

**CLASSIFIERS**

**Discriminant Analysis**

**Laboratory Report**

**Author:**  **Ankit Rathi**

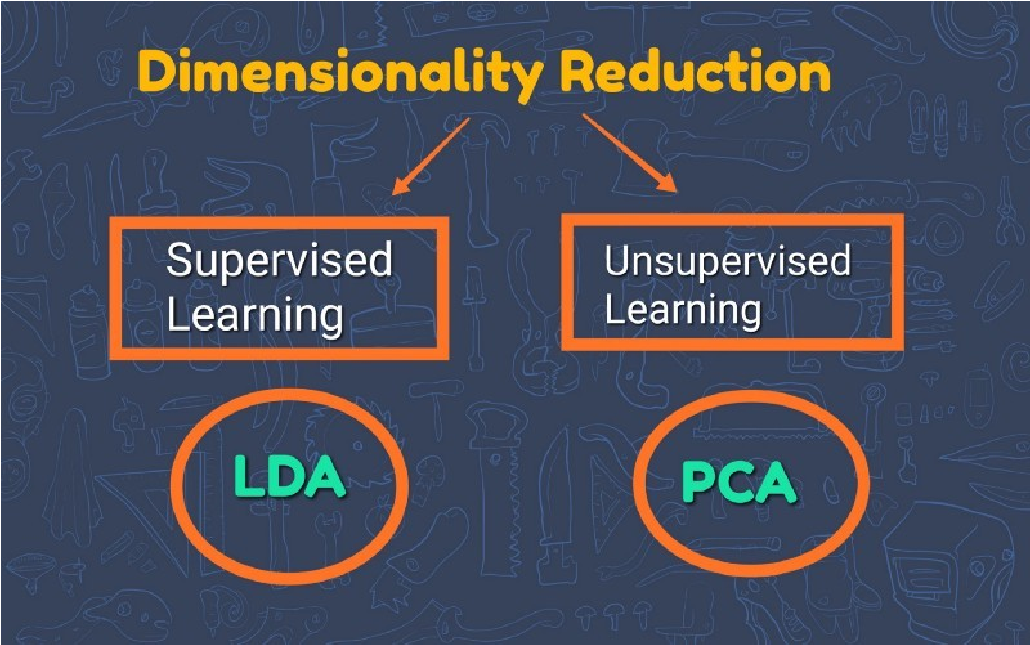
# Short Introduction

Linear Discriminant Analysis(LDA) is a very common technique used for supervised classification problems.

**What is Linear Discriminant Analysis ?**

Linear Discriminant Analysis is a dimensionality reduction technique used as a preprocessing step in Machine Learning and pattern classification applications.

The main goal of dimensionality reduction techniques is to reduce the dimensions by removing the redundant and dependent features by transforming the features from higher dimensional space to a space with lower dimensions.



Linear Discriminant Analysis is a supervised classification technique which takes labels into consideration

**Tasks**

# Analysis of Artificial Data

**1. Perform basic visualization of both datasets (distribution, histogram, boxplot, scatterplot). Determine which is linearly separable and which is not.**

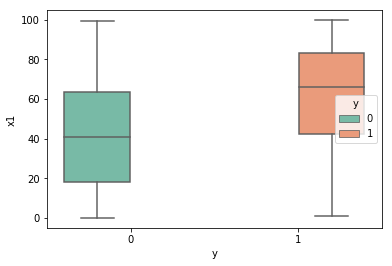


Figure 1. x1 dataset 1

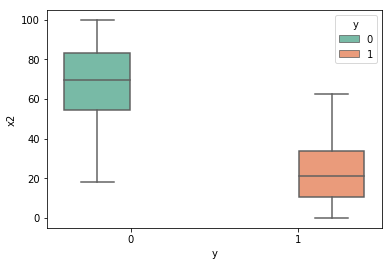


Figure 2. x2 dataset 1

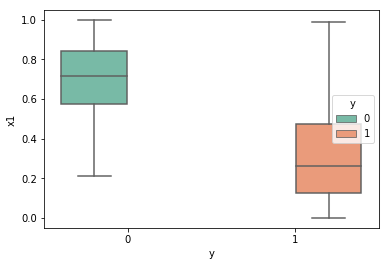


Figure 3. x1 dataset 2

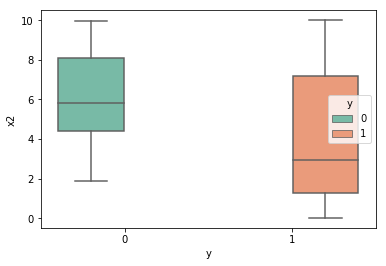


Figure 4. x2 dataset 2

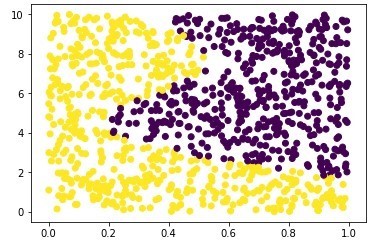


Figure 5. Scatter Plot dataset 2

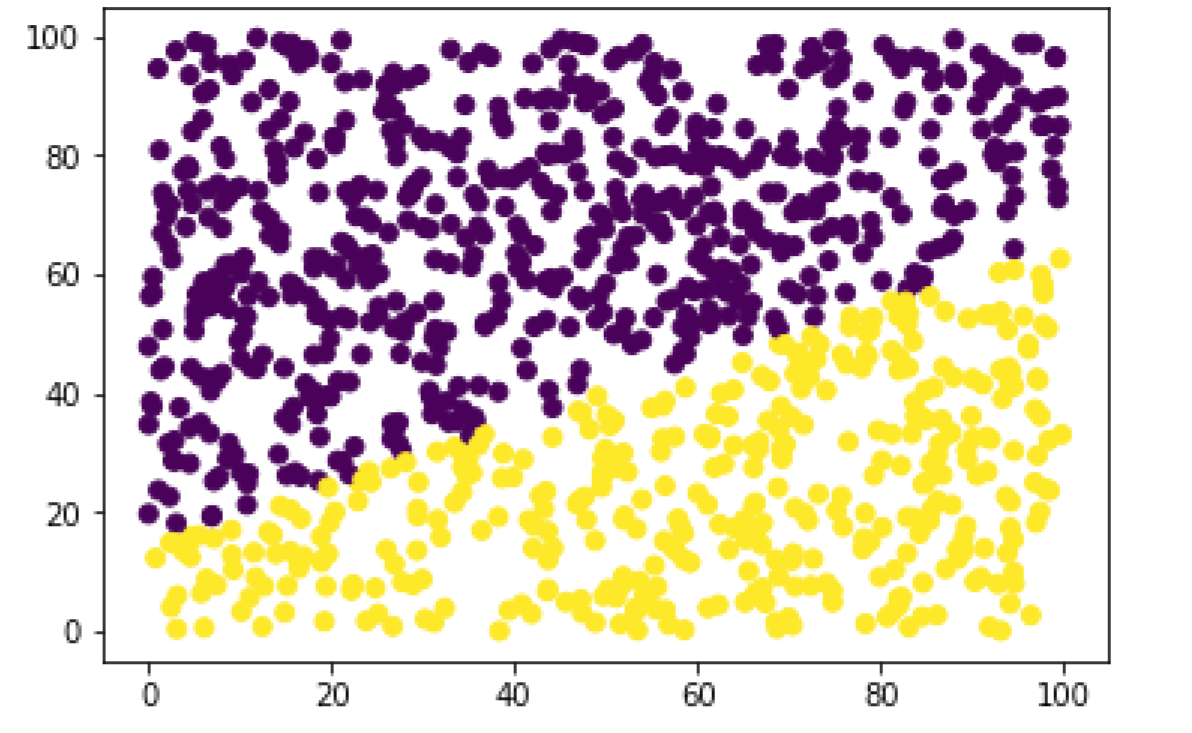


Figure 6. Scatter Plot dataset 1

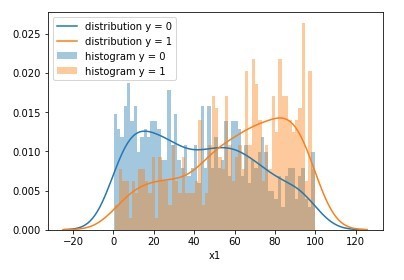


Figure 7. x1 dataset 1

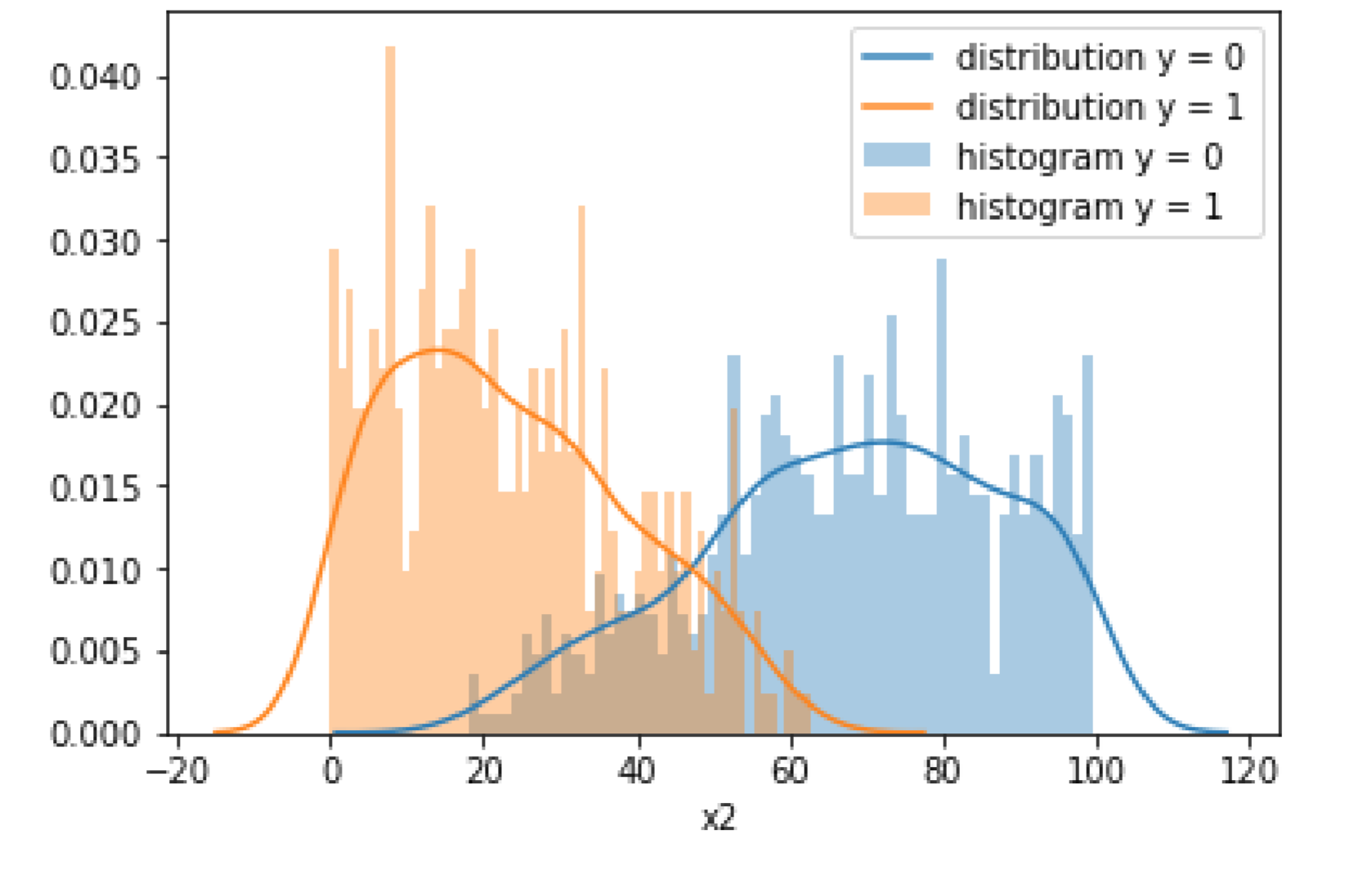


Figure 8. x2 dataset 1

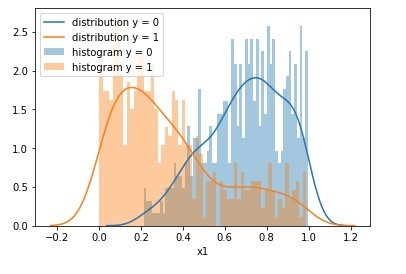


Figure 9. x1 dataset 2

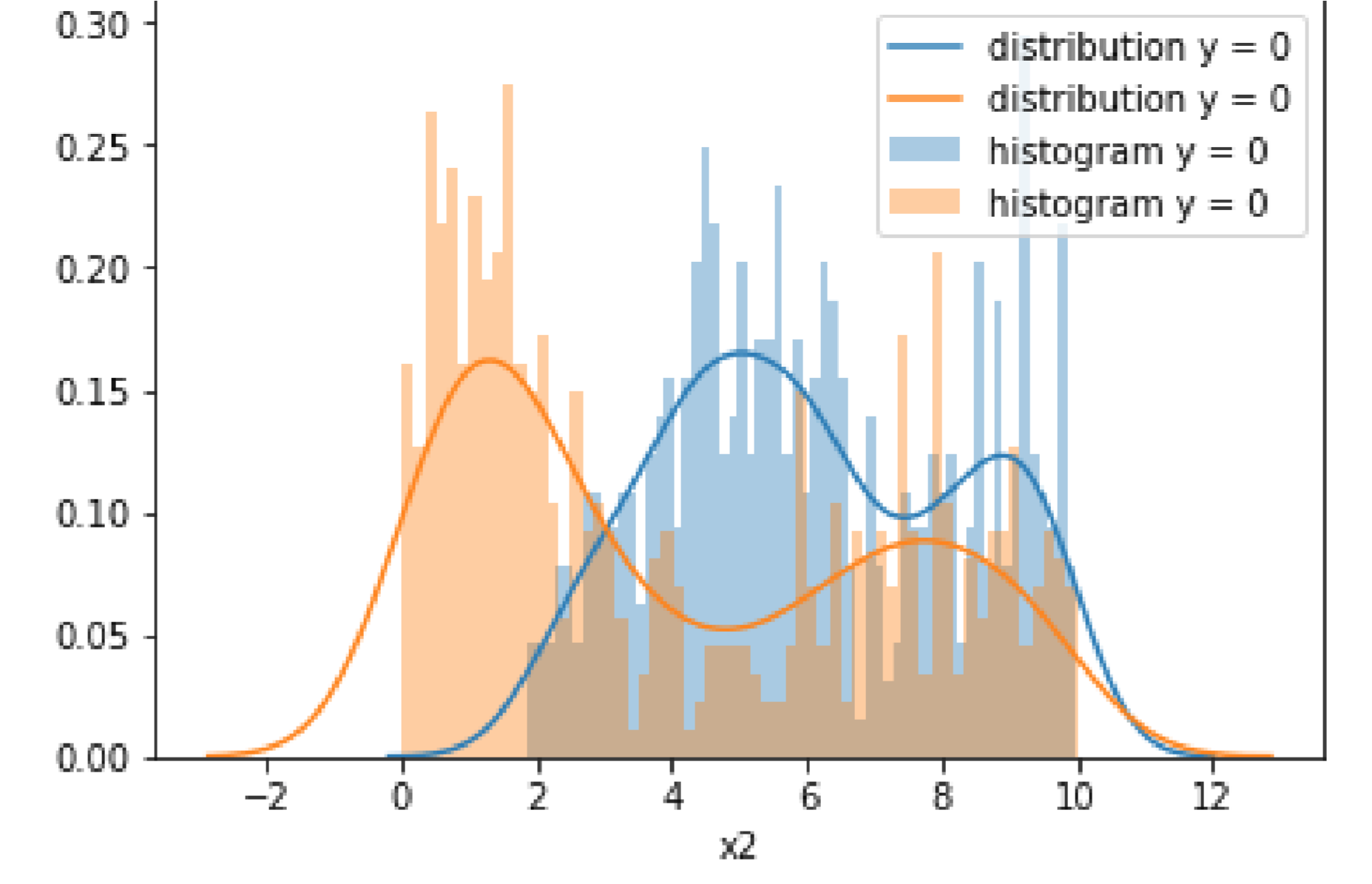


Figure 10. x2 dataset 2

**From the view of the figures above, Histogram and Scatterplot are linearly separable While Distribution and Boxplot are not.**

1. **Split the dataset into a training and a test set.**

Refer to the code

1. **Test the operations of classifiers. The classification results on the test set should be presented in the contingency table .Specify values of the following quality indicators: sensitivity, specificity, accuracy, error.**

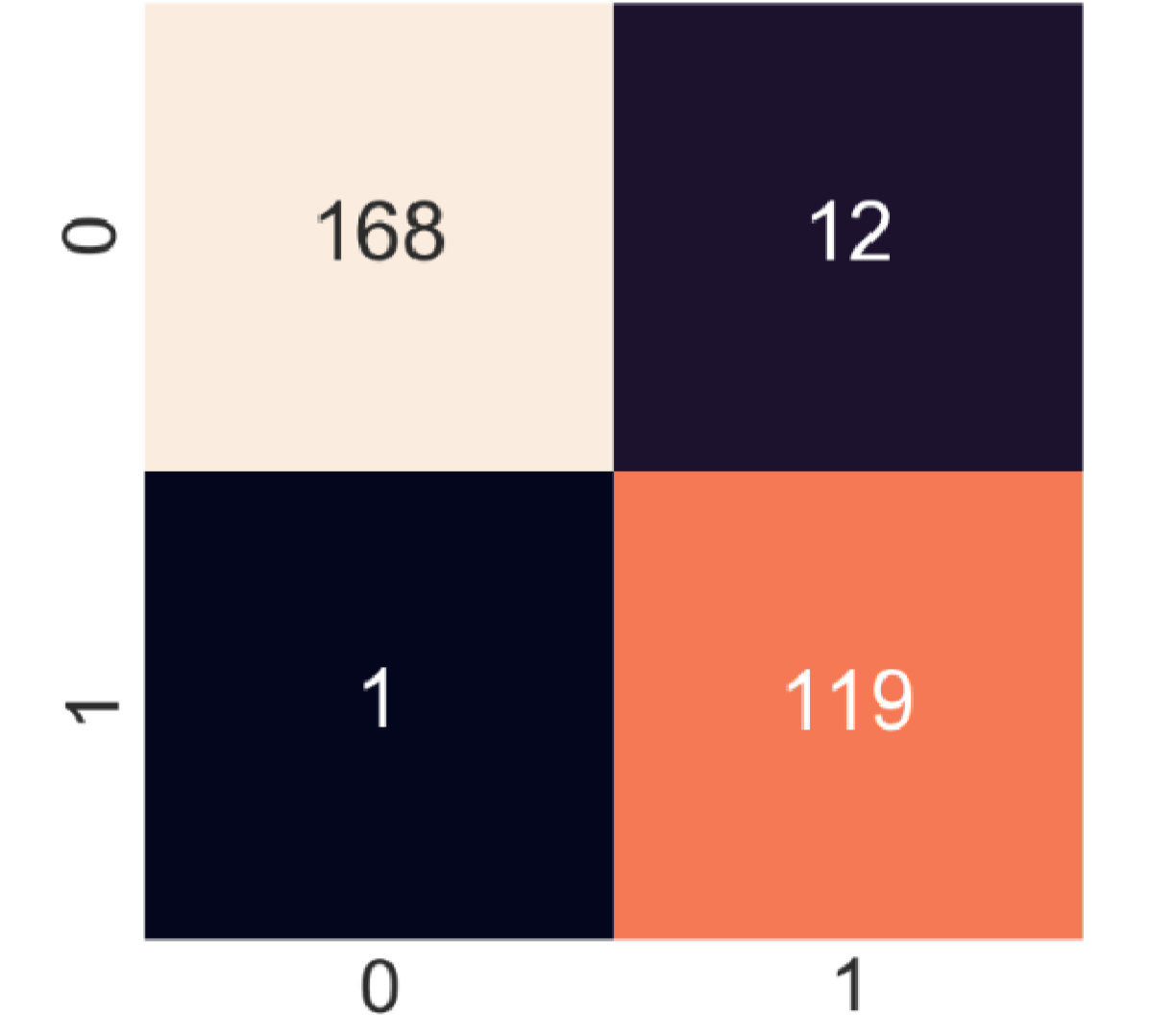


Figure 11. LDA on dataset 1 Contingency Table

**Sensitivity** 0.9940828402366864

**Specificity** 0.9083969465648855

**Accuracy** 0.9566666666666667

**Error** 0.038461538461538464

Figure 12. LDA Quality Metrics

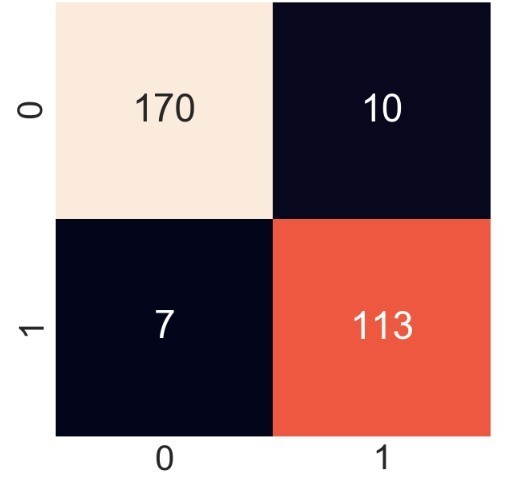


Figure 12. QDA on dataset 1 Contingency Table

**Sensitivity** 0.96045197740113

**Specificity** 0.9186991869918699

**Accuracy** 0.9433333333333334

**Error** 0.0480225988700565

Figure 13. QDA Quality Metrics

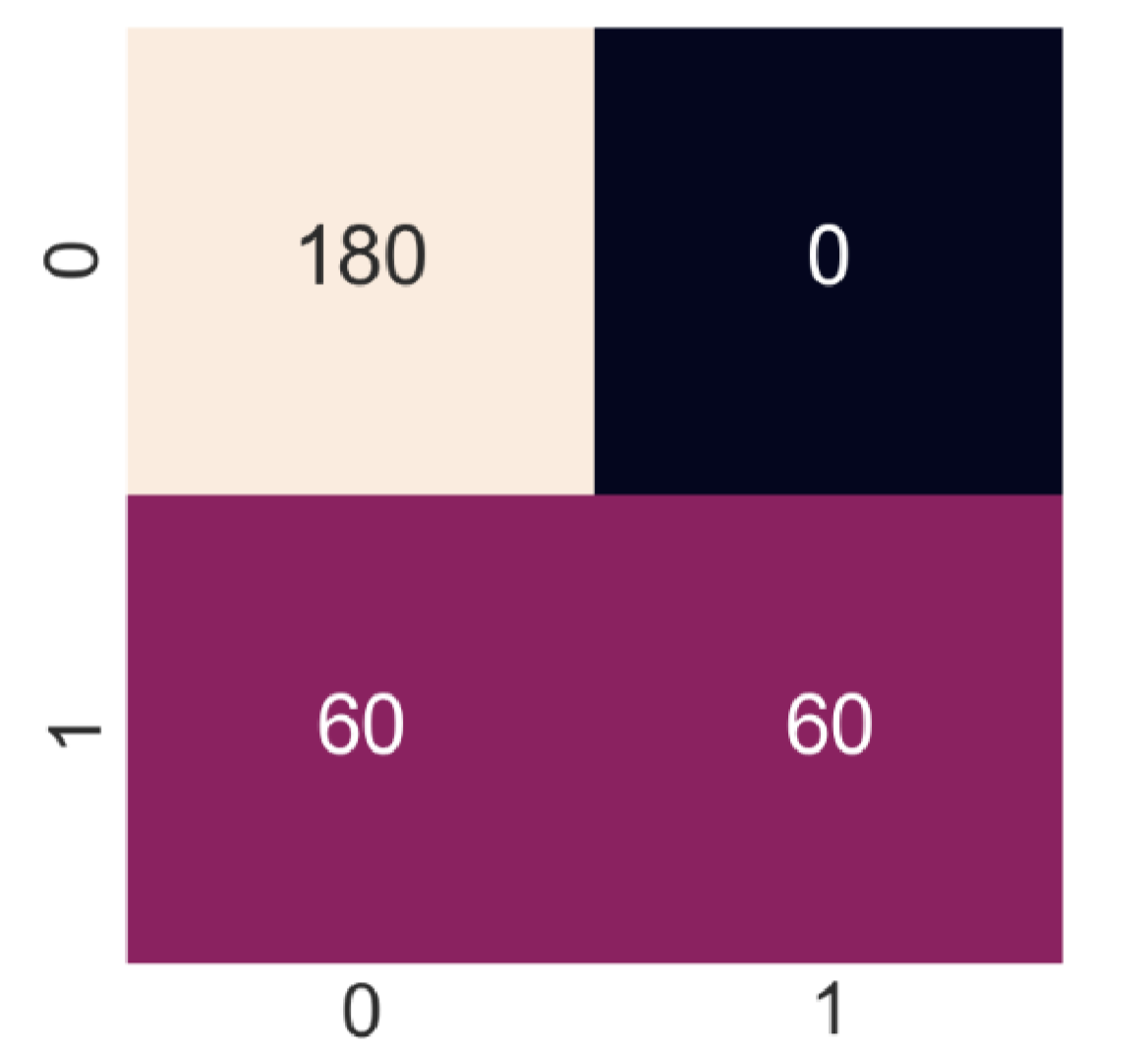


Figure 14. SVM on dataset 1 Contingency Table

**Sensitivity**

0.75

**Specificity**

1.0

**Accuracy**

0.8

**Error**

0.125

Figure 15.SVM Quality Metrics

**4. Plot the ROC curves and compare the area under the curve for both sets.**

**Dataset 1**

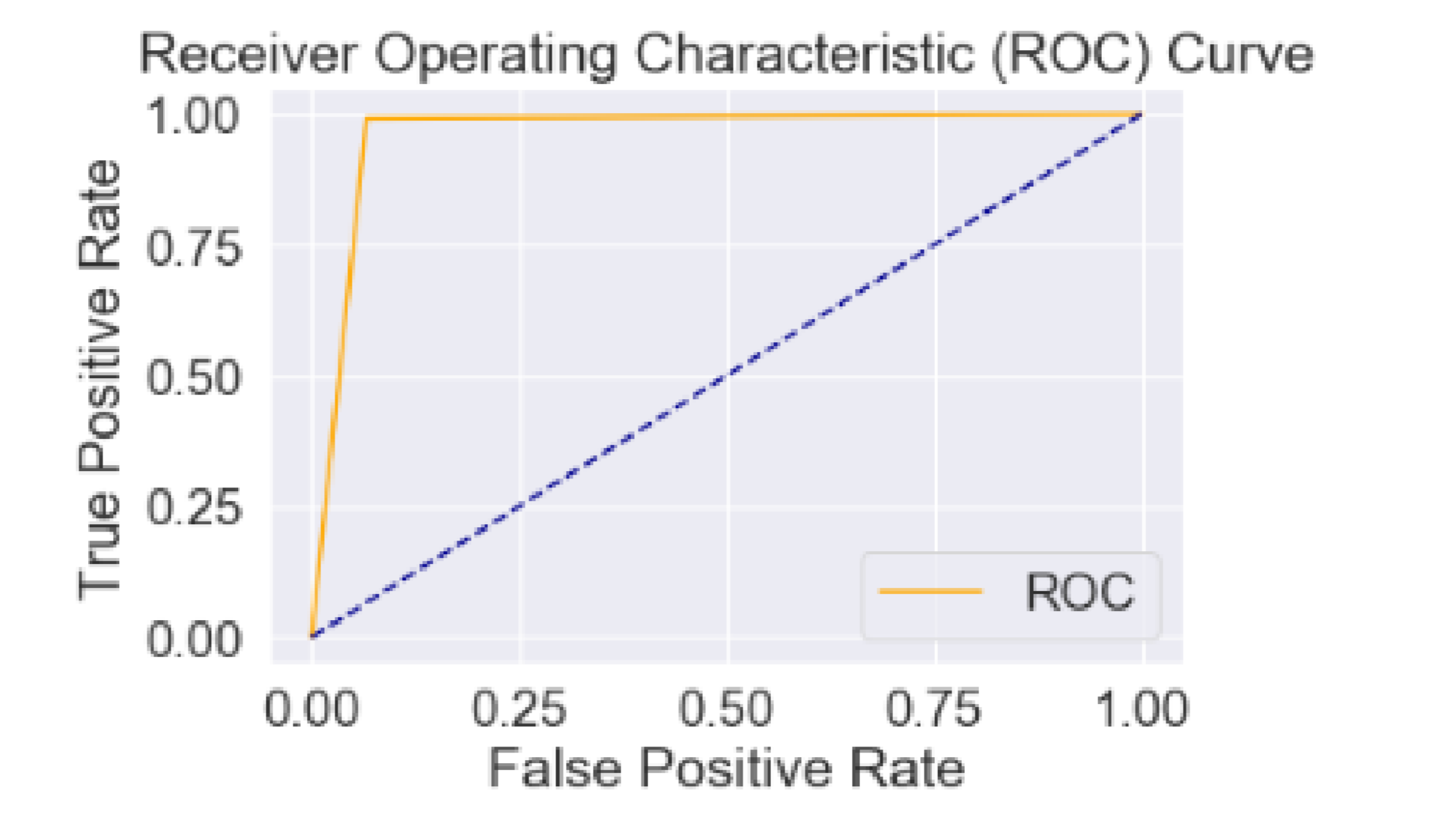


Figure 16. ROC LDA

The Area under the curve (AUC) = 0.9625

* **LDA AUC** = 0.9625 .This can be interpreted as there are slightly over 96% chances that this particular model will distinguish negative and positive classes. As the value of this indicator increase, we have a higher probability to have a better classifier.

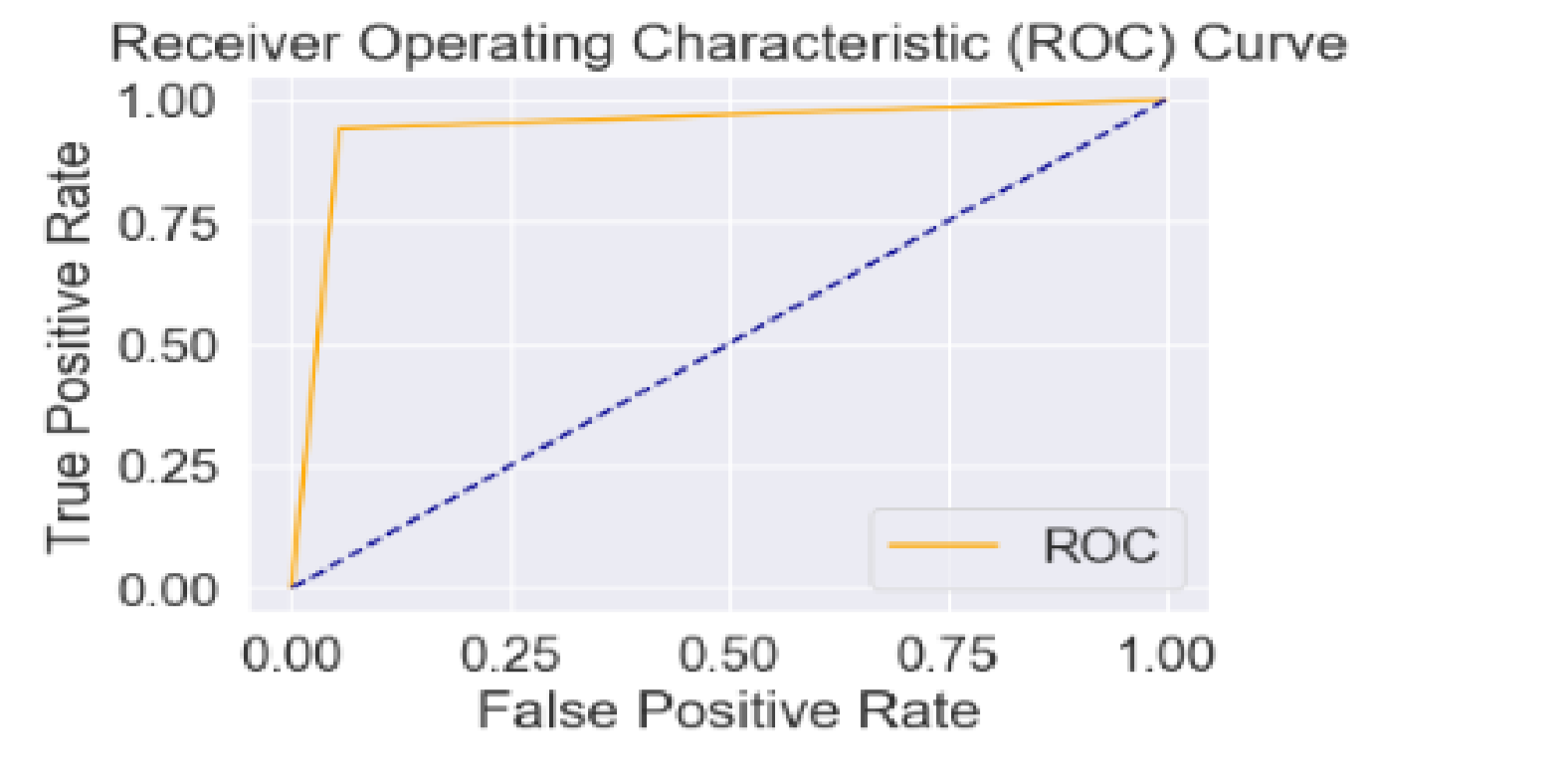


Figure 17 ROC QDA

The Area under the curve (AUC) = 0.9431

* **QDA AUC** = 0.9431. This can be interpreted as there are slightly over 94% chances that this particular model will distinguish negative and positive classes. As the value of this indicator increase, we have a higher probability to have a better classifier.

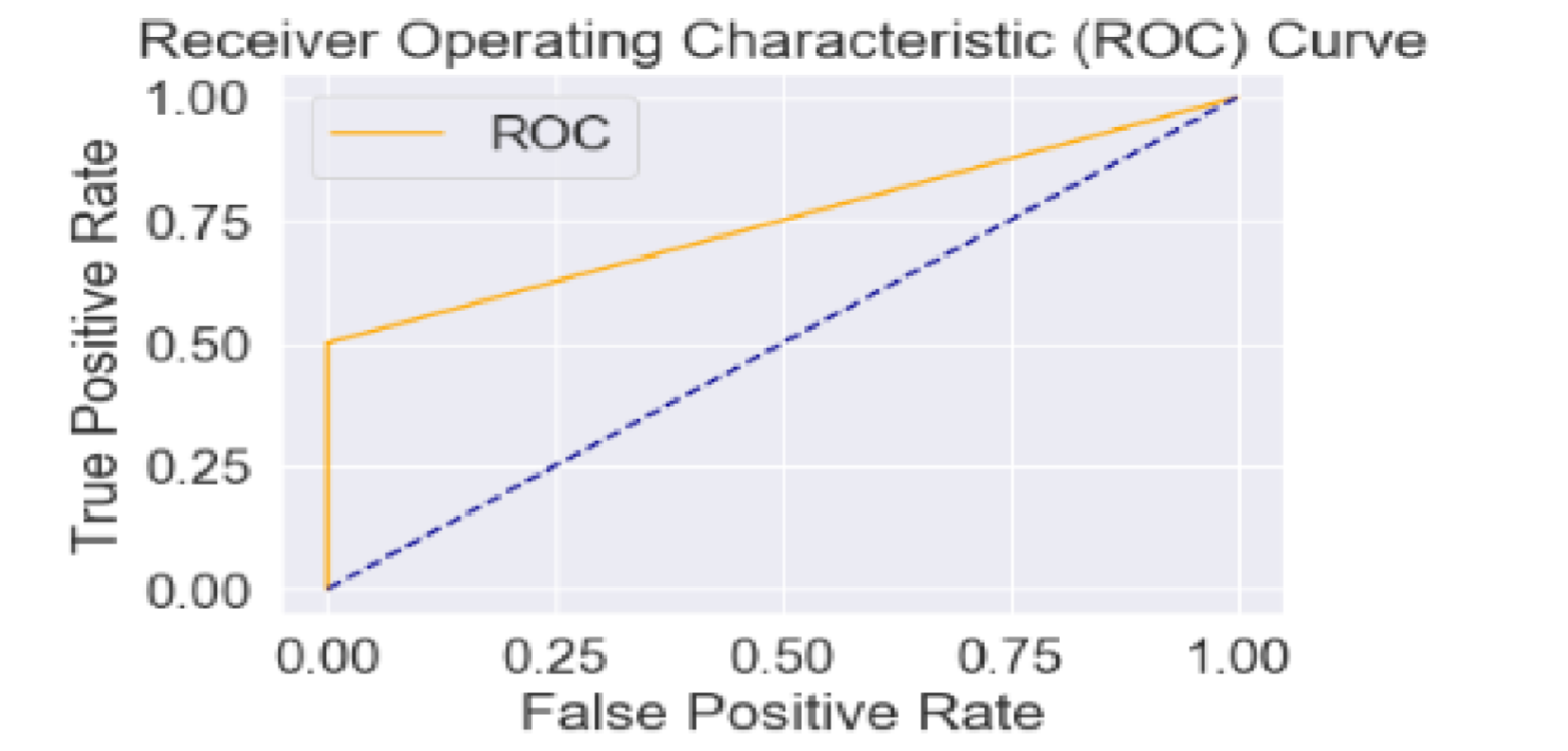


Figure 18 ROC SVM

Area under the curve(AUC) = 0.7500

* **SVM AUC** = 0.7500. This can be interpreted as there are 75% chances that this particular model will distinguish negative and positive classes. As the value of this indicator increase, we have a higher probability to have a better classifier.

**Dataset 2**



Figure 19. ROC LDA

The Area under the curve (AUC) = 0.8132

* **LDA AUC** = 0.8132. This can be interpreted as there is a 81% chance that this particular model will distinguish negative and positive classes. As the value of this indicator increase, we have a higher probability to have a better classifier.



Figure 20. ROC QDA

The Area under the curve(AUC) = 0.8947

* **QDA AUC** = 0.8947. This can be interpreted as there is a slightly over 89% chance that this particular model will distinguish negative and positive classes. As the value of this indicator increase, we have a higher probability to have a better classifier.

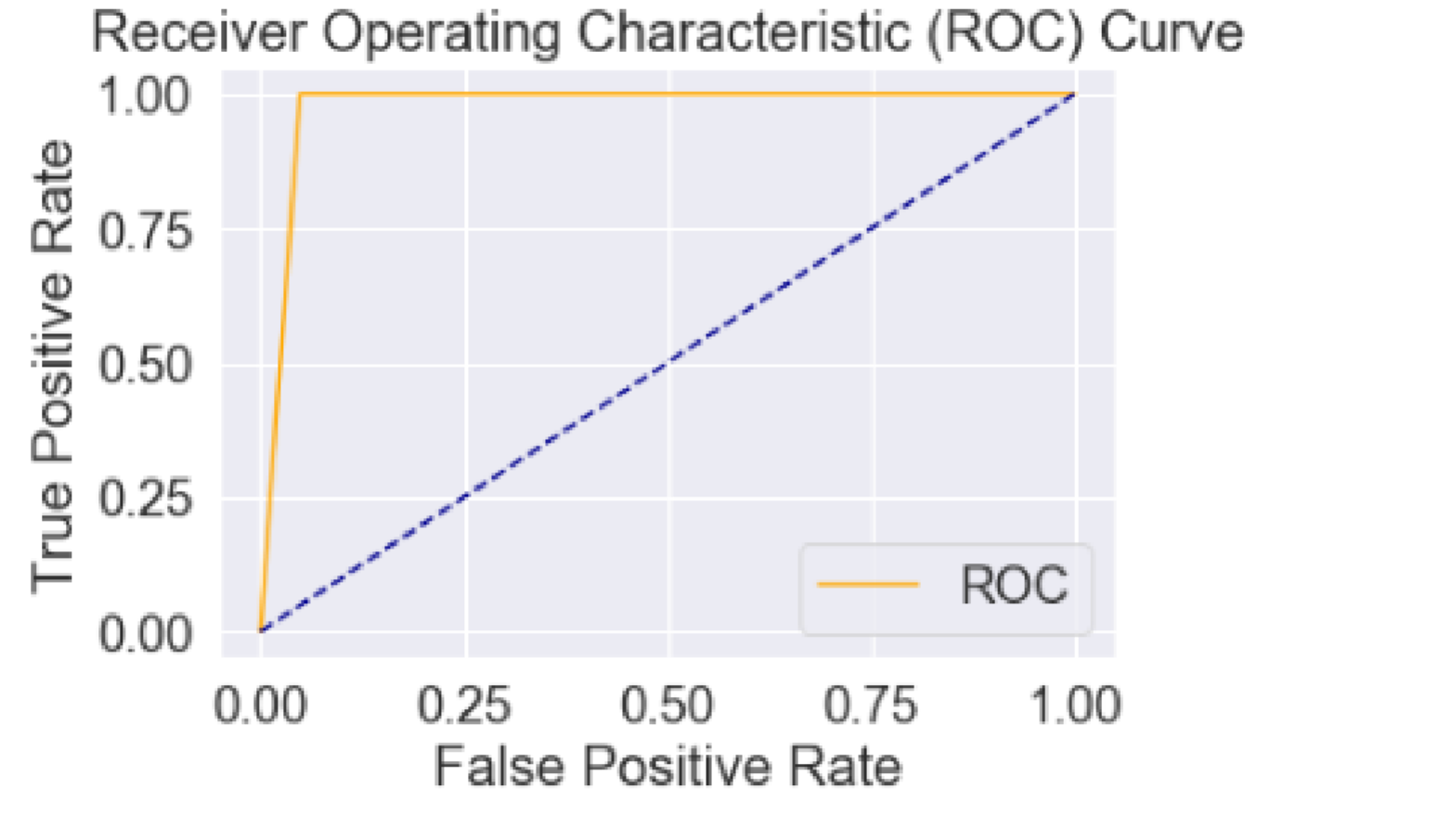


Figure 21 ROC SVM

The Area under the curve (AUC) = 0.9755

* **SVM AUC** = 0.9755. This can be interpreted as there is a slightly over 97% chance that this particular model will distinguish negative and positive classes. As the value of this indicator increase, we have a higher probability to have a better classifier.

**5. Compare (and describe) the operation of classifiers (LDA, QDA, SVM)**

**LDA and QDA**

Linear discriminant Analysis and Quadratic discriminate Analysis are popular traditional classification methods. These two methods assume each class are from multivariate Gaussian distribution and use statistical properties of the data, the variance - covariance and the mean, to establish the classifier. The mainly difference between LDA and QDA is that if we have observed or calculated that each class has similar variance - covariance matrix, we will use LDA to construct a straight line as our classifier; otherwise, if classes have different variance - covariance matrix, we will use QDA to construct a quadratic curve as our classifier.

Linear Discriminant Analysis is a simple and effective method for classification. There are many extensions and variations to the method. Some popular extensions include: **QDA, FDA** and **RDA**

**Quadratic Discriminant Analysis (QDA)**: Each class uses its own estimate of variance (or covariance when there are multiple input variables).

**SVM** classification is an optimization problem, LDA has an analytical solution. The optimization problem for the SVM has a dual and a primal formulation that allows the user to optimize over either the number of data points or the number of variables, depending on which method is the most computationally feasible. SVM can also make use of kernels to transform the SVM classifier from a linear classifier into a non-linear classifier.

LDA makes use of the ***entire*** data set to estimate covariance matrices and thus is somewhat prone to outliers. SVM is optimized over a subset of the data, which is those data points that lie on the separating margin. The data points used for optimization are called support vectors, because they determine how the SVM discriminate between groups, and thus support the classification.

SVM focuses only on the points that are difficult to classify, LDA focuses on all data points. **LDA is generative, SVM is discriminative.**

# Analysis of actual data

1. **Make a basic selection of features and Data Visualization.**

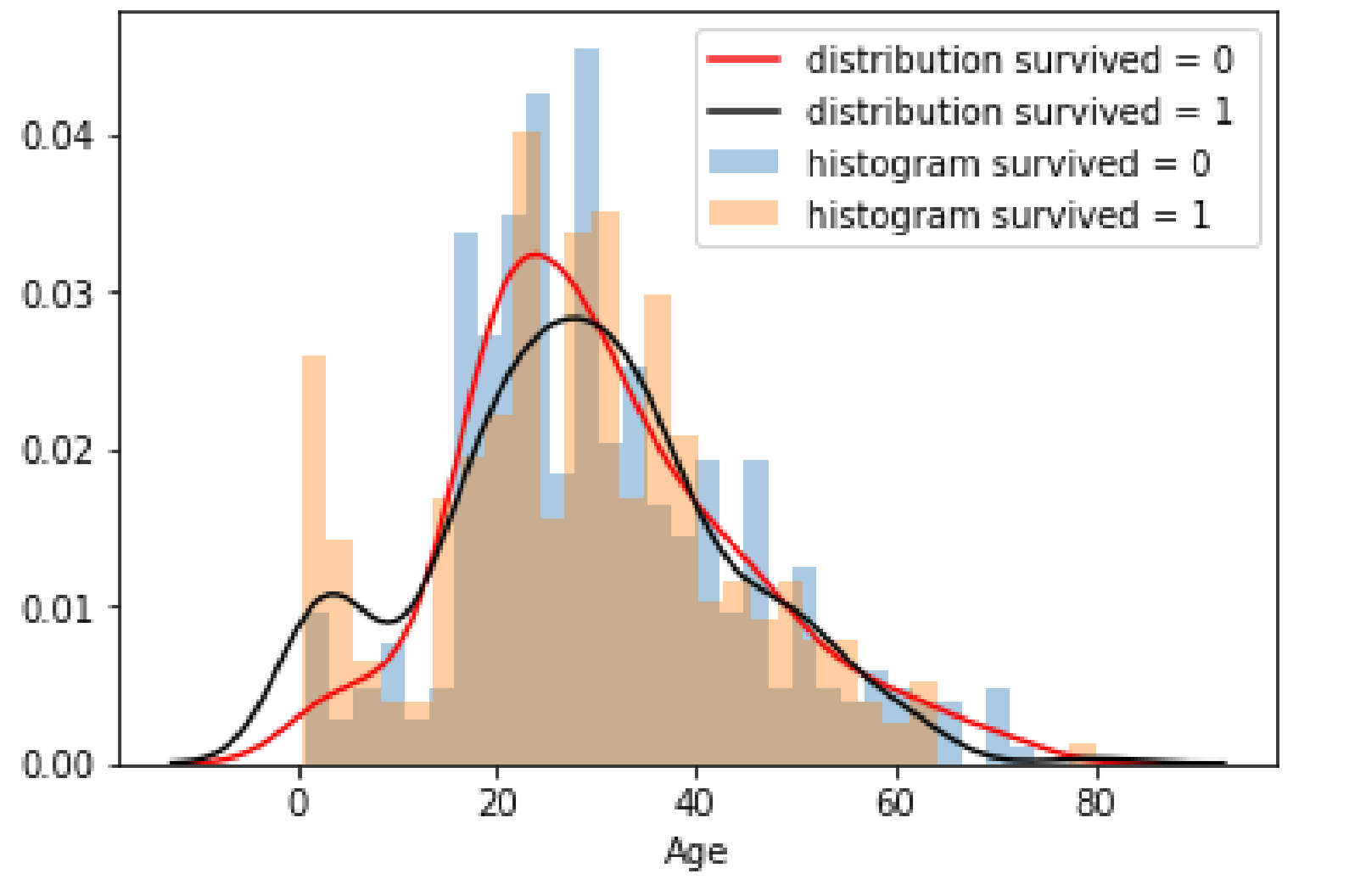


Figure 22. Histogram and Distribution of Age.

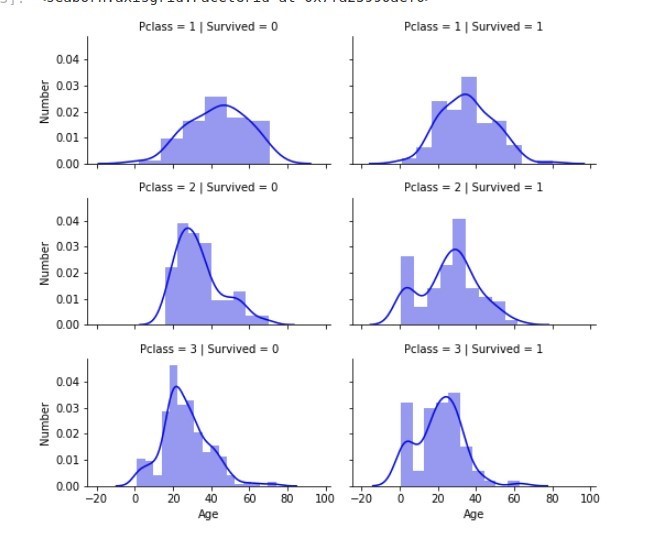


Figure 22. Distribution and Histogram of Age dependent on P Class and survived

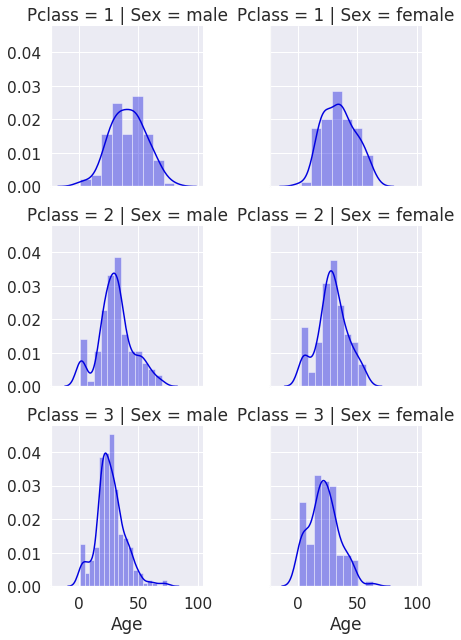


Figure 23. The Distribution and Histogram of Sex dependent on P Class and survived

1. **Test the operation of classifiers**

**LDA**

**Specificity** 0.945205479452054

8

**Sensitivity** 0.948529411764705

**Accuracy** 0.947368421052631

5

**Error** 0.040441176470588

24

**QDA**

**Specificity** 0.8773006134969326

**Sensitivity** 0.9647058823529412

**Accuracy** 0.930622009569378

**Error** 0.0568627450980392

15

**SVM**

**Specificity** 0.90625

**Sensitivity** 0.9728682170542635

**Accuracy** 0.9473684210526315

**Error** 0.0426356589147286

8

1. **Compare (and describe) the results.**

**Accuracy**

By analyzing the parameter “***Accuracy***”, we can see that whatever classifier you use, you get a good accuracy. But we can also see that SVM with an accuracy of slightly over 94%,the predictions(whether the person will survive or will die) were true.

**Sensitivity** and **Specificity** are statistical measures of the performance of a [binary classification](https://en.m.wikipedia.org/wiki/Binary_classification) [test.](https://en.m.wikipedia.org/wiki/Classification_rule)

* + **Sensitivity** (also called the **true positive rate**, the [**recall**,](https://en.m.wikipedia.org/wiki/Precision_and_recall#Definition_(classification_context)) or **probability of detection** in some fields) measures the proportion of actual positives that are correctly identified as such (in our example it is the percentage of dead people who are correctly identified as dead).
  + **Specificity** (also called the **true negative rate**) measures the proportion of actual negatives that are correctly identified as such (in our example it is the percentage of alive people who are correctly identified as dead).

Generally the best Classifier is that one which has high sensitivity with low specificity. We can say here that the LDA, with both Specificity and Sensitivity of 94% is the best Classifier, because for practical reasons, tests with sensitivity and specificity values above 90% have high credibility, albeit usually no certainty, in [differential diagnosis.](https://en.m.wikipedia.org/wiki/Differential_diagnosis)

The parameter “***Error*”** tells us the proportion of people that were badly predicted. The QDA has the biggest error rate of 5%.

1. **Choose the best classifier for the analyzing dataset**

The AUC of LDA= 0.93891

The AUC of SVM= 0.94878

The AUC of QDA= 0.93280

From my point of view, the best predictor here is SVM, with an AUC of almost 95%.This model will predict that such people will die or live, with a precision of 94.8%

**The Code**

import numpy as np import pandas as pd

from sklearn.model\_selection import train\_test\_split import matplotlib.pyplot as plt import scipy.stats from functools import reduce import math

from sklearn.metrics import confusion\_matrix import scipy.io

from sklearn.discriminant\_analysis import

LinearDiscriminantAnalysis,QuadraticDiscriminantAnalysis from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC from sklearn import metrics import scikitplot as skplt import pandas as pd import seaborn as sns from sklearn.datasets import make\_classification from sklearn.neighbors import KNeighborsClassifier from sklearn.ensemble import RandomForestClassifier from sklearn.model\_selection import train\_test\_split from sklearn.metrics import roc\_curve from sklearn.metrics import roc\_auc\_score from sklearn.metrics import confusion\_matrix from sklearn.metrics import accuracy\_score

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis as LDA data1 = pd.read\_csv('dataset\_1.csv',index\_col=0) data2 = pd.read\_csv('dataset\_2.csv',index\_col=0) plt.scatter(data1['x1'], data1['x2'], c=data1['y']) plt.show()

plt.scatter(data2['x1'], data2['x2'], c=data2['y']) plt.show() #dataset 1 sns\_plot = sns.boxplot(x="y", y="x1", hue="y",data=data1, palette="coolwarm") sns\_plot = sns.boxplot(x="y", y="x2", hue="y",data=data1, palette="coolwarm") #dataset 2 sns\_plot = sns.boxplot(x="y", y="x1", hue="y",data=data2, palette="coolwarm") sns\_plot = sns.boxplot(x="y", y="x2", hue="y",data=data2, palette="coolwarm")

#dataset 1

g = sns.distplot(data1['x1'][data1['y'] == 0], kde=True,bins=60,kde\_kws={"label": "distribution y =

0"}, hist\_kws={"label": "histogram y = 0" })

sns.distplot(data1['x1'][data1['y'] == 1], kde=True,bins=60,kde\_kws={"label": "distribution y = 1"}, hist\_kws={"label": "histogram y = 1" })

g = sns.distplot(data1['x2'][data1['y'] == 0], kde=True,bins=60,kde\_kws={"label": "distribution y =

0"}, hist\_kws={"label": "histogram y = 0" })

sns.distplot(data1['x2'][data1['y'] == 1], kde=True,bins=60,kde\_kws={"label": "distribution y = 1"}, hist\_kws={"label": "histogram y = 1" })

#dataset 2

g = sns.distplot(data2['x1'][data2['y'] == 0], kde=True,bins=60,kde\_kws={"label": "distribution y =

0"}, hist\_kws={"label": "histogram y = 0" })

sns.distplot(data2['x1'][data2['y'] == 1], kde=True,bins=60,kde\_kws={"label": "distribution y = 1"}, hist\_kws={"label": "histogram y = 1" })

g = sns.distplot(data2['x2'][data2['y'] == 0], kde=True,bins=60,kde\_kws={"label": "distribution y =

0"}, hist\_kws={"label": "histogram y = 0" })

sns.distplot(data2['x2'][data2['y'] == 1], kde=True,bins=60,kde\_kws={"label": "distribution y = 0"}, hist\_kws={"label": "histogram y = 0" })

X\_train\_linear, X\_test\_linear, y\_train\_linear, y\_test\_linear = train\_test\_split( data1[['x1','x2']], data1['y'], test\_size=0.3, random\_state=42)

X\_train\_nonlinear, X\_test\_nonlinear, y\_train\_nonlinear, y\_test\_nonlinear = train\_test\_split( data2[['x1','x2']], data2['y'], test\_size=0.3, random\_state=42) def get\_acc(true\_val,predicted): matrix=confusion\_matrix(true\_val, predicted).ravel() tn, fp, fn, tp = matrix acc=(tn+tp)/(tp+tn+fp+fn) return (acc,matrix,tp,fp,fn,tp) model\_lda = LinearDiscriminantAnalysis()

model\_qda=QuadraticDiscriminantAnalysis() model\_svm=SVC()

model\_lda.fit(X\_train\_linear, y\_train\_linear) model\_qda.fit(X\_train\_linear, y\_train\_linear) model\_svm.fit(X\_train\_linear, y\_train\_linear) prediction\_linear\_data\_lda=model\_lda.predict(X\_test\_linear) prediction\_linear\_data\_qda=model\_qda.predict(X\_test\_linear) prediction\_linear\_data\_svm=model\_svm.predict(X\_test\_linear) def plot\_roc\_curve(fpr, tpr): plt.plot(fpr, tpr, color='orange', label='ROC') plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--') plt.xlabel('False Positive Rate') plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve') plt.legend() plt.show()

fpr, tpr, thresholds = roc\_curve(y\_test\_linear, prediction\_linear\_data\_lda) plot\_roc\_curve(fpr, tpr)

fpr, tpr, thresholds = roc\_curve(y\_test\_linear, prediction\_linear\_data\_qda) plot\_roc\_curve(fpr, tpr)

fpr, tpr, thresholds = roc\_curve(y\_test\_linear, prediction\_linear\_data\_svm) plot\_roc\_curve(fpr, tpr)

cm = confusion\_matrix(y\_test\_linear, prediction\_linear\_data\_lda) ax= plt.subplots(figsize=(10,10)) sns.set(font\_scale=4) sns\_plot = sns.heatmap(cm, cbar=False, square=True, annot=True,fmt='g') Sensitivity = cm[0][0] / (cm[0][0] + cm[1][0]) # Sensitivity = TP / (TP + FN)

Specificity = cm[1][1] / (cm[1][1] + cm[0][1]) # Specificity = TN / (TN + FN)

Accuracy = (cm[0][0] + cm[1][1]) / (cm[0][0] + cm[1][0] + cm[0][1] + cm[1][1]) # Accuracy = TP + TN / (TP + TN + FP + FN)

Error = (cm[1][0] + cm[0][1]) / (cm[0][0] + cm[1][0] + cm[0][0] + cm[1][0]) # Error = FP + FN /

(TP + TN + FN + FP) print(Sensitivity) print(Specificity) print(Accuracy) print(Error)

auc = roc\_auc\_score(y\_test\_linear, prediction\_linear\_data\_lda) print('AUC: %.4f' % auc)

cm = confusion\_matrix(y\_test\_linear, prediction\_linear\_data\_qda) ax= plt.subplots(figsize=(10,10)) sns.set(font\_scale=4) sns\_plot = sns.heatmap(cm, cbar=False, square=True, annot=True,fmt='g') Sensitivity = cm[0][0] / (cm[0][0] + cm[1][0]) # Sensitivity = TP / (TP + FN)

Specificity = cm[1][1] / (cm[1][1] + cm[0][1]) # Specificity = TN / (TN + FN)

Accuracy = (cm[0][0] + cm[1][1]) / (cm[0][0] + cm[1][0] + cm[0][1] + cm[1][1]) # Accuracy = TP + TN / (TP + TN + FP + FN)

Error = (cm[1][0] + cm[0][1]) / (cm[0][0] + cm[1][0] + cm[0][0] + cm[1][0]) # Error = FP + FN /

(TP + TN + FN + FP) print(Sensitivity) print(Specificity) print(Accuracy) print(Error)

auc = roc\_auc\_score(y\_test\_linear, prediction\_linear\_data\_qda) print('AUC: %.4f' % auc)

cm = confusion\_matrix(y\_test\_linear, prediction\_linear\_data\_svm) ax= plt.subplots(figsize=(10,10)) sns.set(font\_scale=1.7) sns\_plot = sns.heatmap(cm, cbar=False, square=True, annot=True,fmt='g') Sensitivity = cm[0][0] / (cm[0][0] + cm[1][0]) # Sensitivity = TP / (TP + FN)

Specificity = cm[1][1] / (cm[1][1] + cm[0][1]) # Specificity = TN / (TN + FN)

Accuracy = (cm[0][0] + cm[1][1]) / (cm[0][0] + cm[1][0] + cm[0][1] + cm[1][1]) # Accuracy = TP + TN / (TP + TN + FP + FN)

Error = (cm[1][0] + cm[0][1]) / (cm[0][0] + cm[1][0] + cm[0][0] + cm[1][0]) # Error = FP + FN /

(TP + TN + FN + FP) print(Sensitivity) print(Specificity) print(Accuracy) print(Error)

auc = roc\_auc\_score(y\_test\_linear, prediction\_linear\_data\_svm) print('AUC: %.4f' % auc)

model\_lda = LinearDiscriminantAnalysis() model\_qda=QuadraticDiscriminantAnalysis()

model\_svm=SVC()

model\_lda.fit(X\_train\_nonlinear, y\_train\_nonlinear) model\_qda.fit(X\_train\_nonlinear, y\_train\_nonlinear) model\_svm.fit(X\_train\_nonlinear, y\_train\_nonlinear) prediction\_nonlinear\_data\_lda=model\_lda.predict(X\_test\_nonlinear) prediction\_nonlinear\_data\_qda=model\_qda.predict(X\_test\_nonlinear) prediction\_nonlinear\_data\_svm=model\_svm.predict(X\_test\_nonlinear) fpr, tpr, thresholds = roc\_curve(y\_test\_nonlinear, prediction\_nonlinear\_data\_lda) plot\_roc\_curve(fpr, tpr)

Sensitivity = cm[0][0] / (cm[0][0] + cm[1][0]) # Sensitivity = TP / (TP + FN)

Specificity = cm[1][1] / (cm[1][1] + cm[0][1]) # Specificity = TN / (TN + FN)

Accuracy = (cm[0][0] + cm[1][1]) / (cm[0][0] + cm[1][0] + cm[0][1] + cm[1][1]) # Accuracy = TP + TN / (TP + TN + FP + FN)

Error = (cm[1][0] + cm[0][1]) / (cm[0][0] + cm[1][0] + cm[0][0] + cm[1][0]) # Error = FP + FN /

(TP + TN + FN + FP) print(Sensitivity) print(Specificity) print(Accuracy) print(Error)

auc = roc\_auc\_score(y\_test\_nonlinear, prediction\_nonlinear\_data\_lda) print('AUC: %.4f' % auc)

fpr, tpr, thresholds = roc\_curve(y\_test\_nonlinear, prediction\_nonlinear\_data\_qda) plot\_roc\_curve(fpr, tpr)

Sensitivity = cm[0][0] / (cm[0][0] + cm[1][0]) # Sensitivity = TP / (TP + FN)

Specificity = cm[1][1] / (cm[1][1] + cm[0][1]) # Specificity = TN / (TN + FN)

Accuracy = (cm[0][0] + cm[1][1]) / (cm[0][0] + cm[1][0] + cm[0][1] + cm[1][1]) # Accuracy = TP + TN / (TP + TN + FP + FN)

Error = (cm[1][0] + cm[0][1]) / (cm[0][0] + cm[1][0] + cm[0][0] + cm[1][0]) # Error = FP + FN /

(TP + TN + FN + FP) print(Sensitivity) print(Specificity) print(Accuracy) print(Error)

auc = roc\_auc\_score(y\_test\_nonlinear, prediction\_nonlinear\_data\_qda) print('AUC: %.4f' % auc)

fpr, tpr, thresholds = roc\_curve(y\_test\_nonlinear, prediction\_nonlinear\_data\_qda) plot\_roc\_curve(fpr, tpr)

Sensitivity = cm[0][0] / (cm[0][0] + cm[1][0]) # Sensitivity = TP / (TP + FN)

Specificity = cm[1][1] / (cm[1][1] + cm[0][1]) # Specificity = TN / (TN + FN)

Accuracy = (cm[0][0] + cm[1][1]) / (cm[0][0] + cm[1][0] + cm[0][1] + cm[1][1]) # Accuracy = TP + TN / (TP + TN + FP + FN)

Error = (cm[1][0] + cm[0][1]) / (cm[0][0] + cm[1][0] + cm[0][0] + cm[1][0]) # Error = FP + FN /

(TP + TN + FN + FP) print(Sensitivity) print(Specificity) print(Accuracy) print(Error)

auc = roc\_auc\_score(y\_test\_nonlinear, prediction\_nonlinear\_data\_qda) print('AUC: %.4f' % auc)

fpr, tpr, thresholds = roc\_curve(y\_test\_nonlinear, prediction\_nonlinear\_data\_svm) plot\_roc\_curve(fpr, tpr)

Sensitivity = cm[0][0] / (cm[0][0] + cm[1][0]) # Sensitivity = TP / (TP + FN)

Specificity = cm[1][1] / (cm[1][1] + cm[0][1]) # Specificity = TN / (TN + FN)

Accuracy = (cm[0][0] + cm[1][1]) / (cm[0][0] + cm[1][0] + cm[0][1] + cm[1][1]) # Accuracy = TP + TN / (TP + TN + FP + FN)

Error = (cm[1][0] + cm[0][1]) / (cm[0][0] + cm[1][0] + cm[0][0] + cm[1][0]) # Error = FP + FN /

(TP + TN + FN + FP) print(Sensitivity) print(Specificity) print(Accuracy) print(Error)

auc = roc\_auc\_score(y\_test\_nonlinear, prediction\_nonlinear\_data\_svm) print('AUC: %.4f' % auc) Titanic code:

# Importing packages

import pandas as pd import numpy as np import random as rnd

# Importing visualization packages import seaborn as sns import matplotlib.pyplot as plt

%matplotlib inline

# Importing machine learning packages from sklearn.linear\_model import LogisticRegression from sklearn.svm import SVC, LinearSVC from sklearn.ensemble import RandomForestClassifier from sklearn.neighbors import KNeighborsClassifier from sklearn.tree import DecisionTreeClassifier from sklearn.metrics import roc\_curve from sklearn.metrics import roc\_auc\_score

from sklearn.discriminant\_analysis import QuadraticDiscriminantAnalysis as QDA df\_train = pd.read\_csv('train.csv')

df\_survived = pd.read\_csv('gender\_submission.csv')

df\_test = pd.read\_csv('test.csv') combine = [df\_train, df\_test]

pclass\_age\_grid = sns.FacetGrid(df\_train, col='Survived',row='Pclass', size=2.2, aspect=1.6) pclass\_age\_grid.map(sns.distplot, "Age", hist=True, color="#0000DD") pclass\_age\_grid.add\_legend() pclass\_age\_grid.set\_ylabels('Number') df\_train = df\_train.drop(['Ticket', 'Cabin'], axis=1) df\_test = df\_test.drop(['Ticket', 'Cabin'], axis=1) combine = [df\_train, df\_test] for dataset in combine: dataset['Title'] = dataset.Name.str.extract('([A-Za-z]+)\.', expand=False) pd.crosstab(df\_train['Title'], df\_train['Sex']) for dataset in combine:

dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss') dataset['Title'] = dataset['Title'].replace('Ms', 'Miss') dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')

dataset['Title'] = dataset['Title'].replace(['Lady', 'Countess','Don', 'Sir', 'Jonkheer', 'Dona'],'Royalty') dataset['Title'] = dataset['Title'].replace(['Capt', 'Col','Dr','Major','Rev'],'Special')

df\_train[['Title','Survived']].groupby(['Title'], as\_index=False).mean() title\_mapping = {"Master": 1, "Miss": 2, "Mrs": 3, "Mr": 4, "Royalty": 5, "Special": 6} for dataset in combine: dataset['Title'] = dataset['Title'].map(title\_mapping)

dataset['Title'] = dataset['Title'].fillna(0)

df\_train = df\_train.drop(['Name', 'PassengerId'], axis=1) df\_test = df\_test.drop(['Name'], axis=1) combine = [df\_train, df\_test] for dataset in combine: dataset['Sex'] = dataset['Sex'].map( {'female': 1, 'male': 0} ).astype(int) df\_train.head() pclass\_sex\_age\_grid = sns.FacetGrid(df\_train, row='Pclass', col='Sex') pclass\_sex\_age\_grid.map(sns.distplot, "Age", hist=True, color="#0000DD") pclass\_sex\_age\_grid.add\_legend() df\_train.Age.isnull().sum() median\_age = np.zeros((2,3)) for dataset in combine: for sex in range(0,2): for pclass in range(0,3): guess\_df = dataset[(dataset['Sex'] == sex) & \ (dataset['Pclass'] == pclass+1)]['Age'].dropna()

age\_guess = guess\_df.median() median\_age[sex,pclass] = age\_guess median\_age for dataset in combine: for i in range(0, 2): for j in range(0, 3): dataset.loc[ (dataset.Age.isnull()) & (dataset.Sex == i) & (dataset.Pclass == j+1),\

'Age'] = median\_age[i,j]

dataset['Age'] = dataset['Age'].astype(int) df\_train.head() df\_train['AgeBand'] = pd.cut(df\_train['Age'], 5) df\_train[['AgeBand', 'Survived']].groupby(['AgeBand'], as\_index=False).mean().sort\_values(by='AgeBand', ascending=True) for dataset in combine: dataset.loc[ dataset['Age'] <= 16, 'AgeG'] = 0 dataset.loc[(dataset['Age'] > 16) & (dataset['Age'] <= 32), 'AgeG'] = 1 dataset.loc[(dataset['Age'] > 32) & (dataset['Age'] <= 48), 'AgeG'] = 2 dataset.loc[(dataset['Age'] > 48) & (dataset['Age'] <= 64), 'AgeG'] = 3 dataset.loc[ dataset['Age'] > 64, 'AgeG'] = 4 df\_train.head()

df\_train = df\_train.drop(['AgeBand','Age'], axis=1) combine = [df\_train, df\_test] df\_train.head() for dataset in combine: dataset['FamilySize'] = dataset['SibSp'] + dataset['Parch'] + 1 df\_train[['FamilySize','Survived']].groupby(['FamilySize'], as\_index=False).mean().sort\_values(by='Survived', ascending=False)

df\_train = df\_train.drop(['SibSp', 'Parch'], axis=1) df\_test = df\_test.drop(['SibSp','Parch'], axis=1) combine = [df\_train, df\_test] df\_train.head()

freq\_port = df\_train.Embarked.dropna().mode()[0] freq\_port for dataset in combine: dataset['Embarked'] = dataset['Embarked'].fillna(freq\_port) df\_train[['Embarked','Survived']].groupby(['Embarked'], as\_index = False).mean().sort\_values(by='Survived', ascending=False) for dataset in combine: dataset['Embarked'] = dataset['Embarked'].map( {'S': 0, 'C': 1, 'Q': 2} ).astype(int) df\_train.head() df\_train = df\_train.drop(['Fare'], axis=1) df\_test = df\_test.drop(['Fare'], axis=1) combine = [df\_train, df\_test] df\_train.head() for dataset in combine: dataset.AgeG = dataset.AgeG.astype(int) df\_train.head()

df\_test = df\_test.drop(['Age'], axis=1) df\_test.head()

X\_train = df\_train.drop("Survived", axis=1)

Y\_train = df\_train['Survived']

X\_test = df\_test.drop('PassengerId', axis=1).copy()

svc = SVC()

svc.fit(X\_train, Y\_train) Y\_pred = svc.predict(X\_test) acc\_svm = round(svc.score(X\_train, Y\_train) \*100 ,2) def plot\_roc\_curve(fpr, tpr): plt.plot(fpr, tpr, color='orange', label='ROC') plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--') plt.xlabel('False Positive Rate') plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve') plt.legend() plt.show() fig = plt.figure()

fpr, tpr, thresholds = roc\_curve(df\_survived['Survived'], Y\_pred) auc = roc\_auc\_score(df\_survived['Survived'], Y\_pred) print('AUC: %.5f' % auc)

from sklearn.metrics import classification\_report,confusion\_matrix cm = confusion\_matrix(df\_survived['Survived'], Y\_pred) print(confusion\_matrix(df\_survived['Survived'], Y\_pred))

Sensitivity = cm[0][0] / (cm[0][0] + cm[1][0]) # Sensitivity = TP / (TP + FN)

Specificity = cm[1][1] / (cm[1][1] + cm[0][1]) # Specificity = TN / (TN + FN)

Accuracy = (cm[0][0] + cm[1][1]) / (cm[0][0] + cm[1][0] + cm[0][1] + cm[1][1]) # Accuracy = TP + TN / (TP + TN + FP + FN)

Error = (cm[1][0] + cm[0][1]) / (cm[0][0] + cm[1][0] + cm[0][0] + cm[1][0]) # Error = FP + FN /

(TP + TN + FN + FP) print(Sensitivity) print(Specificity) print(Accuracy) print(Error)

qda\_model=QDA()

qda\_model.fit(X\_train, Y\_train) Y\_pred = qda\_model.predict(X\_test)

fpr, tpr, thresholds = roc\_curve(df\_survived['Survived'], Y\_pred) auc = roc\_auc\_score(df\_survived['Survived'], Y\_pred) print('AUC: %.5f' % auc) cm = confusion\_matrix(df\_survived['Survived'], Y\_pred) print(confusion\_matrix(df\_survived['Survived'], Y\_pred)) Sensitivity = cm[0][0] / (cm[0][0] + cm[1][0]) # Sensitivity = TP / (TP + FN)

Specificity = cm[1][1] / (cm[1][1] + cm[0][1]) # Specificity = TN / (TN + FN)

Accuracy = (cm[0][0] + cm[1][1]) / (cm[0][0] + cm[1][0] + cm[0][1] + cm[1][1]) # Accuracy = TP + TN / (TP + TN + FP + FN)

Error = (cm[1][0] + cm[0][1]) / (cm[0][0] + cm[1][0] + cm[0][0] + cm[1][0]) # Error = FP + FN /

(TP + TN + FN + FP) print(Sensitivity) print(Specificity) print(Accuracy) print(Error)

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis as LDA lda=LDA()

lda.fit(X\_train, Y\_train) Y\_pred = lda.predict(X\_test) auc = roc\_auc\_score(y\_test, y\_pred) print('AUC: %.5f' % auc) cm = confusion\_matrix(df\_survived['Survived'], Y\_pred) print(confusion\_matrix(df\_survived['Survived'], Y\_pred)) fpr, tpr, thresholds = roc\_curve(df\_survived['Survived'], Y\_pred) auc = roc\_auc\_score(df\_survived['Survived'], Y\_pred)

print('AUC: %.5f' % auc)

Sensitivity = cm[0][0] / (cm[0][0] + cm[1][0]) # Sensitivity = TP / (TP + FN)

Specificity = cm[1][1] / (cm[1][1] + cm[0][1]) # Specificity = TN / (TN + FN)

Accuracy = (cm[0][0] + cm[1][1]) / (cm[0][0] + cm[1][0] + cm[0][1] + cm[1][1]) # Accuracy = TP + TN / (TP + TN + FP + FN)

Error = (cm[1][0] + cm[0][1]) / (cm[0][0] + cm[1][0] + cm[0][0] + cm[1][0]) # Error = FP + FN /

(TP + TN + FN + FP) print(Sensitivity) print(Specificity) print(Accuracy) print(Error)