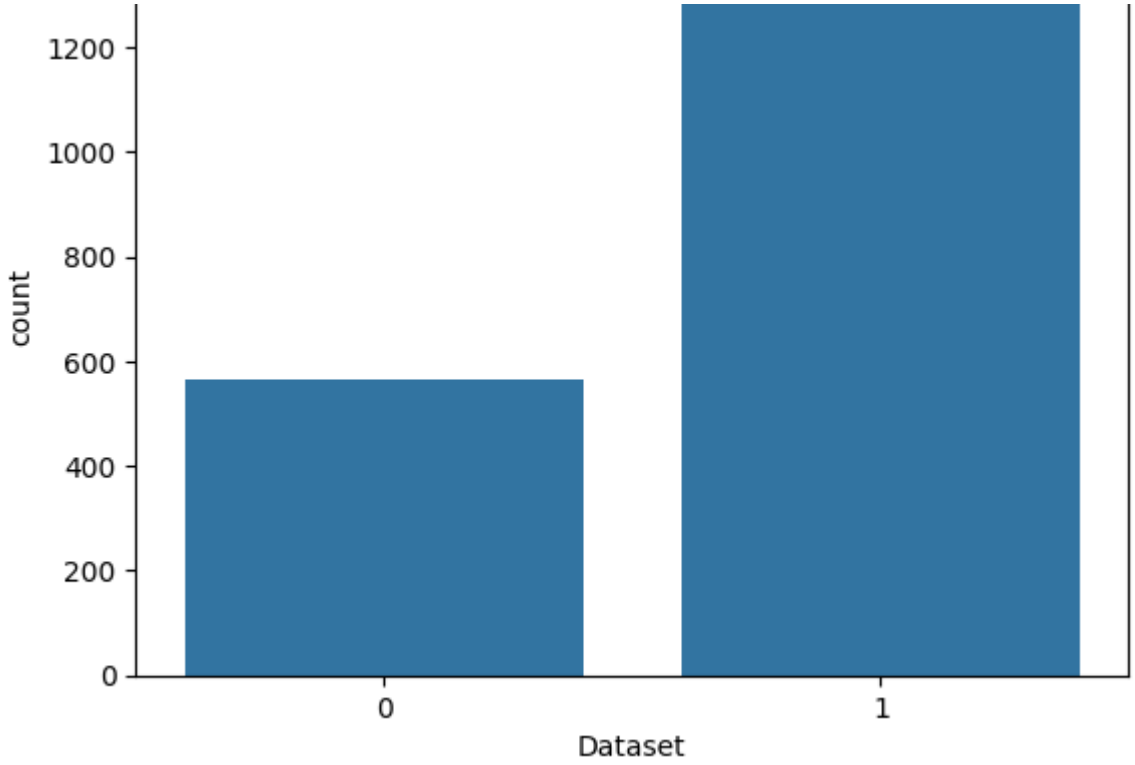
```
np.float64(2.384070796460177)
```

```
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

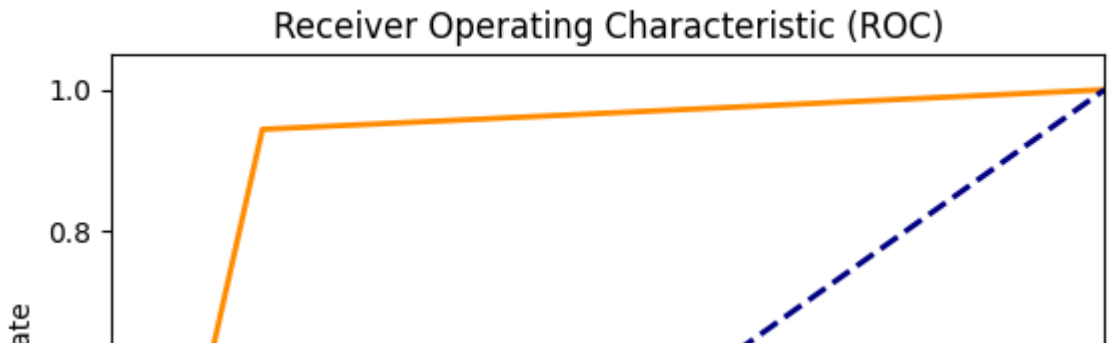
KNN

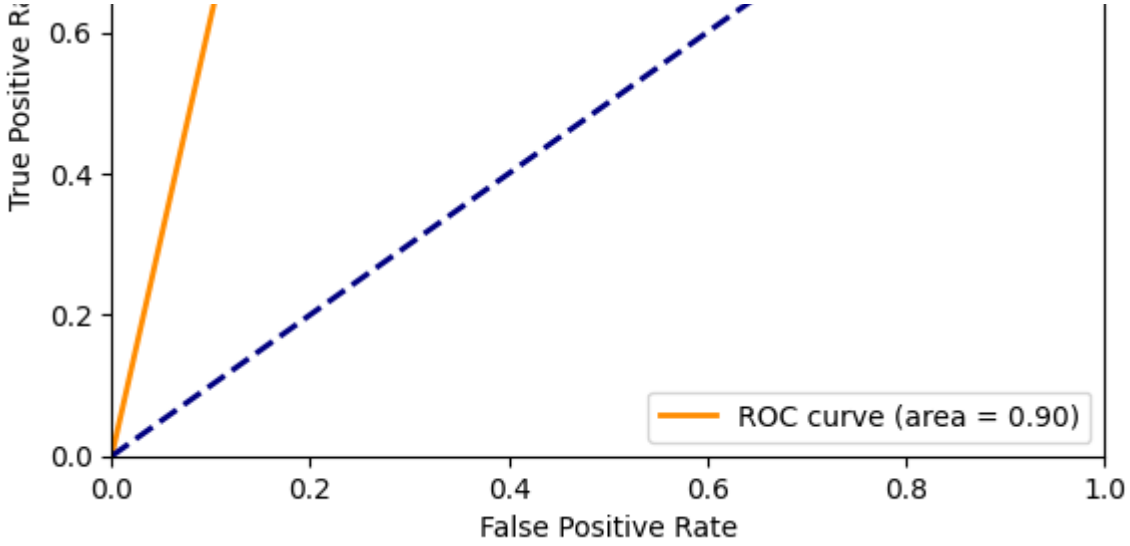
KNN

```
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')
```

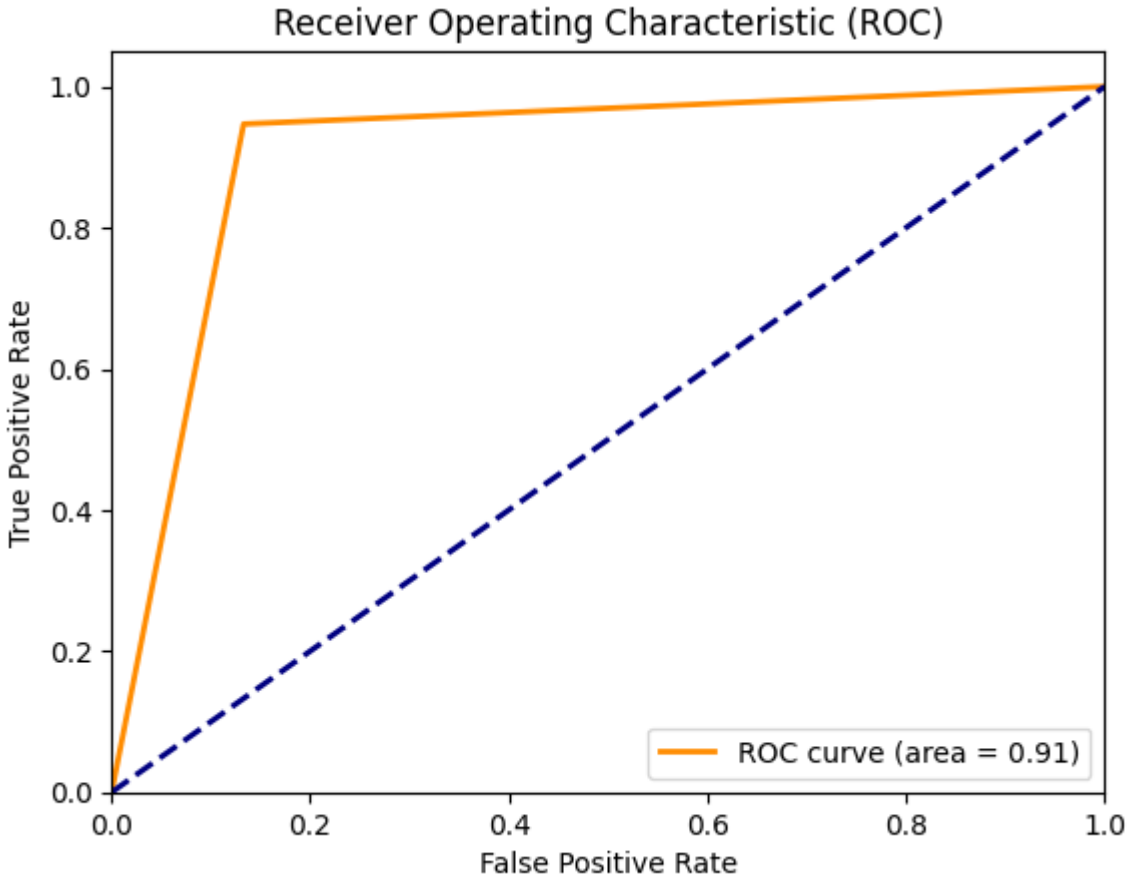
Result

```
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: 0.9441152657853935
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)
```

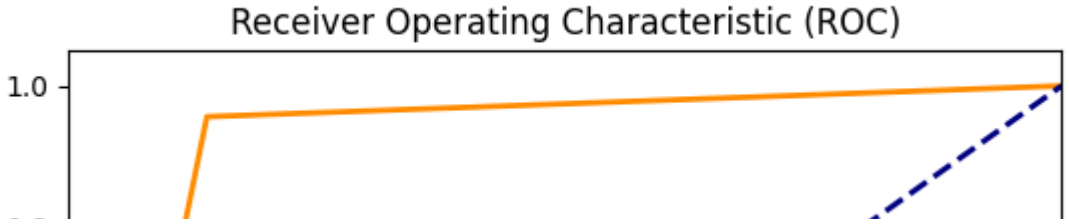
```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

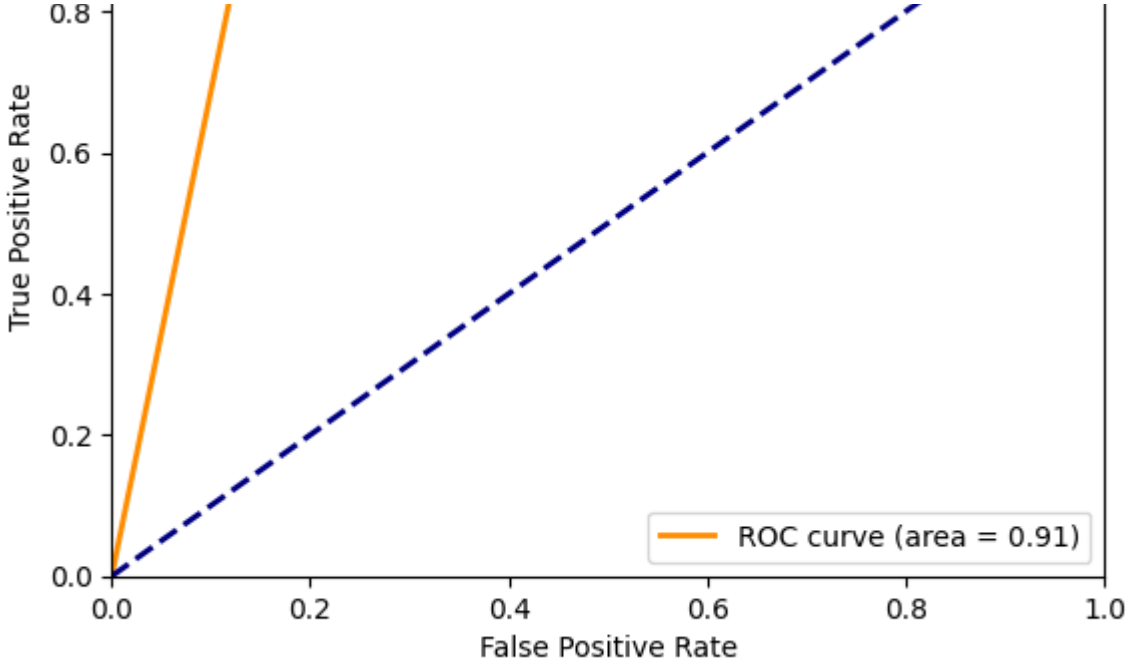
Result

```
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

[ ] ↳ 8 cells hidden

Added l2 regularization

[ ] ↳ 3 cells hidden

L1,L2 regularization

[ ] ↳ 5 cells hidden

Added dropout layer

[ ] ↳ 5 cells hidden

Using CNN

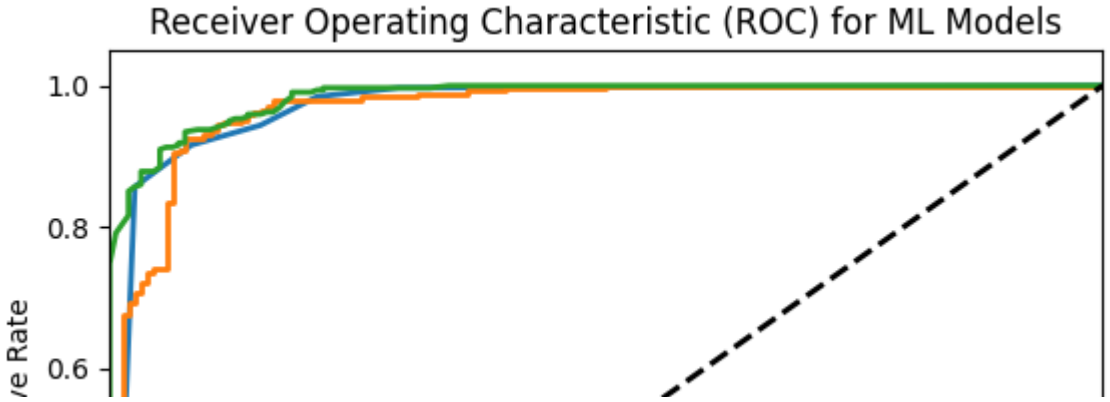
[ ] ↳ 5 cells hidden

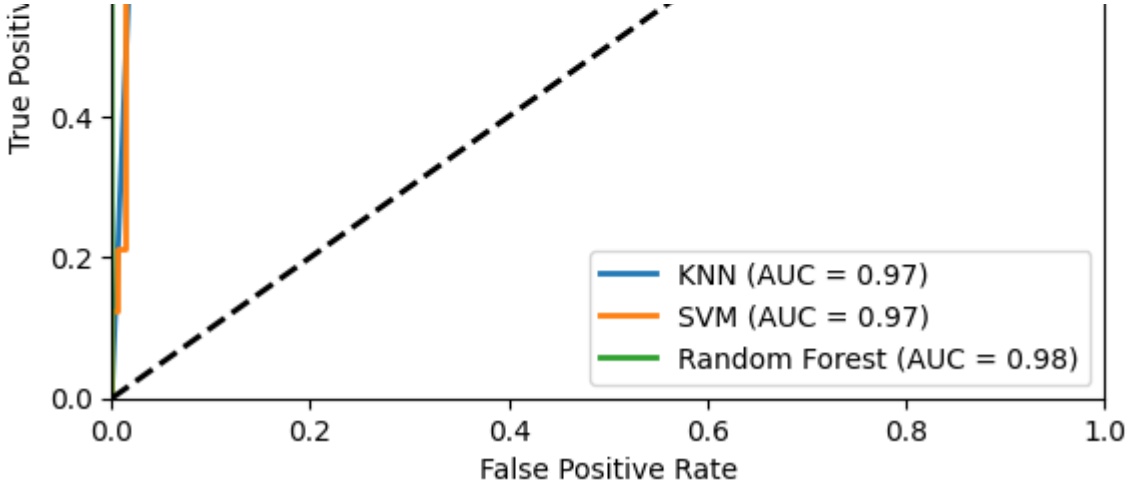
Using multiple convolutional layer and pooling *layer*

[ ] ↳ 3 cells hidden

Plot ROC curve for all the models

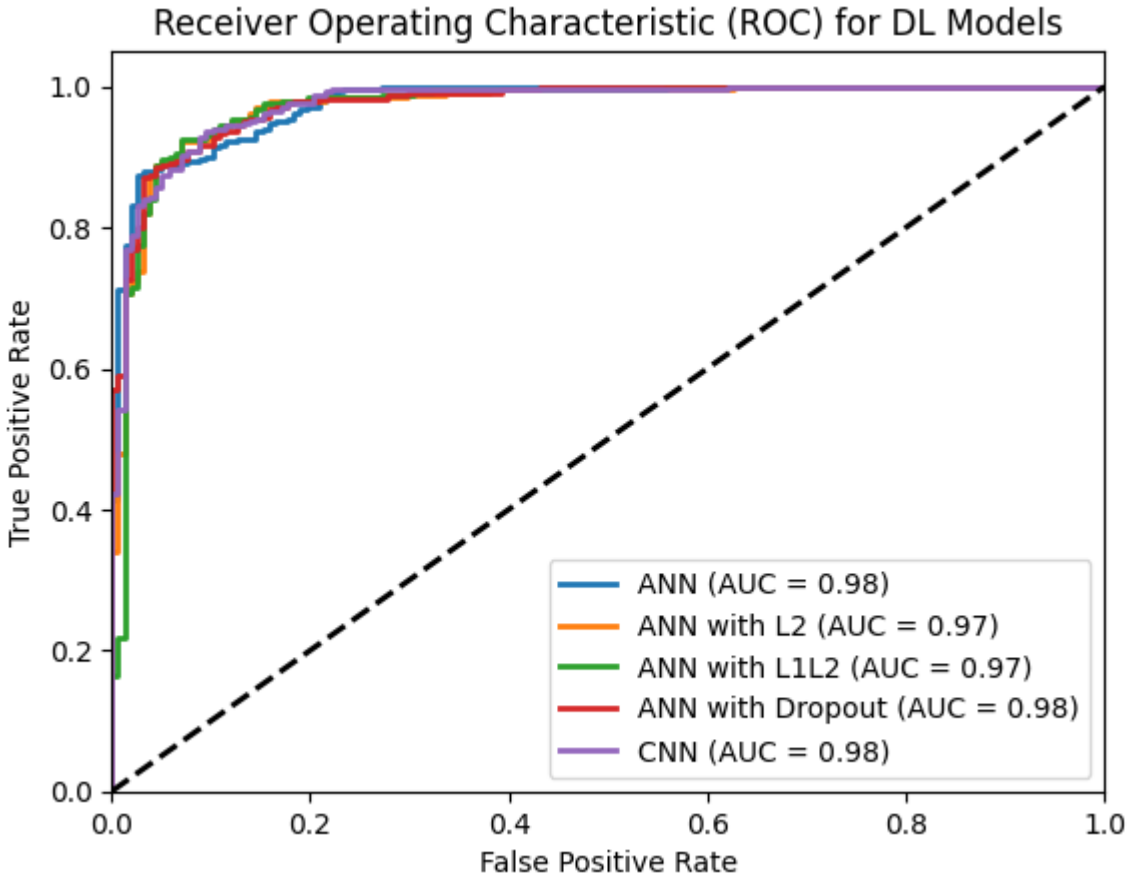
```
# plot ROC in single figure
from sklearn.metrics import roc_curve, auc
def plot_roc_curve(y_test, y_pred_prob, model_name):
    fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, lw=2, label=f'{model_name} (AUC = {roc_auc:.2f})')


# 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()
```





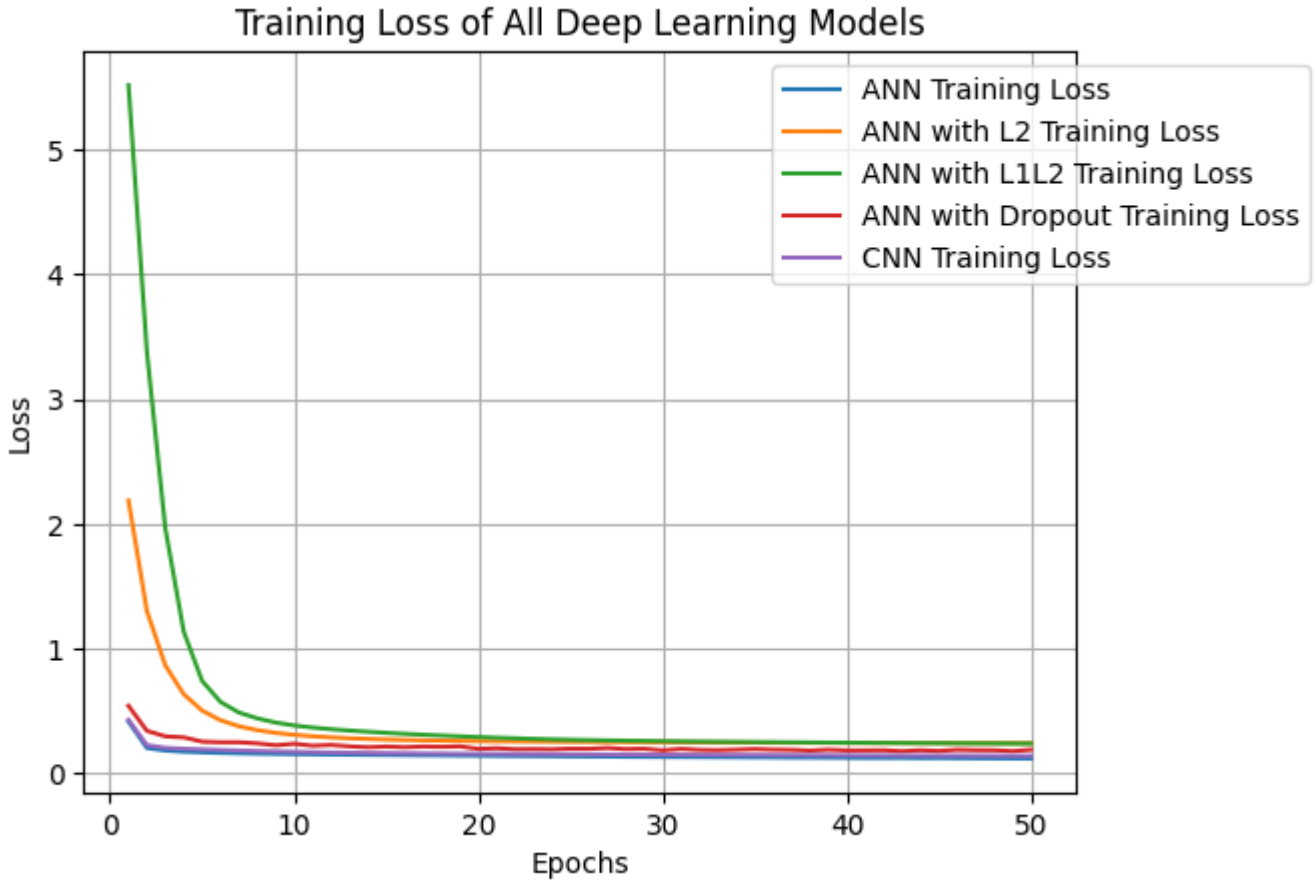

```
# DL
ann_y_pred_prob = ann_model.predict(X_test_std).ravel()
l2_y_pred_prob = l2_model.predict(X_test_std).ravel()
l1l2_y_pred_prob = l1l2_model.predict(X_test_std).ravel()
dropout_y_pred_prob = dropout_model.predict(X_test_std).ravel()
cnn_y_pred_prob = cnnModel.predict(X_test_cnn).ravel()

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')

# 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
```

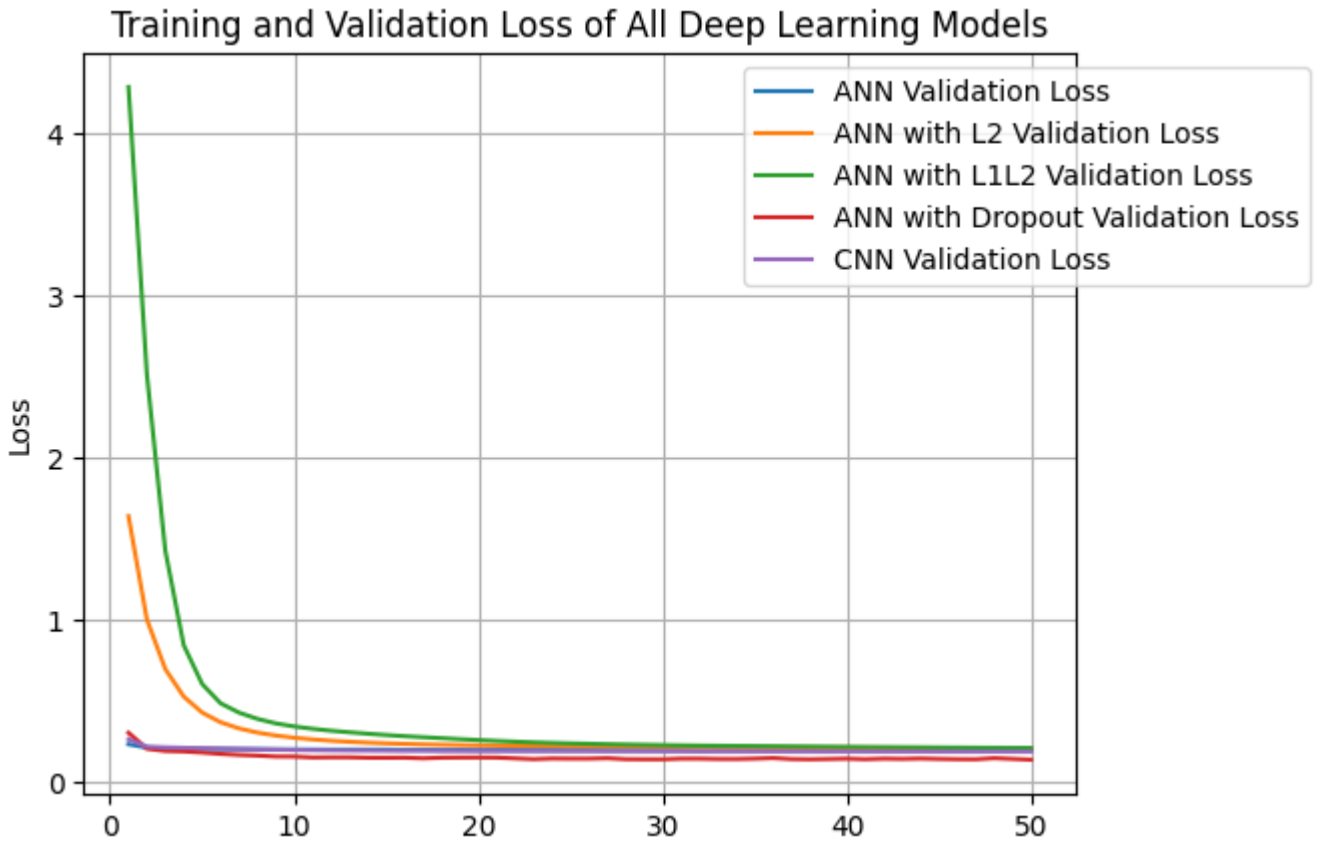



```
epochs = range(1, EPOCH + 1)

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, 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()
```

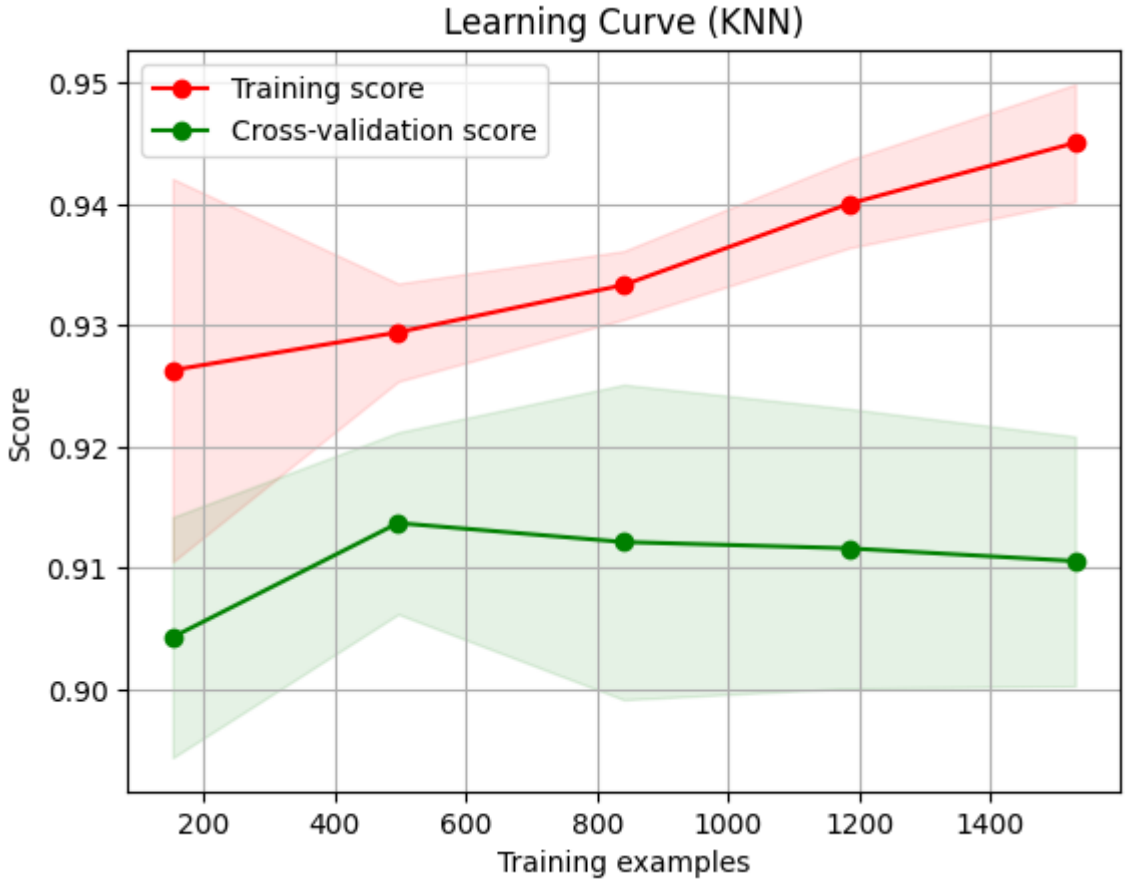


Epochs

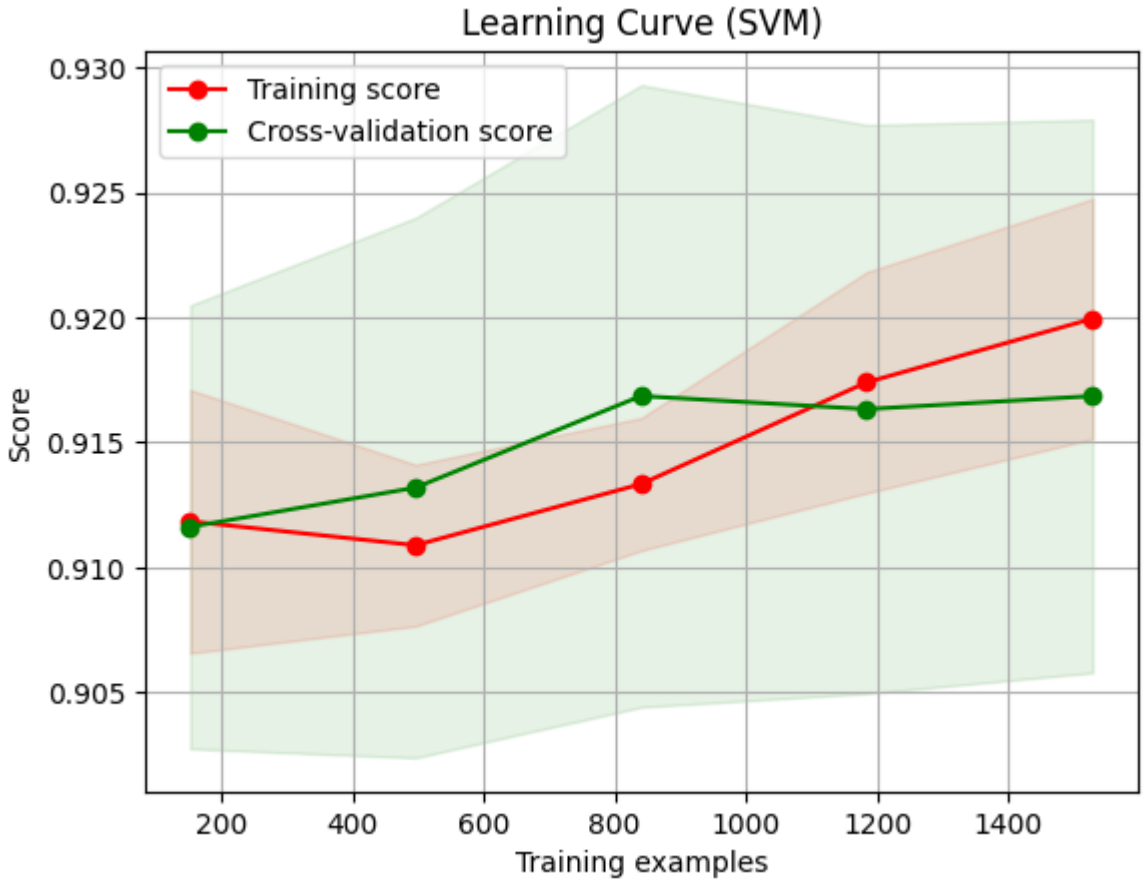
Plot Leraning Curve fot ML models

```
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"
```

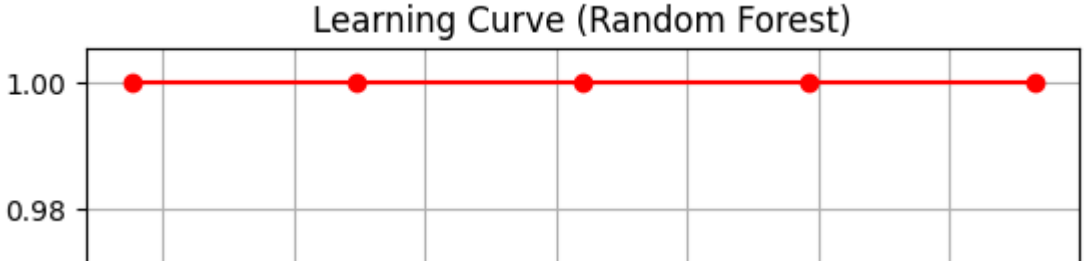
```
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

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


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

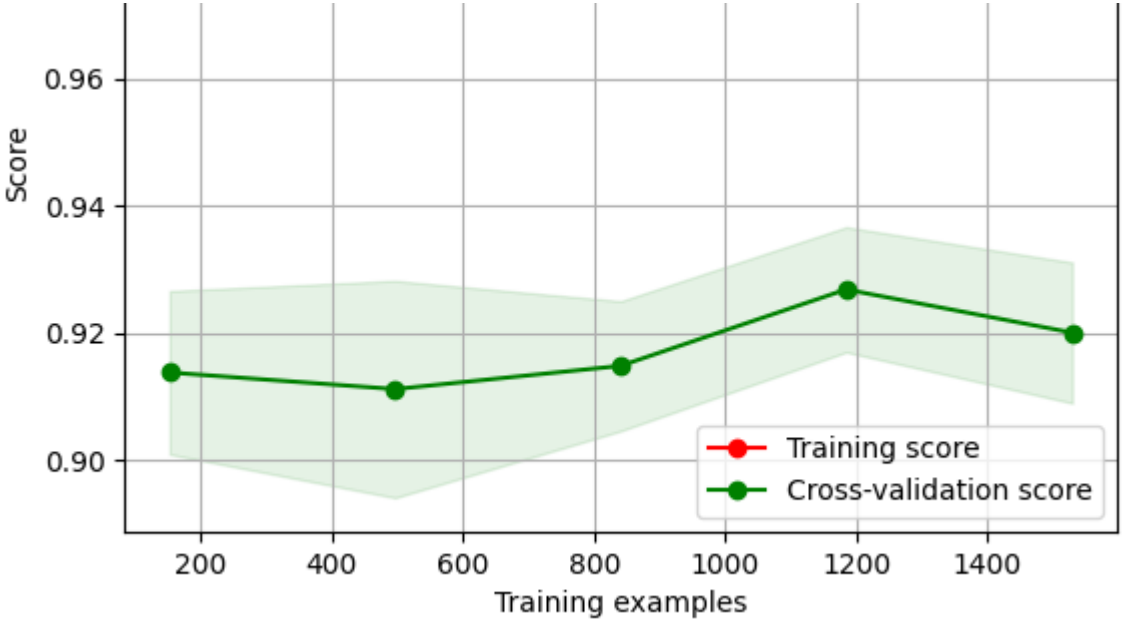



```
plot_learning_curve(svm, "Learning Curve (SVM)", X_train_std, y_train, cv=5)
plt.show()
```




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





Start coding or generate with AI.