# Coursework 1 Group 18

Group Number 18

1. Pratiksha Rajendran (K2203709)
2. Sachit Nanajkar (K2166043)
3. Sujay Grama Suresh Kumar (K2201621)
4. Vishnuvardhan Gorantla (K2259314)

# Part I: Load Data

#### Importing all the required libraries

import numpy as np  
from sklearn.model\_selection import train\_test\_split  
from sklearn.cluster import KMeans  
import pandas as pd #loaded for visualizing the data  
  
import warnings  
warnings.filterwarnings('ignore')

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.datasets import load\_wine  
from sklearn.metrics import silhouette\_score, calinski\_harabasz\_score, davies\_bouldin\_score, v\_measure\_score, adjusted\_rand\_score, accuracy\_score, roc\_auc\_score  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import StandardScaler  
from sklearn.cluster import KMeans, AgglomerativeClustering, Birch, DBSCAN, AffinityPropagation, SpectralClustering, SpectralCoclustering  
from sklearn.decomposition import PCA  
from sklearn.manifold import TSNE  
from sklearn.cluster import MeanShift, estimate\_bandwidth  
import seaborn as sns

### Load data (code)

data = load\_wine()  
  
X = data.data  
  
Y = data.target

### Identifying the Key aspects

print("The number of samples/instances in the wine data is:\033[1m",len(Y),"\033[0m")  
  
print("The number of dimensions/features/attributes in the dataset is:\033[1m",len(X[0]),"\033[0m")  
  
print("The number of classes in the dataset is:\033[1m",len(data.target\_names),"\033[0m")  
  
print("The class names of the data are:\033[1m", ', '.join(data.target\_names),"\033[0m")

The number of samples/instances in the wine data is: 178   
The number of dimensions/features/attributes in the dataset is: 13   
The number of classes in the dataset is: 3   
The class names of the data are: class\_0, class\_1, class\_2

unique\_values, count = np.unique(Y, return\_counts=True)  
print("The number of samples per class:")  
print(" class\_0 wine has\033[1m", count[0],"\033[0m" "samples", "\n", "class\_1 wine has\033[1m", count[1],"\033[0m" "samples", "\n", "class\_2 wine has\033[1m", count[2],"\033[0m" "samples")

The number of samples per class:  
 class\_0 wine has 59 samples   
 class\_1 wine has 71 samples   
 class\_2 wine has 48 samples

# Part II: Clustering

Clustering Methods Used are:

1) KMeans

2) Agglomerative Clustering

3) DBSCAN [5]

4) BIRCH [6]

5) Affinity Propagation [7]

6) Mean-Shift [8]

# 1. KMeans Clutering Method

#### Training using KMeans algorithm for 3 clusters

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

kmeans = KMeans(n\_clusters=3, n\_init=100, init='random')

kmeans.fit(x\_train, y\_train)

KMeans(init='random', n\_clusters=3, n\_init=100)

predicted\_classes = kmeans.predict(x\_test)

#### Internal evaluation metrics for KMeans

shc = silhouette\_score(x\_test, predicted\_classes)  
  
chs = calinski\_harabasz\_score(x\_test, predicted\_classes)  
  
dbs = davies\_bouldin\_score(x\_test, predicted\_classes)   
  
print("The \033[1m silhouette score of KMeans \033[0m is:\033[1m", shc,"\033[0m")  
print("The \033[1m calinski harabasz score of KMeans \033[0m is:\033[1m", chs,"\033[0m")  
print("The \033[1m davies bouldin score of KMeans \033[0m is:\033[1m", dbs,"\033[0m")

The silhouette score of KMeans is: 0.5698497328652201   
The calinski harabasz score of KMeans is: 102.28143693890466   
The davies bouldin score of KMeans is: 0.5202402865293374

#### External evaluation metrics for KMeans

vms = v\_measure\_score(y\_test, predicted\_classes)  
ars = adjusted\_rand\_score(y\_test, predicted\_classes)  
  
print("The \033[1mv measure score of KMeans \033[0m is:\033[1m", vms,"\033[0m")  
print("The \033[1madjusted rand score of KMeans \033[0m is:\033[1m", ars,"\033[0m")

The v measure score of KMeans is: 0.5365783935489853   
The adjusted rand score of KMeans is: 0.4927536231884058

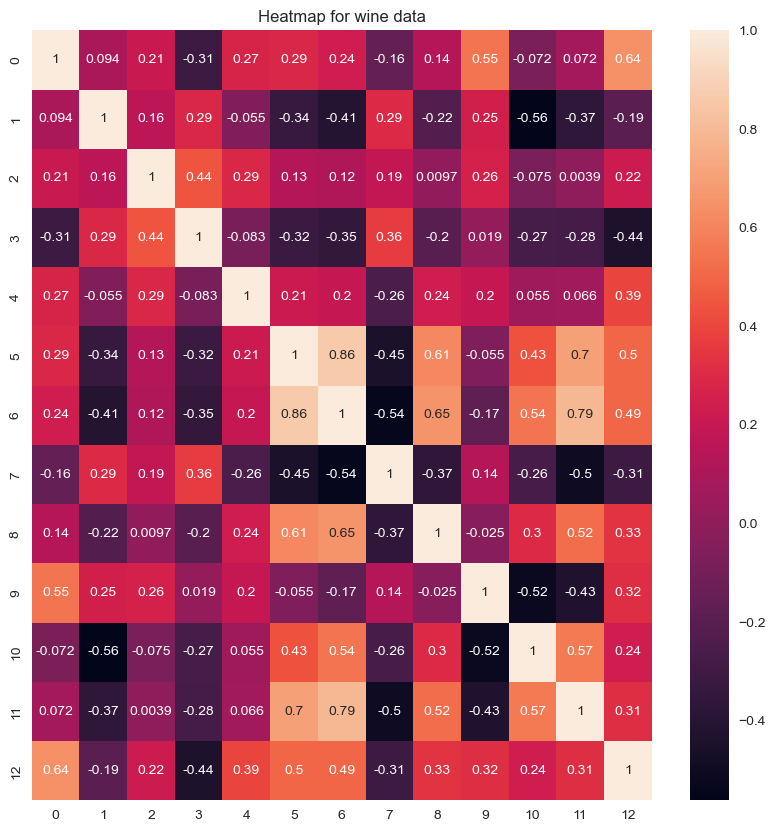
#### Let's try to visualize the clusters and assess the performance of the algorithm

wine = pd.DataFrame(X)  
wine.columns = data.feature\_names  
wine['classes'] = Y  
wine.shape

(178, 14)

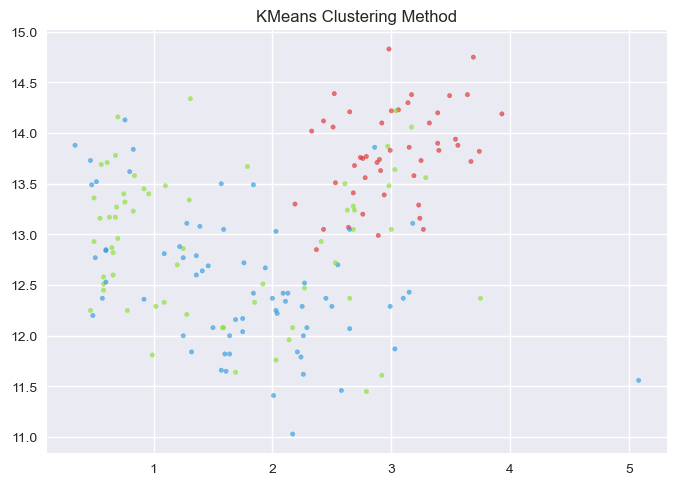
## The heatmap, shows the correlation between all the features

plt.figure(figsize=(10,10))  
wine\_data = pd.DataFrame(X)  
wine\_data\_corr = wine\_data.corr()  
sns.heatmap(wine\_data\_corr, annot=True)  
plt.title('Heatmap for wine data')  
plt.show()



# [4]  
wine['cluster'] = kmeans.fit\_predict(X, Y)  
centroids = kmeans.cluster\_centers\_  
cen\_x = [i[0] for i in centroids]   
cen\_y = [i[1] for i in centroids]  
## add to df  
wine['cen\_x'] = wine.cluster.map({0:cen\_x[0], 1:cen\_x[1], 2:cen\_x[2]})  
wine['cen\_y'] = wine.cluster.map({0:cen\_y[0], 1:cen\_y[1], 2:cen\_y[2]})  
# define and map colors  
colors = ['#DF2020', '#81DF20', '#2095DF']  
wine['c'] = wine.cluster.map({0:colors[0], 1:colors[1], 2:colors[2]})  
  
plt.scatter(wine['flavanoids'], wine['alcohol'], c=wine.c, alpha = 0.6, s=10)  
plt.title("KMeans Clustering Method")

Text(0.5, 1.0, 'KMeans Clustering Method')



#### Let's use PCA and decrease the dimensionality of the data to 7 dimensions from 13 dimensions

pca = PCA(n\_components=7)  
x\_pca = pca.fit\_transform(X)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_pca, Y, test\_size=0.2, random\_state=42)

kmeans\_pca = KMeans(n\_clusters=3, n\_init=100, init='random')

kmeans\_pca.fit(x\_train, y\_train)

KMeans(init='random', n\_clusters=3, n\_init=100)

y\_pred\_kmeans\_pca = kmeans\_pca.predict(x\_test)

#### Internal Evaluation metrics

shc = silhouette\_score(x\_test, y\_test)  
  
chs = calinski\_harabasz\_score(x\_test, y\_test)  
  
dbs = davies\_bouldin\_score(x\_test, y\_test)   
  
print("The \033[1m silhouette score of KMeans with PCA \033[0m is:\033[1m", shc,"\033[0m")  
print("The \033[1m calinski harabasz score of KMeans with PCA \033[0m is:\033[1m", chs,"\033[0m")  
print("The \033[1m davies bouldin score of KMeans with PCA \033[0m is:\033[1m", dbs,"\033[0m")

The silhouette score of KMeans with PCA is: 0.23691906958028433   
The calinski harabasz score of KMeans with PCA is: 53.417795125668285   
The davies bouldin score of KMeans with PCA is: 1.1395026889910211

#### External evaluation metrics

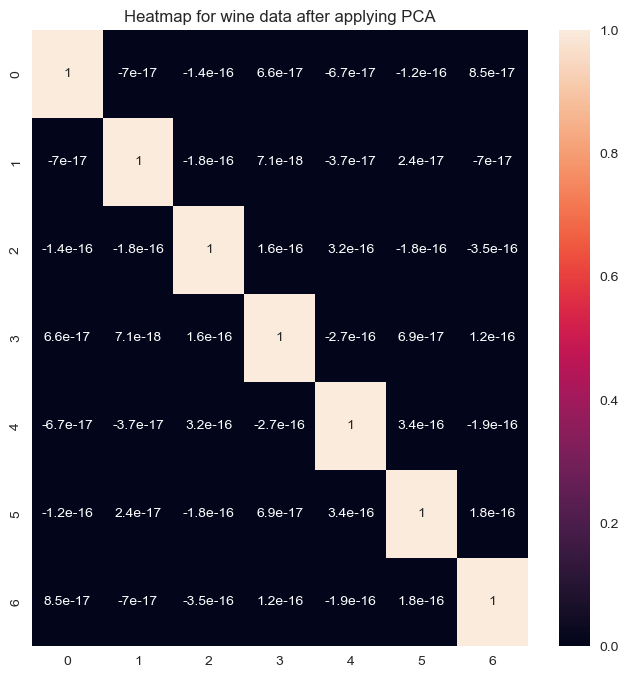
vms = v\_measure\_score(y\_test, predicted\_classes)  
ars = adjusted\_rand\_score(y\_test, predicted\_classes)  
  
print("The \033[1mv measure score of KMeans with PCA \033[0m is:\033[1m", vms,"\033[0m")  
print("The \033[1madjusted rand score of KMeans with PCA \033[0m is:\033[1m", ars,"\033[0m")

The v measure score of KMeans with PCA is: 0.5365783935489853   
The adjusted rand score of KMeans with PCA is: 0.4927536231884058

wine\_pca = pd.DataFrame(x\_pca)  
wine\_pca.columns = ['pca\_1', 'pca\_2', 'pca\_3', 'pca\_4', 'pca\_5', 'pca\_6', 'pca\_7']  
wine\_pca['classes'] = Y  
wine\_pca.shape

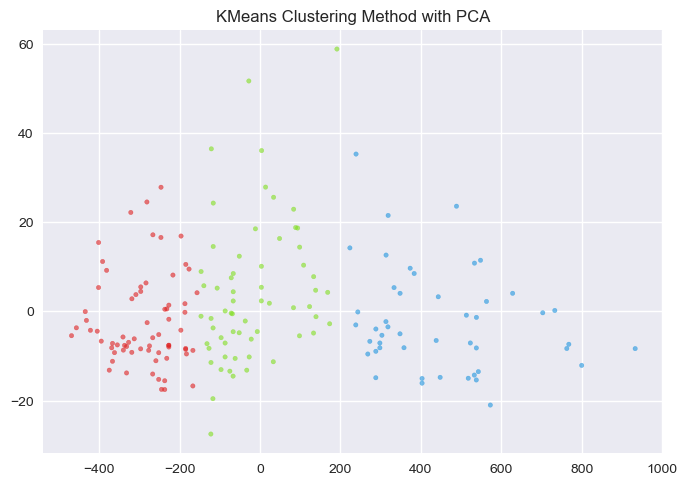
(178, 8)

plt.figure(figsize=(8,8))  
wine\_pca\_data = pd.DataFrame(x\_pca)  
wine\_pca\_corr = wine\_pca\_data.corr()  
sns.heatmap(wine\_pca\_corr, annot=True)  
plt.title('Heatmap for wine data after applying PCA')  
plt.show()



#[4]  
wine\_pca['cluster'] = kmeans\_pca.fit\_predict(X, Y)  
centroids = kmeans\_pca.cluster\_centers\_  
cen\_x = [i[0] for i in centroids]   
cen\_y = [i[1] for i in centroids]  
## add to df  
wine\_pca['cen\_x'] = wine\_pca.cluster.map({0:cen\_x[0], 1:cen\_x[1], 2:cen\_x[2]})  
wine\_pca['cen\_y'] = wine\_pca.cluster.map({0:cen\_y[0], 1:cen\_y[1], 2:cen\_y[2]})  
# define and map colors  
colors = ['#DF2020', '#81DF20', '#2095DF']  
wine\_pca['c'] = wine\_pca.cluster.map({0:colors[0], 1:colors[1], 2:colors[2]})  
  
plt.scatter(wine\_pca['pca\_1'], wine\_pca['pca\_2'], c=wine\_pca.c, alpha = 0.6, s=10)  
plt.title("KMeans Clustering Method with PCA")

Text(0.5, 1.0, 'KMeans Clustering Method with PCA')



## Let's increase the number of clusters and assess the performance of the same algorithm

kmeans\_5clusters = KMeans(n\_clusters=5, n\_init=100, init='random')

kmeans\_5clusters.fit(x\_train, y\_train)

KMeans(init='random', n\_clusters=5, n\_init=100)

y\_pred\_5clusters = kmeans\_5clusters.predict(x\_test)

#### Internal evaluation metrics

shc = silhouette\_score(x\_test, y\_pred\_5clusters)  
  
chs = calinski\_harabasz\_score(x\_test, y\_pred\_5clusters)  
  
dbs = davies\_bouldin\_score(x\_test, y\_pred\_5clusters)   
  
print("The \033[1m silhouette score of KMeans wih 5 clusters \033[0m is:\033[1m", shc,"\033[0m")  
print("The \033[1m calinski harabasz score of KMeans wih 5 clusters \033[0m is:\033[1m", chs,"\033[0m")  
print("The \033[1m davies bouldin score of KMeans wih 5 clusters \033[0m is:\033[1m", dbs,"\033[0m")

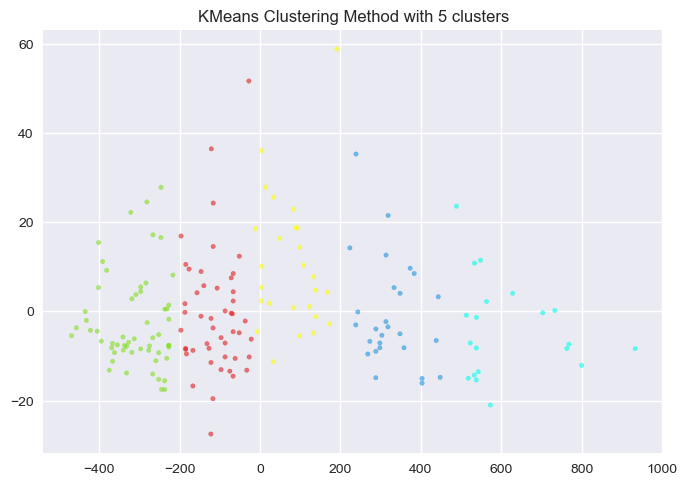
The silhouette score of KMeans wih 5 clusters is: 0.603774596055664   
The calinski harabasz score of KMeans wih 5 clusters is: 156.13047548667868   
The davies bouldin score of KMeans wih 5 clusters is: 0.47954292333880844

wine\_pca\_5clusters = pd.DataFrame(x\_pca)  
wine\_pca\_5clusters.columns = ['pca\_1', 'pca\_2', 'pca\_3', 'pca\_4', 'pca\_5', 'pca\_6', 'pca\_7']  
wine\_pca\_5clusters['classes'] = Y  
wine\_pca\_5clusters.shape

(178, 8)

#[4]  
wine\_pca\_5clusters['cluster'] = kmeans\_5clusters.fit\_predict(X, Y)  
centroids = kmeans\_5clusters.cluster\_centers\_  
cen\_x = [i[0] for i in centroids]   
cen\_y = [i[1] for i in centroids]  
## add to df  
wine\_pca\_5clusters['cen\_x'] = wine\_pca\_5clusters.cluster.map({0:cen\_x[0], 1:cen\_x[1], 2:cen\_x[2], 3:cen\_x[3], 4:cen\_x[4]})  
wine\_pca\_5clusters['cen\_y'] = wine\_pca\_5clusters.cluster.map({0:cen\_y[0], 1:cen\_y[1], 2:cen\_y[2], 3:cen\_x[3], 4:cen\_x[4]})  
# define and map colors  
colors = ['#DF2020', '#81DF20', '#2095DF', '#03FFEC', '#FFFF03']  
wine\_pca\_5clusters['c'] = wine\_pca\_5clusters.cluster.map({0:colors[0], 1:colors[1], 2:colors[2], 3:colors[3], 4:colors[4]})  
  
plt.scatter(wine\_pca\_5clusters['pca\_1'],   
 wine\_pca\_5clusters['pca\_2'],   
 c=wine\_pca\_5clusters.c,   
 alpha = 0.6,   
 s=10)  
plt.title("KMeans Clustering Method with 5 clusters")

Text(0.5, 1.0, 'KMeans Clustering Method with 5 clusters')



# 2. Agglomerative Clustering Method

agc = AgglomerativeClustering(n\_clusters=3)

agc.fit(x\_train, y\_train)

AgglomerativeClustering(n\_clusters=3)

y\_pred\_agc = agc.fit\_predict(x\_test)

accuracy\_score(y\_test, y\_pred\_agc)

0.5277777777777778

#### Internal Evaluation Metrics for Agglomerative Custering

shc = silhouette\_score(x\_test, y\_test)  
  
chs = calinski\_harabasz\_score(x\_test, y\_test)  
  
dbs = davies\_bouldin\_score(x\_test, y\_test)   
  
print("The \033[1m silhouette score of Agglomerative Clustering \033[0m is:\033[1m", shc,"\033[0m")  
print("The \033[1m calinski harabasz score of Agglomerative Clustering \033[0m is:\033[1m", chs,"\033[0m")  
print("The \033[1m davies bouldin score of Agglomerative Clustering \033[0m is:\033[1m", dbs,"\033[0m")

The silhouette score of Agglomerative Clustering is: 0.23691906958028433   
The calinski harabasz score of Agglomerative Clustering is: 53.417795125668285   
The davies bouldin score of Agglomerative Clustering is: 1.1395026889910211

#### External Evaluation Metrics for Agglomerative Custering

vms = v\_measure\_score(y\_test, y\_pred\_agc)  
ars = adjusted\_rand\_score(y\_test, y\_pred\_agc)  
  
print("The \033[1mv measure score of Agglomerative Clustering \033[0m is:\033[1m", vms,"\033[0m")  
print("The \033[1madjusted rand score of Agglomerative Clustering \033[0m is:\033[1m", ars,"\033[0m")

The v measure score of Agglomerative Clustering is: 0.5157977648795584   
The adjusted rand score of Agglomerative Clustering is: 0.45355850422195415

wine = pd.DataFrame(X)  
wine.columns = data.feature\_names  
wine['classes'] = Y  
wine.shape

(178, 14)

wine.head()

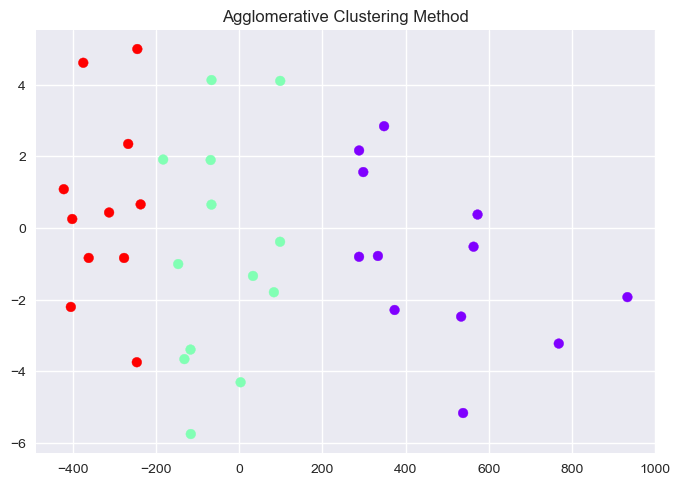
alcohol malic\_acid ash alcalinity\_of\_ash magnesium total\_phenols \  
0 14.23 1.71 2.43 15.6 127.0 2.80   
1 13.20 1.78 2.14 11.2 100.0 2.65   
2 13.16 2.36 2.67 18.6 101.0 2.80   
3 14.37 1.95 2.50 16.8 113.0 3.85   
4 13.24 2.59 2.87 21.0 118.0 2.80   
  
 flavanoids nonflavanoid\_phenols proanthocyanins color\_intensity hue \  
0 3.06 0.28 2.29 5.64 1.04   
1 2.76 0.26 1.28 4.38 1.05   
2 3.24 0.30 2.81 5.68 1.03   
3 3.49 0.24 2.18 7.80 0.86   
4 2.69 0.39 1.82 4.32 1.04   
  
 od280/od315\_of\_diluted\_wines proline classes   
0 3.92 1065.0 0   
1 3.40 1050.0 0   
2 3.17 1185.0 0   
3 3.45 1480.0 0   
4 2.93 735.0 0

agc.labels\_

array([1, 0, 1, 0, 2, 0, 1, 1, 1, 1, 1, 2, 0, 2, 0, 2, 2, 1, 0, 2, 0, 2,  
 1, 1, 2, 1, 2, 1, 2, 0, 0, 2, 1, 0, 0, 0], dtype=int64)

#[3]  
plt.scatter(x=x\_test[:,0], y=x\_test[:,2], c = agc.labels\_, cmap='rainbow')  
plt.title("Agglomerative Clustering Method")

Text(0.5, 1.0, 'Agglomerative Clustering Method')



#### Agglomerative clustering with PCA and Standard Scaler

sclr = StandardScaler()  
pca = PCA(n\_components=7)

x\_pca = pca.fit\_transform(sclr.fit\_transform(X))

y\_pred\_agc\_scle\_pca = agc.fit\_predict(x\_pca,Y)

#### Internal Evaluation Metrics for Agglomerative Clustering with PCA and Standard Scaler

shc = silhouette\_score(x\_pca, y\_pred\_agc\_scle\_pca)  
  
chs = calinski\_harabasz\_score(x\_pca, y\_pred\_agc\_scle\_pca)  
  
dbs = davies\_bouldin\_score(x\_pca, y\_pred\_agc\_scle\_pca)   
  
print("The \033[1m silhouette score of Agglomerative Clustering with PCA and Standard Scaler \033[0m is:\033[1m", shc,"\033[0m")  
print("The \033[1m calinski harabasz score of Agglomerative Clustering with PCA and Standard Scaler \033[0m is:\033[1m", chs,"\033[0m")  
print("The \033[1m davies bouldin score of Agglomerative Clustering with PCA and Standard Scaler \033[0m is:\033[1m", dbs,"\033[0m")

The silhouette score of Agglomerative Clustering with PCA and Standard Scaler is: 0.30874273928315893   
The calinski harabasz score of Agglomerative Clustering with PCA and Standard Scaler is: 81.42337491844717   
The davies bouldin score of Agglomerative Clustering with PCA and Standard Scaler is: 1.2531223471706436

accuracy\_score(Y, y\_pred\_agc\_scle\_pca)

0.29775280898876405

#### External Evaluation Metrics for Agglomerative Clustering with PCA and Standard Scaler

vms = v\_measure\_score(Y, y\_pred\_agc\_scle\_pca)  
ars = adjusted\_rand\_score(Y, y\_pred\_agc\_scle\_pca)  
  
print("The \033[1mv measure score of Agglomerative Clustering with PCA and Standard Scaler \033[0m is:\033[1m", vms,"\033[0m")  
print("The \033[1madjusted rand score of Agglomerative Clustering with PCA and Standard Scaler \033[0m is:\033[1m", ars,"\033[0m")

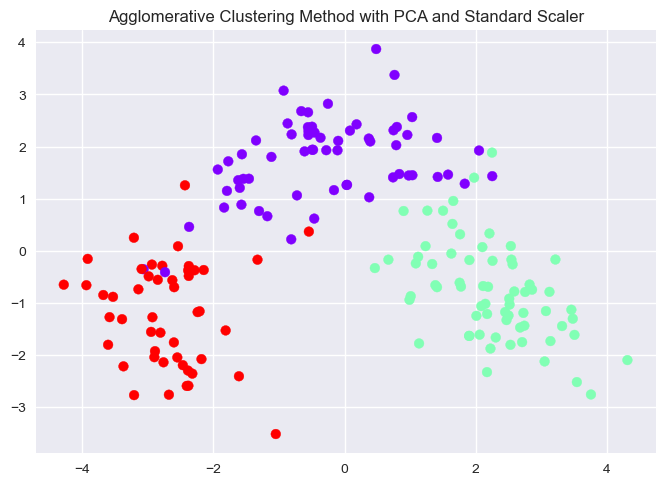
The v measure score of Agglomerative Clustering with PCA and Standard Scaler is: 0.7518426313859891   
The adjusted rand score of Agglomerative Clustering with PCA and Standard Scaler is: 0.7718410789332827

wine = pd.DataFrame(x\_pca)  
wine.columns = ['pca\_1', 'pca\_2', 'pca\_3', 'pca\_4', 'pca\_5', 'pca\_6', 'pca\_7']  
wine['classes'] = Y  
wine.shape

(178, 8)

#[3]  
plt.scatter(x=wine['pca\_1'], y=wine['pca\_2'], c = agc.labels\_, cmap='rainbow')  
plt.title("Agglomerative Clustering Method with PCA and Standard Scaler")

Text(0.5, 1.0, 'Agglomerative Clustering Method with PCA and Standard Scaler')



#### Let's increase the number of clusters to 5 to assess the performance

sclr = StandardScaler()  
pca = PCA(n\_components=7)

x\_pca = pca.fit\_transform(sclr.fit\_transform(X))

agc\_5clusters = AgglomerativeClustering(n\_clusters=5)

y\_pred\_5clusters\_pca\_sclr = agc\_5clusters.fit\_predict(x\_pca, Y)

accuracy\_score(Y, y\_pred\_5clusters\_pca\_sclr)

0.7415730337078652

#### Internal Evaluation Metrics for Agglomerative Clustering with 5 clusters

shc = silhouette\_score(x\_pca, y\_pred\_5clusters\_pca\_sclr)  
  
chs = calinski\_harabasz\_score(x\_pca, y\_pred\_5clusters\_pca\_sclr)  
  
dbs = davies\_bouldin\_score(x\_pca, y\_pred\_agc\_scle\_pca)   
  
print("The \033[1m silhouette score of Agglomerative wih 5 clusters \033[0m is:\033[1m", shc,"\033[0m")  
print("The \033[1m calinski harabasz score of Agglomerative wih 5 clusters \033[0m is:\033[1m", chs,"\033[0m")  
print("The \033[1m davies bouldin score of Agglomerative wih 5 clusters \033[0m is:\033[1m", dbs,"\033[0m")

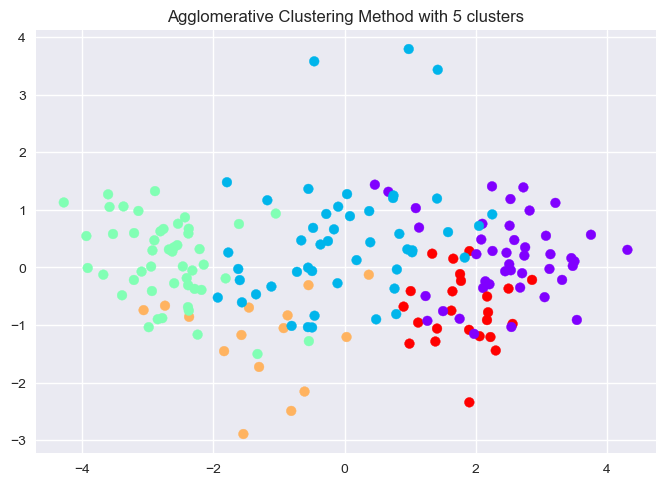
The silhouette score of Agglomerative wih 5 clusters is: 0.19947797979753834   
The calinski harabasz score of Agglomerative wih 5 clusters is: 54.54199506613617   
The davies bouldin score of Agglomerative wih 5 clusters is: 1.2531223471706436

wine = pd.DataFrame(x\_pca)  
wine.columns = ['pca\_1', 'pca\_2', 'pca\_3', 'pca\_4', 'pca\_5', 'pca\_6', 'pca\_7']  
wine['classes'] = Y  
wine.shape

(178, 8)

# [3]  
plt.scatter(x=wine['pca\_1'], y=wine['pca\_4'], c = agc\_5clusters.labels\_, cmap='rainbow')  
plt.title("Agglomerative Clustering Method with 5 clusters")

Text(0.5, 1.0, 'Agglomerative Clustering Method with 5 clusters')



# 3. DBSCAN Clustering Method [5]

dbscan=DBSCAN(eps=40,min\_samples=13)  
y\_pred\_dbscan = dbscan.fit\_predict(X)

#### Internal Evaluation Metrics for DBSCAN

shc = silhouette\_score(X, y\_pred\_dbscan)  
  
chs = calinski\_harabasz\_score(X, y\_pred\_dbscan)  
  
dbs = davies\_bouldin\_score(X, y\_pred\_dbscan)   
  
print("The \033[1m silhouette score of DBSCAN Clustering \033[0m is:\033[1m", shc,"\033[0m")  
print("The \033[1m calinski harabasz score of DBSCAN Clustering \033[0m is:\033[1m", chs,"\033[0m")  
print("The \033[1m davies bouldin score of DBSCAN Clustering \033[0m is:\033[1m", dbs,"\033[0m")

The silhouette score of DBSCAN Clustering is: 0.4131010483149045   
The calinski harabasz score of DBSCAN Clustering is: 178.17422851755185   
The davies bouldin score of DBSCAN Clustering is: 4.167135382259382

#### Extenal Evaluation Metrics for DBSCAN

vms = v\_measure\_score(Y, y\_pred\_dbscan)  
ars = adjusted\_rand\_score(Y, y\_pred\_dbscan)  
  
print("The \033[1mv measure score of DBSCAN Clustering \033[0m is:\033[1m", vms,"\033[0m")  
print("The \033[1madjusted rand score of DBSCAN Clustering \033[0m is:\033[1m", ars,"\033[0m")

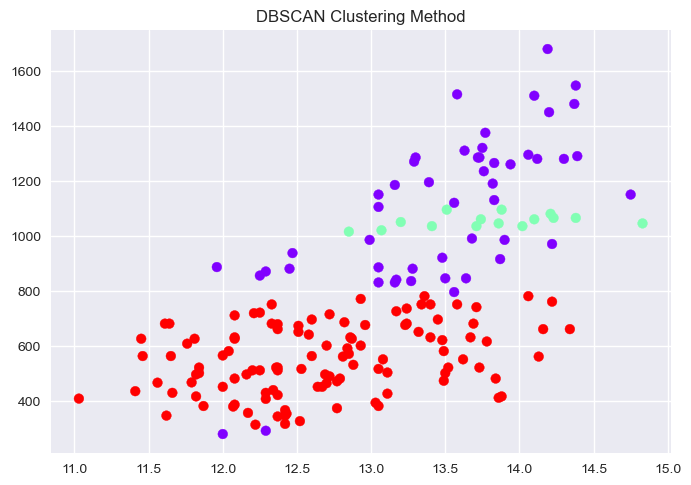
The v measure score of DBSCAN Clustering is: 0.3841182539606525   
The adjusted rand score of DBSCAN Clustering is: 0.3269014234534921

wine = pd.DataFrame(X)  
wine.columns = data.feature\_names  
wine['classes'] = Y  
wine.shape

(178, 14)

# [3]  
plt.scatter(x=wine['alcohol'], y=wine['proline'], c = dbscan.labels\_, cmap='rainbow')  
plt.title("DBSCAN Clustering Method")

Text(0.5, 1.0, 'DBSCAN Clustering Method')



#### DBSCAN with Standard Scaler and TSNE

sclr = StandardScaler()  
tsne = TSNE(n\_components=5, method='exact')  
  
x\_tsne = tsne.fit\_transform(sclr.fit\_transform(X))  
  
dbscan\_tsne=DBSCAN(eps=178,min\_samples=13)  
y\_pred\_dbscan\_tsne = dbscan\_tsne.fit\_predict(x\_tsne)

#### Internal Evaluation Metrics for DBSCAN with Stnadard Scaler and TSNE

shc = silhouette\_score(X, y\_pred\_dbscan\_tsne)  
  
chs = calinski\_harabasz\_score(X, y\_pred\_dbscan\_tsne)  
  
dbs = davies\_bouldin\_score(X, y\_pred\_dbscan\_tsne)   
  
print("The \033[1m silhouette score of DBSCAN with Standard Scaler and TSNE \033[0m is:\033[1m", shc,"\033[0m")  
print("The \033[1m calinski harabasz score of DBSCAN with Standard Scaler and TSNE \033[0m is:\033[1m", chs,"\033[0m")  
print("The \033[1m davies bouldin score of DBSCAN with Standard Scaler and TSNE \033[0m is:\033[1m", dbs,"\033[0m")

The silhouette score of DBSCAN with Standard Scaler and TSNE is: -0.12934132684947203   
The calinski harabasz score of DBSCAN with Standard Scaler and TSNE is: 0.024214668035690636   
The davies bouldin score of DBSCAN with Standard Scaler and TSNE is: 16.579548290636957

#### External Evaluation Metrics for DBSCAN with Stnadard Scaler and TSNE

vms = v\_measure\_score(Y, y\_pred\_dbscan)  
ars = adjusted\_rand\_score(Y, y\_pred\_dbscan)  
  
print("The \033[1mv measure score of DBSCAN with Standard Scaler and TSNE \033[0m is:\033[1m", vms,"\033[0m")  
print("The \033[1madjusted rand score of DBSCAN with Standard Scaler and TSNE \033[0m is:\033[1m", ars,"\033[0m")

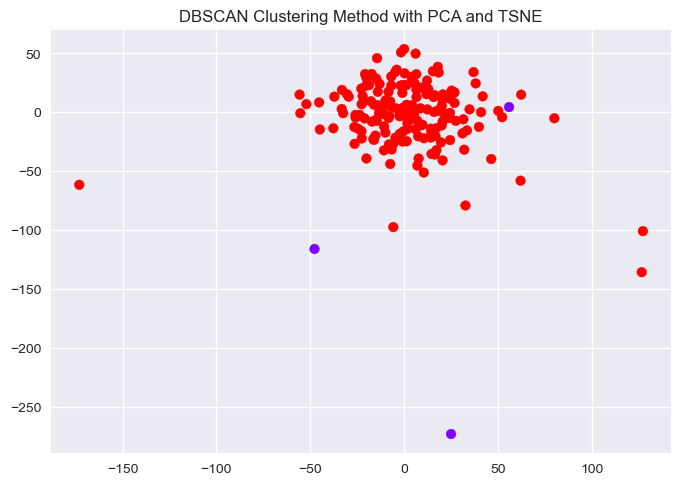
The v measure score of DBSCAN with Standard Scaler and TSNE is: 0.3841182539606525   
The adjusted rand score of DBSCAN with Standard Scaler and TSNE is: 0.3269014234534921

wine = pd.DataFrame(x\_tsne)  
wine.columns = ['tsne\_1', 'tsne\_2', 'tsne\_3', 'tsne\_4', 'tsne\_5']  
wine['classes'] = Y  
wine.shape

(178, 6)

# [3]  
plt.scatter(x=wine['tsne\_1'], y=wine['tsne\_2'], c = dbscan\_tsne.labels\_, cmap='rainbow')  
plt.title("DBSCAN Clustering Method with PCA and TSNE")

Text(0.5, 1.0, 'DBSCAN Clustering Method with PCA and TSNE')



# 4. BIRCH Clustering Method [6]

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

bch = Birch(n\_clusters=3)

bch.fit(x\_train,y\_train)

Birch()

y\_pred\_bch = bch.predict(x\_test)

#### Internal Evaluation Metrics for BIRCH

shc = silhouette\_score(x\_test, y\_pred\_bch)  
  
chs = calinski\_harabasz\_score(x\_test, y\_pred\_bch)  
  
dbs = davies\_bouldin\_score(x\_test, y\_pred\_bch)  
  
print("The \033[1m silhouette score of BIRCH Clustering \033[0m is:\033[1m", shc,"\033[0m")  
print("The \033[1m calinski harabasz score of BIRCH Clustering \033[0m is:\033[1m", chs,"\033[0m")  
print("The \033[1m davies bouldin score of BIRCH Clustering \033[0m is:\033[1m", dbs,"\033[0m")

The silhouette score of BIRCH Clustering is: 0.5334632414925647   
The calinski harabasz score of BIRCH Clustering is: 94.05241922023522   
The davies bouldin score of BIRCH Clustering is: 0.5866562445301988

#### External Evaluation Metrics for BIRCH

vms = v\_measure\_score(y\_test, y\_pred\_bch)  
ars = adjusted\_rand\_score(y\_test, y\_pred\_bch)  
  
print("The \033[1mv measure score of BIRCH Clustering \033[0m is:\033[1m", vms,"\033[0m")  
print("The \033[1madjusted rand score of BIRCH Clustering \033[0m is:\033[1m", ars,"\033[0m")

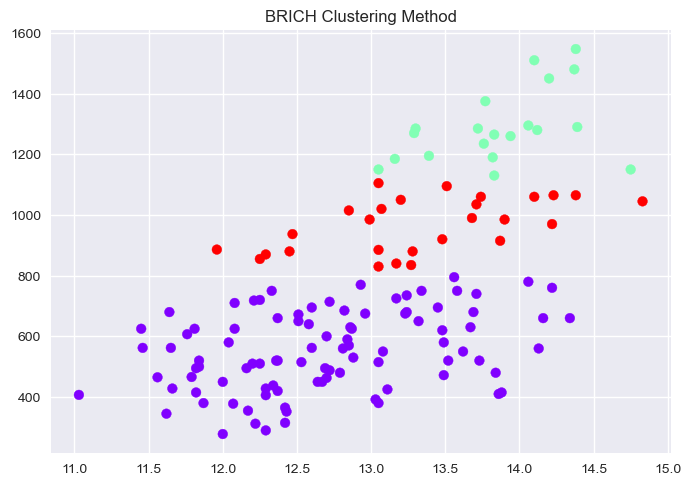
The v measure score of BIRCH Clustering is: 0.6029711140174827   
The adjusted rand score of BIRCH Clustering is: 0.4566929133858268

wine\_bch = pd.DataFrame(X)  
wine\_bch.columns = data.feature\_names  
wine\_bch['classes'] = Y  
wine\_bch.shape

(178, 14)

# [3]  
plt.scatter(x=x\_train[:,0], y=x\_train[:,12], c = bch.labels\_, cmap='rainbow')  
plt.title("BRICH Clustering Method")

Text(0.5, 1.0, 'BRICH Clustering Method')



#### BIRCH with Standard scaler and PCA

sclr = StandardScaler()  
pca = PCA(n\_components=7)

x\_pca = pca.fit\_transform(sclr.fit\_transform(X))

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_pca, Y, test\_size=0.2, random\_state=42)

bch = Birch(n\_clusters=3)

bch.fit(x\_train,y\_train)

Birch()

y\_pred\_bch\_pca\_sclr = bch.predict(x\_test)

#### Internal Evaluation Metrics for BIRCH with Standard Scaler and PCA

shc = silhouette\_score(x\_test, y\_pred\_bch\_pca\_sclr)  
  
chs = calinski\_harabasz\_score(x\_test, y\_pred\_bch\_pca\_sclr)  
  
dbs = davies\_bouldin\_score(x\_test, y\_pred\_bch\_pca\_sclr)  
  
print("The \033[1m silhouette score of BIRCH with Standard Scaler and PCA \033[0m is:\033[1m", shc,"\033[0m")  
print("The \033[1m calinski harabasz score of BIRCH with Standard Scaler and PCA \033[0m is:\033[1m", chs,"\033[0m")  
print("The \033[1m davies bouldin score of BIRCH with Standard Scaler and PCA \033[0m is:\033[1m", dbs,"\033[0m")

The silhouette score of BIRCH with Standard Scaler and PCA is: 0.37265786008711865   
The calinski harabasz score of BIRCH with Standard Scaler and PCA is: 23.00445039465537   
The davies bouldin score of BIRCH with Standard Scaler and PCA is: 1.0117866290118738

#### External Evaluation Metrics for BIRCH with Standard Scaler and PCA

vms = v\_measure\_score(y\_test, y\_pred\_bch\_pca\_sclr)  
ars = adjusted\_rand\_score(y\_test, y\_pred\_bch\_pca\_sclr)  
  
print("The \033[1mv measure score of BIRCH with Standard Scaler and PCA \033[0m is:\033[1m", vms,"\033[0m")  
print("The \033[1madjusted rand score of BIRCH with Standard Scaler and PCA \033[0m is:\033[1m", ars,"\033[0m")

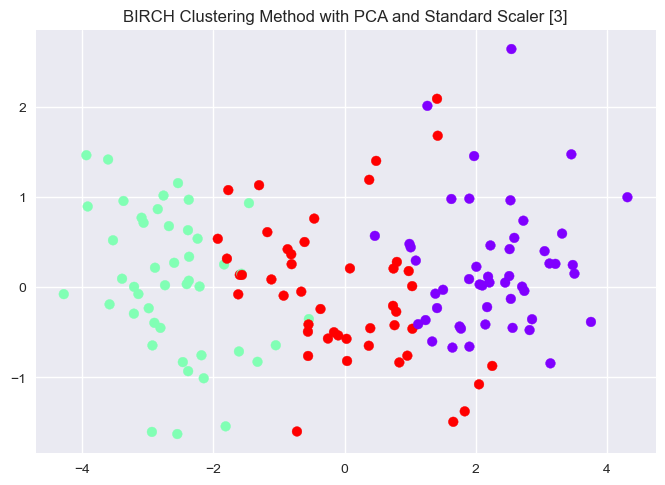
The v measure score of BIRCH with Standard Scaler and PCA is: 0.8195750057686682   
The adjusted rand score of BIRCH with Standard Scaler and PCA is: 0.8315412186379928

wine\_bch = pd.DataFrame(x\_pca)  
wine\_bch.columns = ['pca\_1', 'pca\_2', 'pca\_3', 'pca\_4', 'pca\_5', 'pca\_6', 'pca\_7']  
wine\_bch['classes'] = Y  
wine\_bch.shape

(178, 8)

plt.scatter(x=x\_train[:,0], y=x\_train[:,6], c = bch.labels\_, cmap='rainbow')  
plt.title("BIRCH Clustering Method with PCA and Standard Scaler [3]")

Text(0.5, 1.0, 'BIRCH Clustering Method with PCA and Standard Scaler [3]')



# 5. Affinity Propagation Clustering Method [7]

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

afp = AffinityPropagation(damping=0.89,random\_state=3)

afp.fit(x\_train, y\_train)

AffinityPropagation(damping=0.89, random\_state=3)

y\_pred\_afp = afp.predict(x\_test)

#### Internal Evaluation Metrics for Affinity Propogation Clustering

shc = silhouette\_score(x\_test,y\_pred\_afp)  
  
chs = calinski\_harabasz\_score(x\_test, y\_pred\_afp)  
  
dbs = davies\_bouldin\_score(x\_test, y\_pred\_afp)   
  
print("The \033[1m silhouette score of Affinity Propogation Clustering \033[0m is:\033[1m", shc,"\033[0m")  
print("The \033[1m calinski harabasz score of Affinity Propogation Clustering \033[0m is:\033[1m", chs,"\033[0m")  
print("The \033[1m davies bouldin score of Affinity Propogation Clustering \033[0m is:\033[1m", dbs,"\033[0m")

The silhouette score of Affinity Propogation Clustering is: 0.6358074966565659   
The calinski harabasz score of Affinity Propogation Clustering is: 393.82549674130735   
The davies bouldin score of Affinity Propogation Clustering is: 0.3790558389837589

#### External Evaluation Metrics for Affinity Propogation Clustering

vms = v\_measure\_score(y\_test, y\_pred\_afp)  
ars = adjusted\_rand\_score(y\_test, y\_pred\_afp)  
  
print("The \033[1mv measure score of Affinity Propogation Clustering \033[0m is:\033[1m", vms,"\033[0m")  
print("The \033[1madjusted rand score of Affinity Propogation Clustering \033[0m is:\033[1m", ars,"\033[0m")

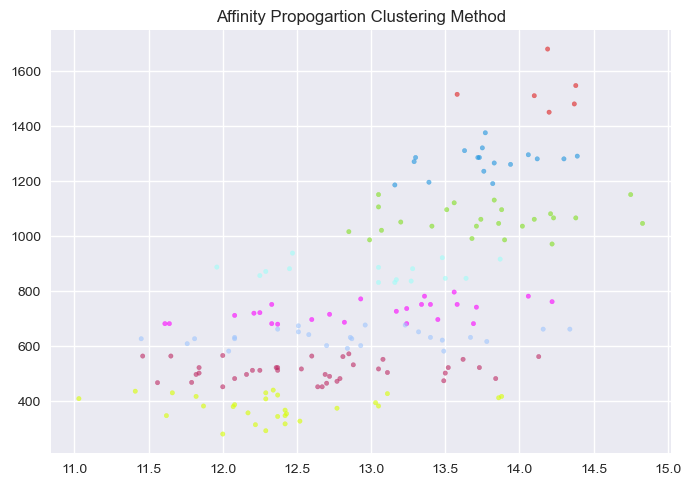
The v measure score of Affinity Propogation Clustering is: 0.5085304530677577   
The adjusted rand score of Affinity Propogation Clustering is: 0.24893917963224893

wine\_afp = pd.DataFrame(X)  
wine\_afp.columns = data.feature\_names  
wine\_afp['classes'] = Y  
wine\_afp.shape

(178, 14)

# [4]  
wine\_afp['cluster']=afp.fit\_predict(X,Y)  
centroids = afp.cluster\_centers\_  
cen\_x = [i[0] for i in centroids]   
cen\_y = [i[1] for i in centroids]  
## add to df  
wine\_afp['cen\_x'] = wine\_afp.cluster.map({0:cen\_x[0], 1:cen\_x[1], 2:cen\_x[2], 3:cen\_x[3], 4:cen\_x[4], 5:cen\_x[5], 6:cen\_x[6], 7:cen\_x[7]})  
wine\_afp['cen\_y'] = wine\_afp.cluster.map({0:cen\_y[0], 1:cen\_y[1], 2:cen\_y[2], 4:cen\_y[4], 4:cen\_y[4], 5:cen\_y[5], 6:cen\_y[6], 7:cen\_y[7]})  
# define and map colors  
colors = ['#DF2020', '#81DF20', '#2095DF', '#A0C6FE', '#A0FEF8', '#D8FF00', '#FB00FF', '#BE2A64']  
wine\_afp['c'] = wine\_afp.cluster.map({0:colors[0], 1:colors[1], 2:colors[2], 3:colors[3], 4:colors[4], 5:colors[5], 6:colors[6], 7:colors[7]})  
  
plt.scatter(wine\_afp['alcohol'], wine\_afp['proline'], c=wine\_afp.c, alpha = 0.6, s=10)  
plt.title("Affinity Propogartion Clustering Method")

Text(0.5, 1.0, 'Affinity Propogartion Clustering Method')



#### Affinity Propagation with Standard Scaler and PCA

sclr = StandardScaler()  
pca = PCA(n\_components=7)  
x\_pca = pca.fit\_transform(sclr.fit\_transform(X))

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_pca, Y, test\_size=0.2, random\_state=42)

afp\_pca\_sclr = AffinityPropagation(damping=0.89,random\_state=3)

afp\_pca\_sclr.fit(x\_train, y\_train)

AffinityPropagation(damping=0.89, random\_state=3)

y\_pred\_afp\_pca\_sclr = afp\_pca\_sclr.predict(x\_test)

#### Internal Evaluation Metrics for Affinity Propogation Clustering with Standard Scaler and PCA

shc = silhouette\_score(x\_test, y\_pred\_afp\_pca\_sclr)  
  
chs = calinski\_harabasz\_score(x\_test, y\_pred\_afp\_pca\_sclr)  
  
dbs = davies\_bouldin\_score(x\_test, y\_pred\_afp\_pca\_sclr)   
  
print("The \033[1m silhouette score of Affinity Propogation with Standard Scaler and PCA \033[0m is:\033[1m", shc,"\033[0m")  
print("The \033[1m calinski harabasz score of Affinity Propogation with Standard Scaler and PCA \033[0m is:\033[1m", chs,"\033[0m")  
print("The \033[1m davies bouldin score of Affinity Propogation with Standard Scaler and PCA \033[0m is:\033[1m", dbs,"\033[0m")

The silhouette score of Affinity Propogation with Standard Scaler and PCA is: 0.06771171807462599   
The calinski harabasz score of Affinity Propogation with Standard Scaler and PCA is: 10.238760685546751   
The davies bouldin score of Affinity Propogation with Standard Scaler and PCA is: 1.878861972802326

#### External Evaluation Metrics for Affinity Propogation Clustering with Standard Scaler and PCA

vms = v\_measure\_score(y\_test, y\_pred\_afp\_pca\_sclr)  
ars = adjusted\_rand\_score(y\_test, y\_pred\_afp\_pca\_sclr)  
  
print("The \033[1mv measure score of Affinity Propogation with Standard Scaler and PCA \033[0m is:\033[1m", vms,"\033[0m")  
print("The \033[1madjusted rand score of Affinity Propogation with Standard Scaler and PCA \033[0m is:\033[1m", ars,"\033[0m")

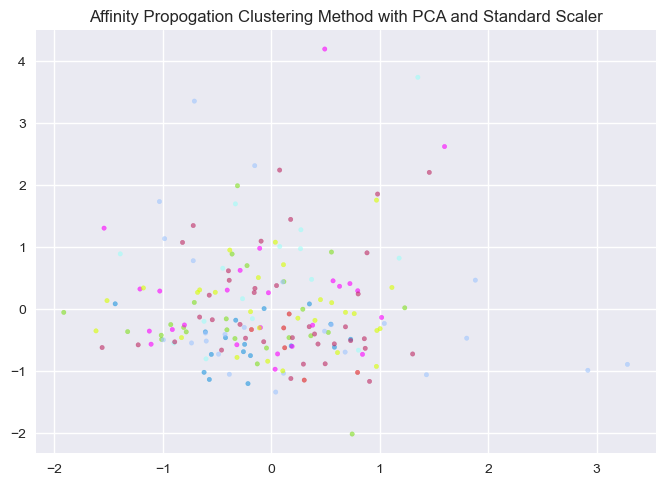
The v measure score of Affinity Propogation with Standard Scaler and PCA is: 0.6149572563012716   
The adjusted rand score of Affinity Propogation with Standard Scaler and PCA is: 0.3693181818181818

wine\_pca\_afp\_sclr = pd.DataFrame(x\_pca)  
wine\_pca\_afp\_sclr.columns = ['pca\_1', 'pca\_2', 'pca\_3', 'pca\_4', 'pca\_5', 'pca\_6', 'pca\_7']  
wine\_pca\_afp\_sclr['classes'] = Y  
wine\_pca\_afp\_sclr.shape

(178, 8)

# [4]  
wine\_pca\_afp\_sclr['cluster']=afp.fit\_predict(X,Y)  
centroids = afp.cluster\_centers\_  
cen\_x = [i[0] for i in centroids]   
cen\_y = [i[1] for i in centroids]  
## add to df  
wine\_pca\_afp\_sclr['cen\_x'] = wine\_pca\_afp\_sclr.cluster.map({0:cen\_x[0], 1:cen\_x[1], 2:cen\_x[2], 3:cen\_x[3], 4:cen\_x[4], 5:cen\_x[5], 6:cen\_x[6], 7:cen\_x[7]})  
wine\_pca\_afp\_sclr['cen\_y'] = wine\_pca\_afp\_sclr.cluster.map({0:cen\_y[0], 1:cen\_y[1], 2:cen\_y[2], 4:cen\_y[4], 4:cen\_y[4], 5:cen\_y[5], 6:cen\_y[6], 7:cen\_y[7]})  
# define and map colors  
colors = ['#DF2020', '#81DF20', '#2095DF', '#A0C6FE', '#A0FEF8', '#D8FF00', '#FB00FF', '#BE2A64']  
wine\_pca\_afp\_sclr['c'] = wine\_pca\_afp\_sclr.cluster.map({0:colors[0], 1:colors[1], 2:colors[2], 3:colors[3], 4:colors[4], 5:colors[5], 6:colors[6], 7:colors[7]})  
  
plt.scatter(wine\_pca\_afp\_sclr['pca\_6'], wine\_pca\_afp\_sclr['pca\_5'], c=wine\_pca\_afp\_sclr.c, alpha = 0.6, s=10)  
plt.title("Affinity Propogation Clustering Method with PCA and Standard Scaler")

Text(0.5, 1.0, 'Affinity Propogation Clustering Method with PCA and Standard Scaler')



# 6. Mean-Shift Clustering Method [8]

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

bandwidth = estimate\_bandwidth(x\_train, quantile = 0.2, n\_samples=178)  
ms\_cls = MeanShift(bandwidth = bandwidth)  
ms\_cls.fit(x\_train, y\_train)

MeanShift(bandwidth=120.59727484964449)

y\_pred\_ms = ms\_cls.predict(x\_test)

#### Internal Evaluation Metrics for Mean Shift Clustering

ms\_shc = silhouette\_score(x\_test, y\_pred\_ms)  
  
ms\_chs = calinski\_harabasz\_score(x\_test, y\_pred\_ms)  
  
ms\_dbs = davies\_bouldin\_score(x\_test, y\_pred\_ms)   
  
print("The \033[1m silhouette score of Mean Shift Clustering \033[0m is:\033[1m", shc,"\033[0m")  
print("The \033[1m calinski harabasz score of Mean Shift Clustering \033[0m is:\033[1m", chs,"\033[0m")  
print("The \033[1m davies bouldin score of Mean Shift Clustering \033[0m is:\033[1m", dbs,"\033[0m")

The silhouette score of Mean Shift Clustering is: 0.06771171807462599   
The calinski harabasz score of Mean Shift Clustering is: 10.238760685546751   
The davies bouldin score of Mean Shift Clustering is: 1.878861972802326

#### External Evaluation Metrics for Mean Shift Clustering

ms\_vms = v\_measure\_score(y\_test, y\_pred\_ms)  
ms\_ars = adjusted\_rand\_score(y\_test, y\_pred\_ms)  
  
print("The \033[1mv measure score of Mean Shift Clustering \033[0m is:\033[1m", vms,"\033[0m")  
print("The \033[1madjusted rand score of Mean Shift Clustering \033[0m is:\033[1m", ars,"\033[0m")

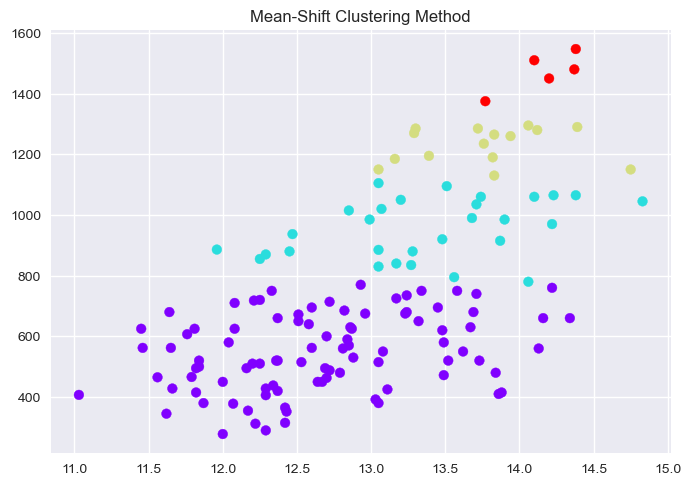
The v measure score of Mean Shift Clustering is: 0.6149572563012716   
The adjusted rand score of Mean Shift Clustering is: 0.3693181818181818

wine = pd.DataFrame(X)  
wine.columns = data.feature\_names  
wine['classes'] = Y  
wine.shape

(178, 14)

# [3]  
plt.scatter(x=x\_train[:,0], y=x\_train[:,12], c = ms\_cls.labels\_, cmap='rainbow')  
plt.title("Mean-Shift Clustering Method")

Text(0.5, 1.0, 'Mean-Shift Clustering Method')



# Part III: Classification

### Classification methods used

1) Logistic Regression

2) Gaussian Naive Bayes [9]

3) Decision Trees

4) Random Forest

5) Support Vector Machines

6) K Nearest Neighbours [10]

7) Ada Boost Classifier [11]

# Classification of Wine dataset using Classification methods offered by sklearn

#### The required imports for classification

import matplotlib.pyplot as plt  
import numpy as np  
from numpy import mean, std  
from sklearn.linear\_model import LogisticRegression  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, AdaBoostClassifier  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.svm import SVC  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.model\_selection import KFold, cross\_val\_score, train\_test\_split  
from sklearn.metrics import classification\_report, confusion\_matrix,ConfusionMatrixDisplay, balanced\_accuracy\_score, roc\_auc\_score, roc\_curve   
from sklearn.preprocessing import StandardScaler, Normalizer  
from sklearn.decomposition import PCA  
  
## use the inbuilt library to load the data  
from sklearn.datasets import load\_wine

#### Load the data into X(features) and Y(classes) variables

wine = load\_wine()  
  
X = wine.data  
  
Y = wine.target

#### K fold cross-validation which will be used for all the methods [1]

cv = KFold(n\_splits = 8, shuffle = True, random\_state = 42)

# Logistic Regression Classificaion Method

#### Logistic Regression with full data and then cross validated with 8 folds and the data is split as 67% for training and 33% as testing

## Split data protocol

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.33, random\_state=42)

clf = LogisticRegression(max\_iter=10000)

clf.fit(x\_train, y\_train)

LogisticRegression(max\_iter=10000)

#### Cross validate and measure the scores and calculate the mean accuracy of the model

scores = cross\_val\_score(clf, x\_train, y\_train, cv=cv, scoring='accuracy')

print("The mean accuracy obtained using Logistic Regression are:\n",scores, scores.mean())

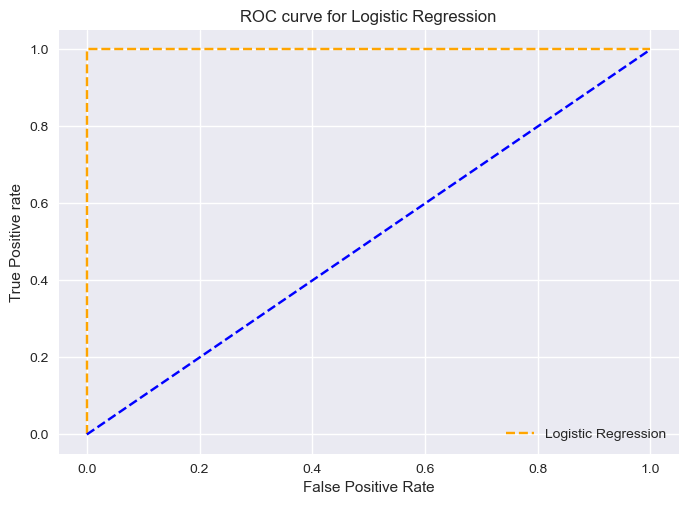
The mean accuracy obtained using Logistic Regression are:  
 [0.8 1. 1. 0.86666667 0.86666667 1.  
 1. 0.92857143] 0.9327380952380953

y\_pred = clf.predict(x\_test)

#### Plot the ROC-AUC curve by calculating the TPR and FPR for the particular model [2]

# [2]  
probs = clf.predict\_proba(x\_test)  
  
fpr1\_full, tpr1\_full, thresh1 = roc\_curve(y\_test, probs[:,1], pos\_label=1)  
# roc curve for tpr = fpr   
random\_probs = [0 for i in range(len(y\_test))]  
p\_fpr\_full, p\_tpr\_full, \_ = roc\_curve(y\_test, random\_probs, pos\_label=1)  
  
##matplotlib  
import matplotlib.pyplot as plt  
plt.style.use('seaborn')  
  
# plot roc curves  
plt.plot(fpr1\_full, tpr1\_full, linestyle='--',color='orange', label='Logistic Regression')  
plt.plot(p\_fpr\_full, p\_tpr\_full, linestyle='--', color='blue')  
#done  
# title  
plt.title('ROC curve for Logistic Regression')  
# x label  
plt.xlabel('False Positive Rate')  
# y label  
plt.ylabel('True Positive rate')  
plt.legend(loc='best')

<matplotlib.legend.Legend at 0x217294e8130>



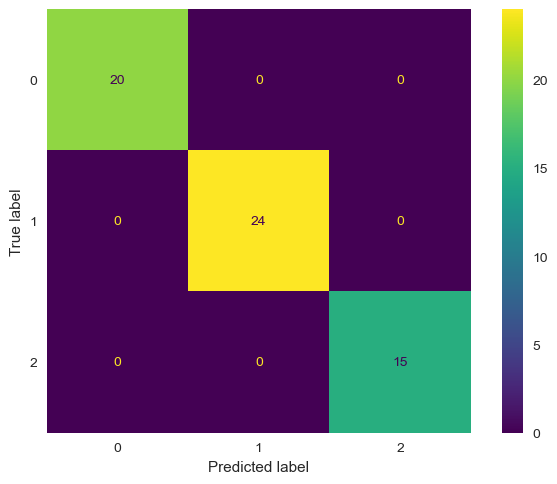
#### ROC AUC score

print('the ROC-AUC score of Logistic Regression', roc\_auc\_score(y\_test, clf.predict\_proba(x\_test), multi\_class='ovr'))

the ROC-AUC score of Logistic Regression 1.0

cm=confusion\_matrix(y\_test,y\_pred,labels=clf.classes\_)  
disp = ConfusionMatrixDisplay(confusion\_matrix=cm,display\_labels=clf.classes\_)  
print(' \033[1mConfusion Matrix of Logistic Regression\033[0m')  
disp.plot()  
plt.grid(False)  
plt.show()

Confusion Matrix of Logistic Regression



#### Plot the confusion matrix

#### To display the classification report

print("\033[1mThe classification report of Logistic Regression is:\n------------------Classification Report------------------\033[0m \n",classification\_report(y\_test,y\_pred))

The classification report of Logistic Regression is:  
------------------Classification Report------------------   
 precision recall f1-score support  
  
 0 1.00 1.00 1.00 20  
 1 1.00 1.00 1.00 24  
 2 1.00 1.00 1.00 15  
  
 accuracy 1.00 59  
 macro avg 1.00 1.00 1.00 59  
weighted avg 1.00 1.00 1.00 59

print("The \033[1m Balanced accuracy score for Logistic Regression \033[0m is:\n \033[0m",balanced\_accuracy\_score(y\_test, y\_pred),"\033[0m")

The Balanced accuracy score for Logistic Regression is:  
 1.0

### Classification without KFold

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.33, random\_state=42)

clf.fit(x\_train, y\_train)

LogisticRegression(max\_iter=10000)

y\_pred = clf.predict(x\_test)

y\_test, y\_pred

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

from sklearn.metrics import accuracy\_score

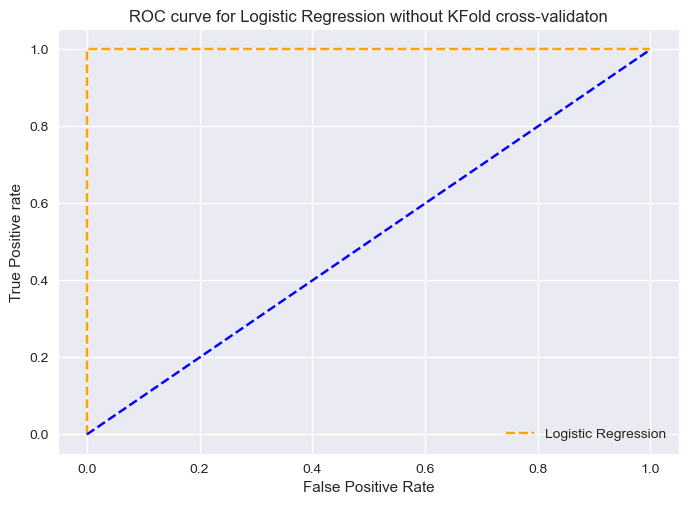
accuracy = accuracy\_score(y\_test, y\_pred)  
print("The accuracy score for Logistic Regression without KFold cross-validation is:", accuracy)

The accuracy score for Logistic Regression without KFold cross-validation is: 1.0

#### Let's plot the ROC-AUC curves, display the confusion matrix and print the classification report which displays the Precision, recall and the F1-score to evaluate the model. The subsequent cells represent the balanced accuracy score for the particular model [2]

# [2]  
probs = clf.predict\_proba(x\_test)  
  
fpr1, tpr1, thresh1 = roc\_curve(y\_test, probs[:,1], pos\_label=1)  
# roc curve for tpr = fpr   
random\_probs = [0 for i in range(len(y\_test))]  
p\_fpr, p\_tpr, \_ = roc\_curve(y\_test, random\_probs, pos\_label=1)  
  
##matplotlib  
import matplotlib.pyplot as plt  
plt.style.use('seaborn')  
  
# plot roc curves  
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='Logistic Regression')  
plt.plot(p\_fpr, p\_tpr, linestyle='--', color='blue')  
# title  
plt.title('ROC curve for Logistic Regression without KFold cross-validaton')  
# x label  
plt.xlabel('False Positive Rate')  
# y label  
plt.ylabel('True Positive rate')  
plt.legend(loc='best')

<matplotlib.legend.Legend at 0x2172997bfa0>



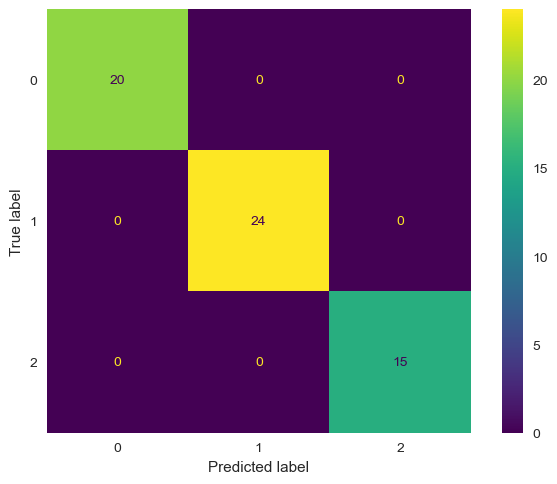
#### ROC AUC score

print('the ROC-AUC score of Logistic Regression without k fold', roc\_auc\_score(y\_test, clf.predict\_proba(x\_test), multi\_class='ovr'))

the ROC-AUC score of Logistic Regression without k fold 1.0

cm=confusion\_matrix(y\_test,y\_pred,labels=clf.classes\_)  
disp = ConfusionMatrixDisplay(confusion\_matrix=cm,display\_labels=clf.classes\_)  
print(' \033[1mConfusion Matrix of Logistic Regression without KFold cross-validation\033[0m')  
disp.plot()  
plt.grid(False)  
plt.show()

Confusion Matrix of Logistic Regression without KFold cross-validation



print("\033[1mThe classification report of Logistic Regression without KFold is:\n------------------Classification Report------------------\033[0m \n",classification\_report(y\_test,y\_pred))

The classification report of Logistic Regression without KFold is:  
------------------Classification Report------------------   
 precision recall f1-score support  
  
 0 1.00 1.00 1.00 20  
 1 1.00 1.00 1.00 24  
 2 1.00 1.00 1.00 15  
  
 accuracy 1.00 59  
 macro avg 1.00 1.00 1.00 59  
weighted avg 1.00 1.00 1.00 59

print("\033[1mThe Balanced accuracy score of Logistic Regression without KFold \033[0m is: \033[1m", balanced\_accuracy\_score(y\_test, y\_pred))

The Balanced accuracy score of Logistic Regression without KFold is: 1.0

## Let's take Gaussian Naive Bayes [9] to keep a reference model, which takes all the features to be independent. This will help us in understanding if the classification algorithms need dimensionality reduction

gnb = GaussianNB(var\_smoothing=0.001)

gnb.fit(x\_train, y\_train)

GaussianNB(var\_smoothing=0.001)

y\_pred\_gnb = gnb.predict(x\_test)

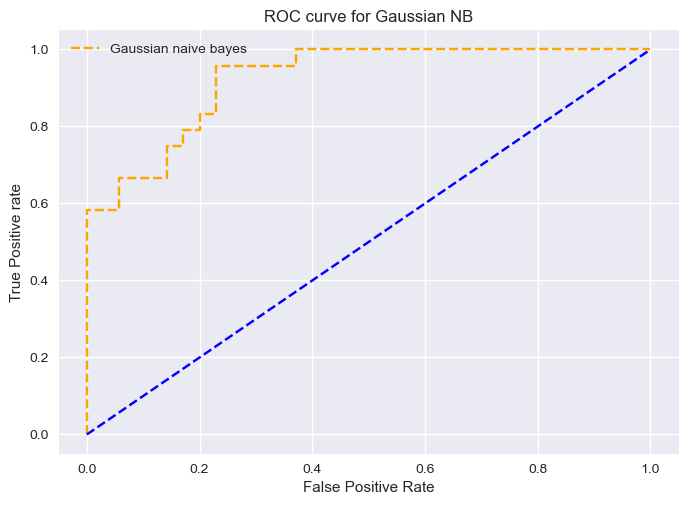
print("The accuracy score when Gaussian Naive Bayes is taken as a reference model is:",accuracy\_score(y\_test, y\_pred\_gnb))

The accuracy score when Gaussian Naive Bayes is taken as a reference model is: 0.7966101694915254

#### Let's plot the ROC-AUC curves, display the confusion matrix and print the classification report which displays the Precision, recall and the F1-score to evaluate the model. The subsequent cells represent the balanced accuracy score for the particular model [2]

# [2]  
probs = gnb.predict\_proba(x\_test)  
  
fpr1\_gnb, tpr1\_gnb, thresh1\_gnb = roc\_curve(y\_test, probs[:,1], pos\_label=1)  
# roc curve for tpr = fpr   
random\_probs = [0 for i in range(len(y\_test))]  
p\_fpr\_gnb, p\_tpr\_gnb, \_ = roc\_curve(y\_test, random\_probs, pos\_label=1)  
  
##matplotlib  
import matplotlib.pyplot as plt  
plt.style.use('seaborn')  
  
# plot roc curves  
plt.plot(fpr1\_gnb, tpr1\_gnb, linestyle='--',color='orange', label='Gaussian naive bayes')  
plt.plot(p\_fpr\_gnb, p\_tpr\_gnb, linestyle='--', color='blue')  
# title  
plt.title('ROC curve for Gaussian NB')  
# x label  
plt.xlabel('False Positive Rate')  
# y label  
plt.ylabel('True Positive rate')  
plt.legend(loc='best')

<matplotlib.legend.Legend at 0x2172996eaf0>



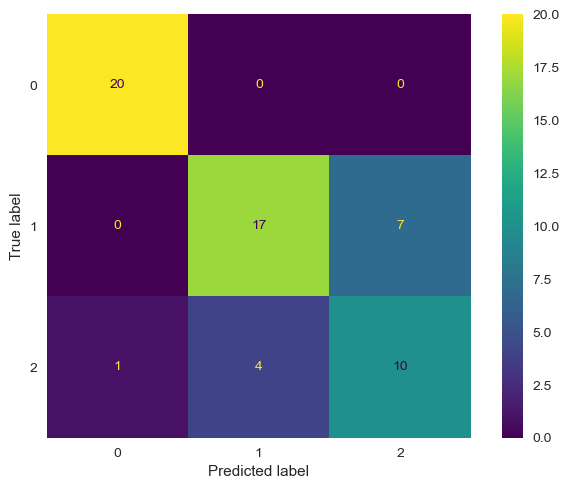
#### ROC AUC score

print('the ROC-AUC score of Gaussian Naive Bayes', roc\_auc\_score(y\_test, gnb.predict\_proba(x\_test), multi\_class='ovr'))

the ROC-AUC score of Gaussian Naive Bayes 0.9463203463203463

cm=confusion\_matrix(y\_test,y\_pred\_gnb,labels=gnb.classes\_)  
disp = ConfusionMatrixDisplay(confusion\_matrix=cm,display\_labels=gnb.classes\_)  
print(' \033[1mConfusion Matrix of Gaussian NB\033[0m')  
disp.plot()  
plt.grid(False)  
plt.show()

Confusion Matrix of Gaussian NB



print("\033[1mThe classification report when Gaussian Naive Bayes is taken as a reference model is:\n\n------------------Classification Report------------------\033[0m \n",classification\_report(y\_test, y\_pred\_gnb))

The classification report when Gaussian Naive Bayes is taken as a reference model is:  
  
------------------Classification Report------------------   
 precision recall f1-score support  
  
 0 0.95 1.00 0.98 20  
 1 0.81 0.71 0.76 24  
 2 0.59 0.67 0.62 15  
  
 accuracy 0.80 59  
 macro avg 0.78 0.79 0.79 59  
weighted avg 0.80 0.80 0.80 59

print("\033[1mThe Balanced accuracy score when Gaussian Naive Bayes is taken as a reference model\033[0m is:\033[1m", balanced\_accuracy\_score(y\_test, y\_pred\_gnb))

The Balanced accuracy score when Gaussian Naive Bayes is taken as a reference model is: 0.7916666666666666

### Logistic regression with PCA

pca = PCA(n\_components = 7, svd\_solver='randomized')

x\_pca = pca.fit\_transform(X)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_pca, Y, test\_size=0.33, random\_state=42)

clf.fit(x\_train, y\_train)

LogisticRegression(max\_iter=10000)

y\_pred\_lr\_pca = clf.predict(x\_test)

#### Cross validate and measure the scores and calculate the mean accuracy of the model

score = cross\_val\_score(clf, x\_train, y\_train, scoring='accuracy', cv = cv)

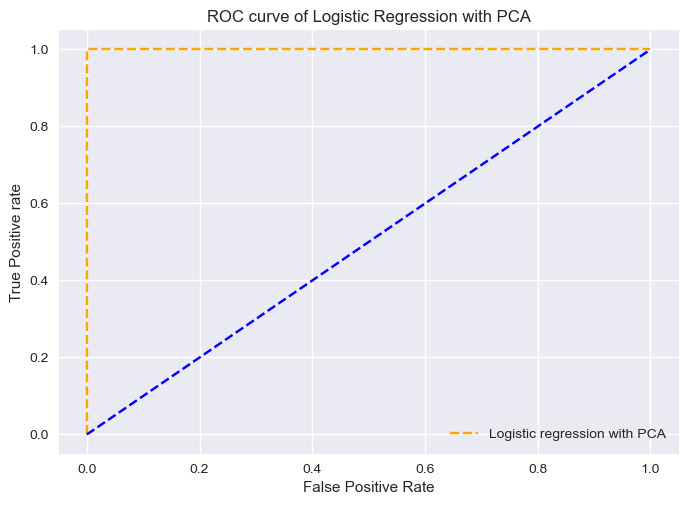
print("The mean accuracy obtained using Logistic Regression with PCA is",score.mean())

The mean accuracy obtained using Logistic Regression with PCA is 0.924404761904762

#### Let's plot the ROC-AUC curves, display the confusion matrix and print the classification report which displays the Precision, recall and the F1-score to evaluate the model. The subsequent cells represent the balanced accuracy score for the particular model [2]

# [2]  
probs = clf.predict\_proba(x\_test)  
  
fpr1\_lr\_pca, tpr1\_lr\_pca, thresh1\_gnb = roc\_curve(y\_test, probs[:,1], pos\_label=1)  
# roc curve for tpr = fpr   
random\_probs = [0 for i in range(len(y\_test))]  
p\_fpr\_lr\_pca, p\_tpr\_lr\_pca, \_ = roc\_curve(y\_test, random\_probs, pos\_label=1)  
  
##matplotlib  
import matplotlib.pyplot as plt  
plt.style.use('seaborn')  
  
# plot roc curves  
plt.plot(fpr1\_lr\_pca, tpr1\_lr\_pca, linestyle='--',color='orange', label='Logistic regression with PCA')  
plt.plot(p\_fpr\_lr\_pca, p\_tpr\_lr\_pca, linestyle='--', color='blue')  
# title  
plt.title('ROC curve of Logistic Regression with PCA')  
# x label  
plt.xlabel('False Positive Rate')  
# y label  
plt.ylabel('True Positive rate')  
plt.legend(loc='best')

<matplotlib.legend.Legend at 0x2172a71bbb0>



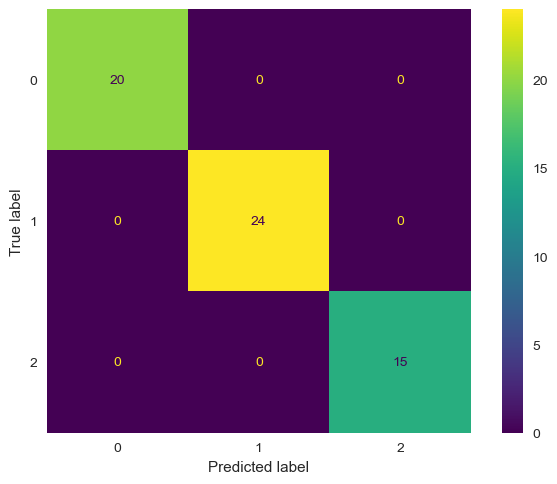
#### ROC AUC score

print('the ROC-AUC score of Logistic Regression with PCA', roc\_auc\_score(y\_test, clf.predict\_proba(x\_test), multi\_class='ovr'))

the ROC-AUC score of Logistic Regression with PCA 1.0

cm=confusion\_matrix(y\_test,y\_pred\_lr\_pca,labels=clf.classes\_)  
disp = ConfusionMatrixDisplay(confusion\_matrix=cm,display\_labels=clf.classes\_)  
print(' \033[1mConfusion Matrix of Logistic Regression with PCA\033[0m')  
disp.plot()  
plt.grid(False)  
plt.show()

Confusion Matrix of Logistic Regression with PCA



print("\033[1mThe classification report of Logistic Regression with PCA is:\n\n------------------Classification Report------------------\033[0m \n",classification\_report(y\_test, y\_pred\_lr\_pca))

The classification report of Logistic Regression with PCA is:  
  
------------------Classification Report------------------   
 precision recall f1-score support  
  
 0 1.00 1.00 1.00 20  
 1 1.00 1.00 1.00 24  
 2 1.00 1.00 1.00 15  
  
 accuracy 1.00 59  
 macro avg 1.00 1.00 1.00 59  
weighted avg 1.00 1.00 1.00 59

print("\033[1mThe Balanced accuracy score obtained using Logistic Regression with PCA\033[0m is: \033[1m", balanced\_accuracy\_score(y\_test, y\_pred\_lr\_pca))

The Balanced accuracy score obtained using Logistic Regression with PCA is: 1.0

### Logistic Regression with PCA and standard scaler

sclr = StandardScaler()

x\_transform = sclr.fit\_transform(X)

x\_pca = PCA(n\_components=7, svd\_solver='randomized').fit\_transform(x\_transform)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_pca, Y, test\_size=0.33, random\_state=42)

clf.fit(x\_train,y\_train)

LogisticRegression(max\_iter=10000)

y\_pred\_lr\_pca\_sclr = clf.predict(x\_test)

#### Cross validate and measure the scores and calculate the mean accuracy of the model

scores = cross\_val\_score(clf, x\_train, y\_train, cv=cv, scoring='accuracy')

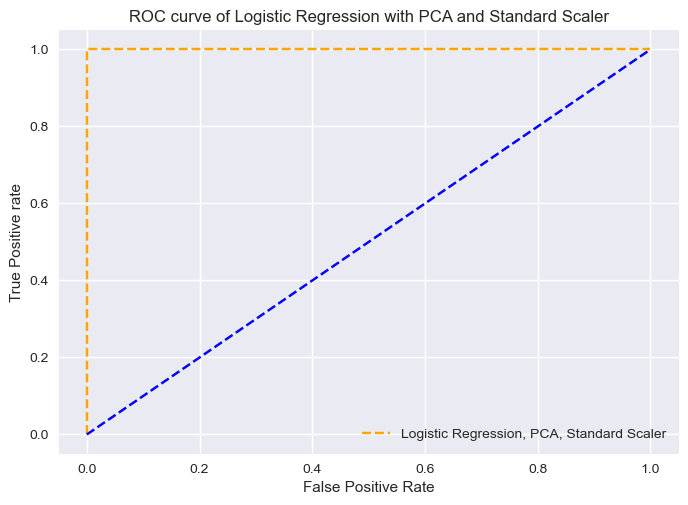
print("The accuracy of Logistic Regression with PCA and Standard Scaler is",scores.mean())

The accuracy of Logistic Regression with PCA and Standard Scaler is 0.9660714285714286

#### Let's plot the ROC-AUC curves, display the confusion matrix and print the classification report which displays the Precision, recall and the F1-score to evaluate the model. The subsequent cells represent the balanced accuracy score for the particular model [2]

# [2]  
probs = clf.predict\_proba(x\_test)  
  
fpr1\_lr\_pca\_sclr, tpr1\_lr\_pca\_sclr, thresh1\_gnb = roc\_curve(y\_test, probs[:,1], pos\_label=1)  
# roc curve for tpr = fpr   
random\_probs = [0 for i in range(len(y\_test))]  
p\_fpr\_lr\_pca\_sclr, p\_tpr\_lr\_pca\_sclr, \_ = roc\_curve(y\_test, random\_probs, pos\_label=1)  
  
##matplotlib  
import matplotlib.pyplot as plt  
plt.style.use('seaborn')  
  
# plot roc curves  
plt.plot(fpr1\_lr\_pca\_sclr, tpr1\_lr\_pca\_sclr, linestyle='--',color='orange', label='Logistic Regression, PCA, Standard Scaler')  
plt.plot(p\_fpr\_lr\_pca\_sclr, p\_tpr\_lr\_pca\_sclr, linestyle='--', color='blue')  
# title  
plt.title('ROC curve of Logistic Regression with PCA and Standard Scaler')  
# x label  
plt.xlabel('False Positive Rate')  
# y label  
plt.ylabel('True Positive rate')  
plt.legend(loc='best')

<matplotlib.legend.Legend at 0x2172a8aa5b0>



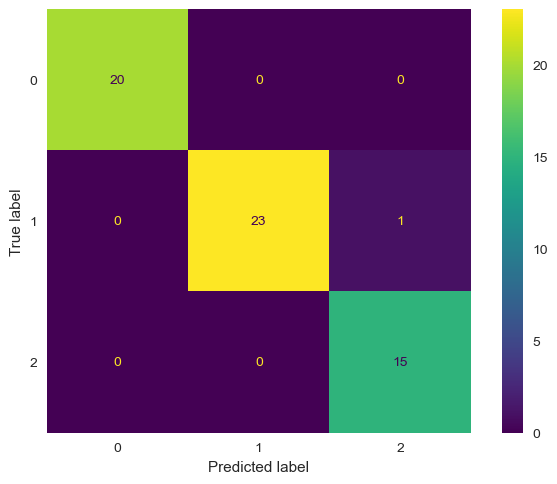
#### ROC AUC score

print('the ROC-AUC score of Logistic Regression with PCA and Standard Scaler', roc\_auc\_score(y\_test, clf.predict\_proba(x\_test), multi\_class='ovr'))

the ROC-AUC score of Logistic Regression with PCA and Standard Scaler 1.0

cm=confusion\_matrix(y\_test,y\_pred\_lr\_pca\_sclr,labels=clf.classes\_)  
disp = ConfusionMatrixDisplay(confusion\_matrix=cm,display\_labels=clf.classes\_)  
print(' \033[1mConfusion Matrix of Logistic Regression with PCA and Standard Scaler\033[0m')  
disp.plot()  
plt.grid(False)  
plt.show()

Confusion Matrix of Logistic Regression with PCA and Standard Scaler



print("\033[1mThe classification report of Logistic Regression with PCA and Standard Scaler is:\n\n------------------Classification Report------------------\033[0m \n",classification\_report(y\_test, y\_pred\_lr\_pca\_sclr))

The classification report of Logistic Regression with PCA and Standard Scaler is:  
  
------------------Classification Report------------------   
 precision recall f1-score support  
  
 0 1.00 1.00 1.00 20  
 1 1.00 0.96 0.98 24  
 2 0.94 1.00 0.97 15  
  
 accuracy 0.98 59  
 macro avg 0.98 0.99 0.98 59  
weighted avg 0.98 0.98 0.98 59

print("\033[1mThe Balanced accuracy score obtained using Logistic Regression with PCA and Standard Scaler\033[0m is:\033[1m", balanced\_accuracy\_score(y\_test, y\_pred\_lr\_pca\_sclr))

The Balanced accuracy score obtained using Logistic Regression with PCA and Standard Scaler is: 0.9861111111111112

# Decision Tree Classification Method

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.33, random\_state=42)  
decision\_tree = DecisionTreeClassifier(random\_state=42)

decision\_tree.fit(x\_train,y\_train)

DecisionTreeClassifier(random\_state=42)

#### Cross validate and measure the scores and calculate the mean accuracy of the model

cross\_val\_score(decision\_tree, x\_train, y\_train, cv=cv, scoring='accuracy')

array([0.8 , 0.93333333, 0.86666667, 0.8 , 0.93333333,  
 0.93333333, 0.8 , 0.92857143])

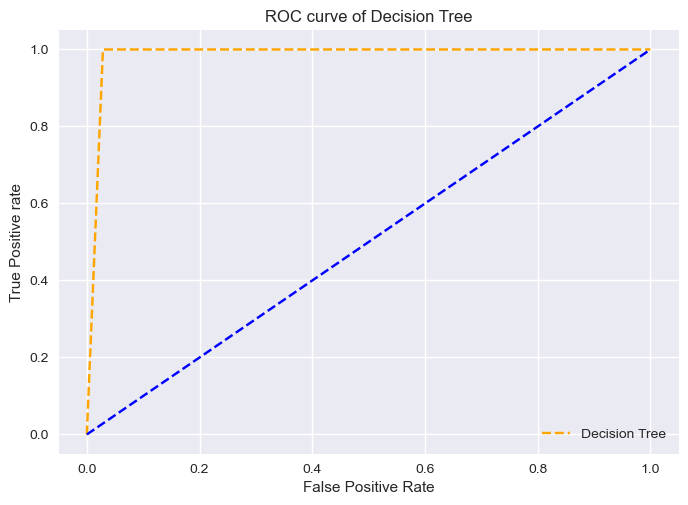
y\_pred\_dt = decision\_tree.predict(x\_test)

print("The mean accuracy obtained using Decision Tree is :",accuracy\_score(y\_test,y\_pred\_dt))

The mean accuracy obtained using Decision Tree is : 0.9661016949152542

#### Let's plot the ROC-AUC curves, display the confusion matrix and print the classification report which displays the Precision, recall and the F1-score to evaluate the model. The subsequent cells represent the balanced accuracy score for the particular model [2]

# [2]  
probs=decision\_tree.predict\_proba(x\_test)  
  
fpr1\_dt, tpr1\_dt, thresh1 = roc\_curve(y\_test, probs[:,1], pos\_label=1)  
# roc curve for tpr = fpr   
random\_probs = [0 for i in range(len(y\_test))]  
p\_fpr\_dt, p\_tpr\_dt, \_ = roc\_curve(y\_test, random\_probs, pos\_label=1)  
  
import matplotlib.pyplot as plt  
plt.style.use('seaborn')  
  
# plot roc curves  
plt.plot(fpr1\_dt, tpr1\_dt, linestyle='--',color='orange', label='Decision Tree')  
plt.plot(p\_fpr\_dt, p\_tpr\_dt, linestyle='--', color='blue')  
# title  
plt.title('ROC curve of Decision Tree')  
# x label  
plt.xlabel('False Positive Rate')  
# y label  
plt.ylabel('True Positive rate')  
plt.legend(loc='best')  
plt.show()



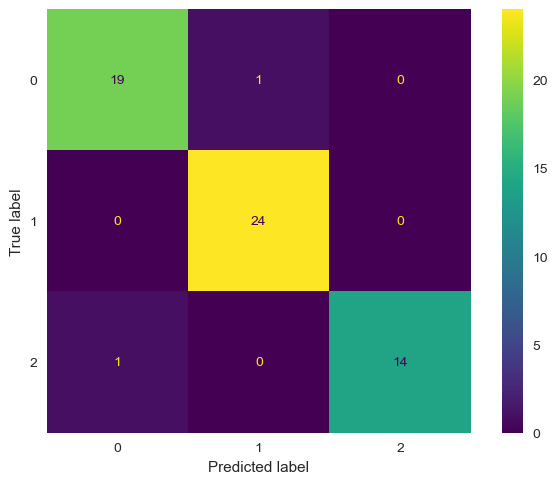
#### ROC AUC score

print('the ROC-AUC score of Decision Tree', roc\_auc\_score(y\_test, decision\_tree.predict\_proba(x\_test), multi\_class='ovr'))

the ROC-AUC score of Decision Tree 0.9715201465201465

cm=confusion\_matrix(y\_test,y\_pred\_dt,labels=decision\_tree.classes\_)  
disp = ConfusionMatrixDisplay(confusion\_matrix=cm,display\_labels=decision\_tree.classes\_)  
print(' \033[1mConfusion Matrix of Decision Tree\033[0m')  
disp.plot()  
plt.grid(False)  
plt.show()

Confusion Matrix of Decision Tree



#### We will change the depth of the decision tree to see the average accuracy achieved for all the specified depths of the decision trees

print("Variation of accuracy in Decision tree with different depths:")  
for d in range (1,10):  
 clf\_d = DecisionTreeClassifier(max\_depth=d, random\_state=0)  
 accuracy\_d=cross\_val\_score(clf\_d, wine.data, wine.target, cv=cv)  
 print("Depth: ", d, " Average accuracy:", accuracy\_d.mean())

Variation of accuracy in Decision tree with different depths:  
Depth: 1 Average accuracy: 0.6180830039525692  
Depth: 2 Average accuracy: 0.8312747035573123  
Depth: 3 Average accuracy: 0.9041501976284585  
Depth: 4 Average accuracy: 0.9041501976284585  
Depth: 5 Average accuracy: 0.8927865612648221  
Depth: 6 Average accuracy: 0.8927865612648221  
Depth: 7 Average accuracy: 0.8927865612648221  
Depth: 8 Average accuracy: 0.8927865612648221  
Depth: 9 Average accuracy: 0.8927865612648221

print("\033[1mThe classification report of Decision Tree is:\n\n------------------Classification Report------------------\033[0m \n",classification\_report(y\_test,y\_pred\_dt))

The classification report of Decision Tree is:  
  
------------------Classification Report------------------   
 precision recall f1-score support  
  
 0 0.95 0.95 0.95 20  
 1 0.96 1.00 0.98 24  
 2 1.00 0.93 0.97 15  
  
 accuracy 0.97 59  
 macro avg 0.97 0.96 0.97 59  
weighted avg 0.97 0.97 0.97 59

print("\033[1mThe Balanced accuracy score obtained using Decision Tree\033[0m is:\033[1m", balanced\_accuracy\_score(y\_test, y\_pred\_dt))

The Balanced accuracy score obtained using Decision Tree is: 0.9611111111111111

### Decision tree with PCA and Standard Scaler

#### Cross validate and measure the scores and calculate the mean accuracy of the model

sclr = StandardScaler()  
x\_transform = sclr.fit\_transform(X)  
x\_pca = PCA(n\_components=7).fit\_transform(sclr.fit\_transform(X))  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_pca,Y,test\_size=0.33,random\_state=42)  
decision\_tree.fit(x\_train,y\_train)  
scores = cross\_val\_score(decision\_tree, x\_train, y\_train, cv=cv, scoring='accuracy')  
y\_pred\_dt\_sclr\_pca = decision\_tree.predict(x\_test)

print("The accuracy obtained using Decision Tree with PCA and Standard Scaler is:",scores.mean())

The accuracy obtained using Decision Tree with PCA and Standard Scaler is: 0.9494047619047619

#### Let's plot the ROC-AUC curves, display the confusion matrix and print the classification report which displays the Precision, recall and the F1-score to evaluate the model. The subsequent cells represent the balanced accuracy score for the particular model [2]

# [2]  
probs=decision\_tree.predict\_proba(x\_test)  
  
fpr1\_dt\_sclr\_pca, tpr1\_dt\_sclr\_pca, thresh1 = roc\_curve(y\_test, probs[:,1], pos\_label=1)  
# roc curve for tpr = fpr   
random\_probs = [0 for i in range(len(y\_test))]  
p\_fpr\_dt\_sclr\_pca, p\_tpr\_dt\_sclr\_pca, \_ = roc\_curve(y\_test, random\_probs, pos\_label=1)  
  
import matplotlib.pyplot as plt  
plt.style.use('seaborn')  
  
# plot roc curves  
plt.plot(fpr1\_dt\_sclr\_pca, tpr1\_dt\_sclr\_pca, linestyle='--',color='orange', label='Decision Tree with PCA and Standard Scaler')  
plt.plot(p\_fpr\_dt\_sclr\_pca, p\_tpr\_dt\_sclr\_pca, linestyle='--', color='blue')  
# title  
plt.title('ROC curve of Decision Tree with PCA and Standard Scaler')  
# x label  
plt.xlabel('False Positive Rate')  
# y label  
plt.ylabel('True Positive rate')  
plt.legend(loc='best')  
plt.show()



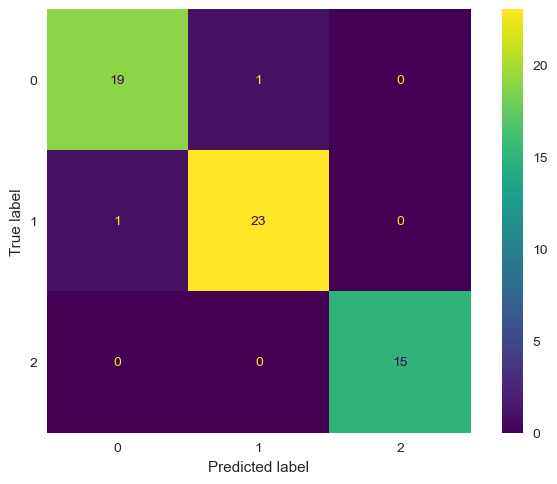
#### ROC AUC score

print('the ROC-AUC score of Decision Tree with PCA and Standard Scaler', roc\_auc\_score(y\_test, decision\_tree.predict\_proba(x\_test), multi\_class='ovr'))

the ROC-AUC score of Decision Tree with PCA and Standard Scaler 0.9756868131868132

cm=confusion\_matrix(y\_test,y\_pred\_dt\_sclr\_pca,labels=decision\_tree.classes\_)  
disp = ConfusionMatrixDisplay(confusion\_matrix=cm,display\_labels=decision\_tree.classes\_)  
print(' \033[1mConfusion Matrix of Decision Tree with PCA and Standard Scaler\033[0m')  
disp.plot()  
plt.grid(False)  
plt.show()

Confusion Matrix of Decision Tree with PCA and Standard Scaler



print("\033[1mThe classification report of Decision Tree with PCA and Standard Scaler is:\n\n------------------Classification Report------------------\033[0m \n",classification\_report(y\_test,y\_pred\_dt\_sclr\_pca))

The classification report of Decision Tree with PCA and Standard Scaler is:  
  
------------------Classification Report------------------   
 precision recall f1-score support  
  
 0 0.95 0.95 0.95 20  
 1 0.96 0.96 0.96 24  
 2 1.00 1.00 1.00 15  
  
 accuracy 0.97 59  
 macro avg 0.97 0.97 0.97 59  
weighted avg 0.97 0.97 0.97 59

print("\033[1mThe Balanced accuracy score of Decision Tree with PCA and Standard Scaler\033[0m is \033[1m:", balanced\_accuracy\_score(y\_test, y\_pred\_dt\_sclr\_pca))

The Balanced accuracy score of Decision Tree with PCA and Standard Scaler is : 0.9694444444444444

# Random Forest Classification

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.33, random\_state=42)

random\_forest = RandomForestClassifier(max\_depth=5,random\_state=0)

random\_forest.fit(x\_train, y\_train)

RandomForestClassifier(max\_depth=5, random\_state=0)

#### Cross validate and measure the scores and calculate the mean accuracy of the model

cross\_val\_score(random\_forest, X, Y, cv=cv, scoring='accuracy')

array([1. , 1. , 1. , 1. , 0.90909091,  
 0.95454545, 1. , 1. ])

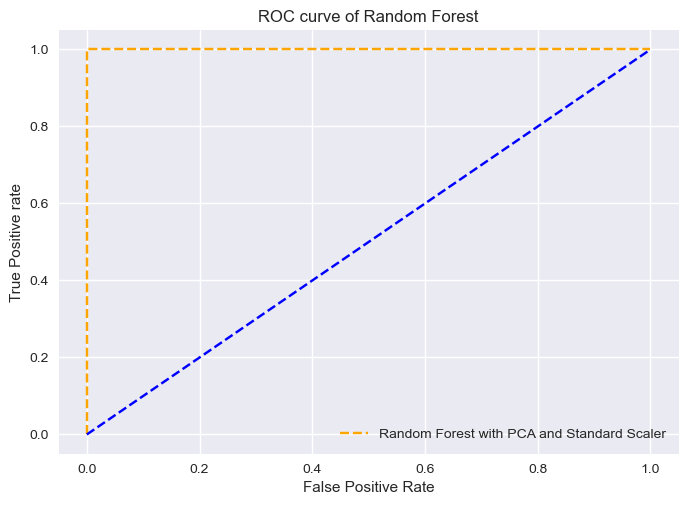
y\_pred\_rf = random\_forest.predict(x\_test)

print("The mean accuracy obtained using Random Forest Classfication is: ",accuracy\_score(y\_test,y\_pred\_rf))

The mean accuracy obtained using Random Forest Classfication is: 1.0

#### Let's plot the ROC-AUC curves, display the confusion matrix and print the classification report which displays the Precision, recall and the F1-score to evaluate the model. The subsequent cells represent the balanced accuracy score for the particular model [2]

# [2]  
probs=random\_forest.predict\_proba(x\_test)  
  
fpr1\_rf, tpr1\_rf, thresh1 = roc\_curve(y\_test, probs[:,1], pos\_label=1)  
# roc curve for tpr = fpr   
random\_probs = [0 for i in range(len(y\_test))]  
p\_fpr\_rf, p\_tpr\_rf, \_ = roc\_curve(y\_test, random\_probs, pos\_label=1)  
  
import matplotlib.pyplot as plt  
plt.style.use('seaborn')  
  
# plot roc curves  
plt.plot(fpr1\_rf, tpr1\_rf, linestyle='--',color='orange', label='Random Forest with PCA and Standard Scaler')  
plt.plot(p\_fpr\_rf, p\_tpr\_rf, linestyle='--', color='blue')  
# title  
plt.title('ROC curve of Random Forest')  
# x label  
plt.xlabel('False Positive Rate')  
# y label  
plt.ylabel('True Positive rate')  
plt.legend(loc='best')  
plt.show()



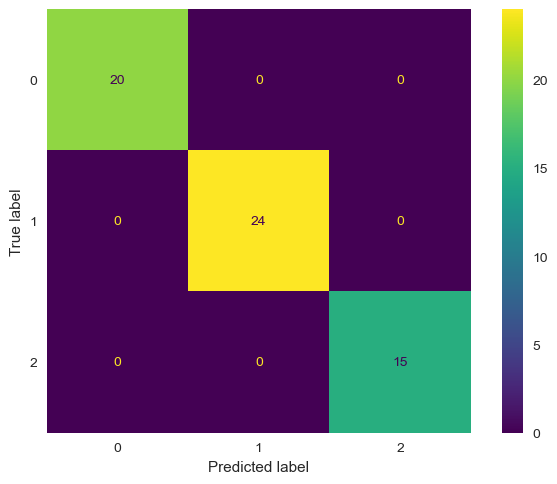
#### ROC AUC score

print('the ROC-AUC score of Random Forest', roc\_auc\_score(y\_test, random\_forest.predict\_proba(x\_test), multi\_class='ovr'))

the ROC-AUC score of Random Forest 1.0

cm=confusion\_matrix(y\_test,y\_pred\_rf,labels=random\_forest.classes\_)  
disp = ConfusionMatrixDisplay(confusion\_matrix=cm,display\_labels=random\_forest.classes\_)  
print(' \033[1mConfusion Matrix of Random Forest\033[0m')  
disp.plot()  
plt.grid(False)  
plt.show()

Confusion Matrix of Random Forest



print("\033[1mThe classification report of Random Forest is:\n\n------------------Classification Report------------------\033[0m \n",classification\_report(y\_test, y\_pred\_rf))

The classification report of Random Forest is:  
  
------------------Classification Report------------------   
 precision recall f1-score support  
  
 0 1.00 1.00 1.00 20  
 1 1.00 1.00 1.00 24  
 2 1.00 1.00 1.00 15  
  
 accuracy 1.00 59  
 macro avg 1.00 1.00 1.00 59  
weighted avg 1.00 1.00 1.00 59

print("\033[1mThe Balanced accuracy score obtained using Random Forest \033[0m is:\033[1m", balanced\_accuracy\_score(y\_test, y\_pred\_rf))

The Balanced accuracy score obtained using Random Forest is: 1.0

### Random Forest Classifier with PCA and Standard scaler

#### Cross validate and measure the scores and calculate the mean accuracy of the model

sclr = StandardScaler()  
x\_transform = sclr.fit\_transform(X)  
x\_pca = PCA(n\_components=7).fit\_transform(sclr.fit\_transform(X))  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_pca, Y, test\_size=0.33, random\_state=42)  
random\_forest.fit(x\_train,y\_train)  
scores = cross\_val\_score(random\_forest, x\_train, y\_train, cv=cv, scoring='accuracy')  
y\_pred\_rf\_sclr\_pca = random\_forest.predict(x\_test)

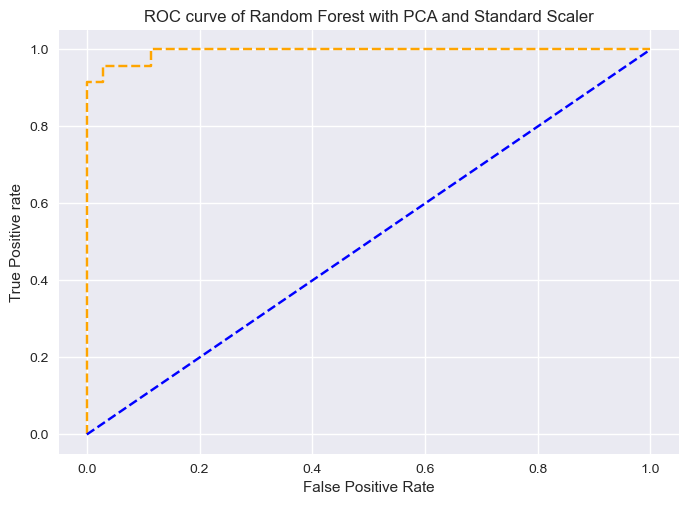
print("The accuracy obtained using Random Forest with PCA and Standard Scaler is: ",scores.mean())

The accuracy obtained using Random Forest with PCA and Standard Scaler is: 0.9660714285714286

#### Let's plot the ROC-AUC curves, display the confusion matrix and print the classification report which displays the Precision, recall and the F1-score to evaluate the model. The subsequent cells represent the balanced accuracy score for the particular model [2]

# [2]  
probs=random\_forest.predict\_proba(x\_test)  
  
fpr1\_rf\_sclr\_pca, tpr1\_rf\_sclr\_pca, thresh1 = roc\_curve(y\_test, probs[:,1], pos\_label=1)  
# roc curve for tpr = fpr   
random\_probs = [0 for i in range(len(y\_test))]  
p\_fpr\_rf\_sclr\_pca, p\_tpr\_rf\_sclr\_pca, \_ = roc\_curve(y\_test, random\_probs, pos\_label=1)  
  
import matplotlib.pyplot as plt  
plt.style.use('seaborn')  
  
# plot roc curves  
plt.plot(fpr1\_rf\_sclr\_pca, tpr1\_rf\_sclr\_pca, linestyle='--',color='orange', label='Decision Tree')  
plt.plot(p\_fpr\_rf\_sclr\_pca, p\_tpr\_rf\_sclr\_pca, linestyle='--', color='blue')  
# title  
plt.title('ROC curve of Random Forest with PCA and Standard Scaler')  
# x label  
plt.xlabel('False Positive Rate')  
# y label  
plt.ylabel('True Positive rate')

Text(0, 0.5, 'True Positive rate')



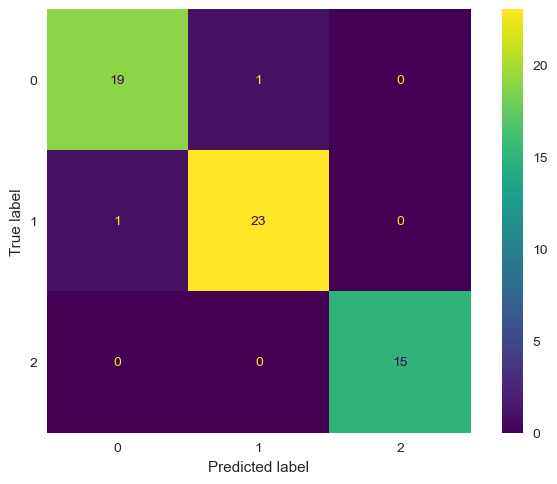
#### ROC AUC score

print('the ROC-AUC score of Random Forest with PCA and Standard Scaler', roc\_auc\_score(y\_test, random\_forest.predict\_proba(x\_test), multi\_class='ovr'))

the ROC-AUC score of Random Forest with PCA and Standard Scaler 0.9975885225885226

cm=confusion\_matrix(y\_test,y\_pred\_rf\_sclr\_pca,labels=random\_forest.classes\_)  
disp = ConfusionMatrixDisplay(confusion\_matrix=cm,display\_labels=random\_forest.classes\_)  
print(' \033[1mConfusion Matrix of Random Forest with PCA and Standard Scaler\033[0m')  
disp.plot()  
plt.grid(False)  
plt.show()

Confusion Matrix of Random Forest with PCA and Standard Scaler



print("\033[1mThe classification report of Random Forest with PCA and Standard Scaler is:\n\n------------------Classification Report------------------\033[0m\n",classification\_report(y\_test,y\_pred\_rf\_sclr\_pca))

The classification report of Random Forest with PCA and Standard Scaler is:  
  
------------------Classification Report------------------  
 precision recall f1-score support  
  
 0 0.95 0.95 0.95 20  
 1 0.96 0.96 0.96 24  
 2 1.00 1.00 1.00 15  
  
 accuracy 0.97 59  
 macro avg 0.97 0.97 0.97 59  
weighted avg 0.97 0.97 0.97 59

print("\033[1mThe Balanced accuracy score obtained using Random Forest with PCA and Standard Scaler\033[0m is:\033[1m", balanced\_accuracy\_score(y\_test, y\_pred\_rf\_sclr\_pca))

The Balanced accuracy score obtained using Random Forest with PCA and Standard Scaler is: 0.9694444444444444

# SVC - Support Vector Classifier

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.33, random\_state=42)

svc = SVC(C=10,kernel='rbf',probability=True)

svc.fit(x\_train, y\_train)

SVC(C=10, probability=True)

#### Cross validate and measure the scores and calculate the mean accuracy of the model

scores = cross\_val\_score(svc, x\_train, y\_train, scoring='accuracy', cv=cv)

print("The mean accuracy obtained using SVC is :",scores.mean())

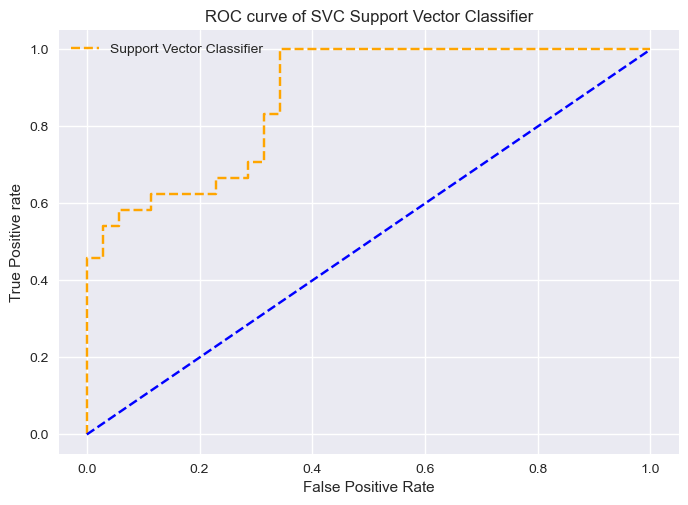
The mean accuracy obtained using SVC is : 0.6994047619047619

y\_pred\_svc = svc.predict(x\_test)

#### Let's plot the ROC-AUC curves, display the confusion matrix and print the classification report which displays the Precision, recall and the F1-score to evaluate the model. The subsequent cells represent the balanced accuracy score for the particular model [2]

# [2]  
probs = svc.predict\_proba(x\_test)  
  
fpr1\_svc, tpr1\_svc, thresh1\_svc = roc\_curve(y\_test, probs[:,1], pos\_label=1)  
# roc curve for tpr = fpr   
random\_probs = [0 for i in range(len(y\_test))]  
p\_fpr\_svc, p\_tpr\_svc, \_ = roc\_curve(y\_test, random\_probs, pos\_label=1)  
  
##matplotlib  
import matplotlib.pyplot as plt  
plt.style.use('seaborn')  
  
# plot roc curves  
plt.plot(fpr1\_svc, tpr1\_svc, linestyle='--',color='orange', label='Support Vector Classifier')  
plt.plot(p\_fpr\_svc, p\_tpr\_svc, linestyle='--', color='blue')  
# title  
plt.title('ROC curve of SVC Support Vector Classifier')  
# x label  
plt.xlabel('False Positive Rate')  
# y label  
plt.ylabel('True Positive rate')  
plt.legend(loc='best')

<matplotlib.legend.Legend at 0x217294d7d60>



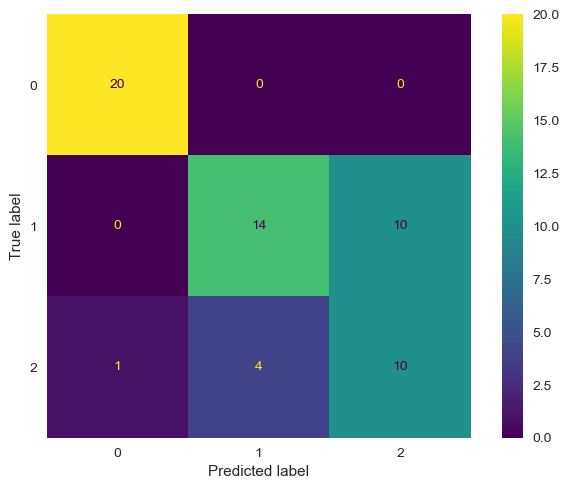
#### ROC AUC score

print('the ROC-AUC score of SVC', roc\_auc\_score(y\_test, svc.predict\_proba(x\_test), multi\_class='ovr'))

the ROC-AUC score of SVC 0.8994588744588744

cm=confusion\_matrix(y\_test,y\_pred\_svc,labels=svc.classes\_)  
disp = ConfusionMatrixDisplay(confusion\_matrix=cm,display\_labels=svc.classes\_)  
print(' \033[1mConfusion Matrix of SVC Support Vector Classifier\033[0m')  
disp.plot()  
plt.grid(False)  
plt.show()

Confusion Matrix of SVC Support Vector Classifier



print("\033[1mThe classification report of SVC is:\n\n------------------Classification Report------------------\033[0m \n",classification\_report(y\_test, y\_pred\_svc))

The classification report of SVC is:  
  
------------------Classification Report------------------   
 precision recall f1-score support  
  
 0 0.95 1.00 0.98 20  
 1 0.78 0.58 0.67 24  
 2 0.50 0.67 0.57 15  
  
 accuracy 0.75 59  
 macro avg 0.74 0.75 0.74 59  
weighted avg 0.77 0.75 0.75 59

print("\033[1mThe Balanced accuracy score obtained using SVC\033[0m is:\033[1m", balanced\_accuracy\_score(y\_test, y\_pred\_svc))

The Balanced accuracy score obtained using SVC is: 0.75

### SVC with standard scaler and PCA

x\_pca = PCA(n\_components=7).fit\_transform(sclr.fit\_transform(X))

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_pca, Y, test\_size=0.33, random\_state=42)

svc.fit(x\_train,y\_train)

SVC(C=10, probability=True)

#### Cross validate and measure the scores and calculate the mean accuracy of the model

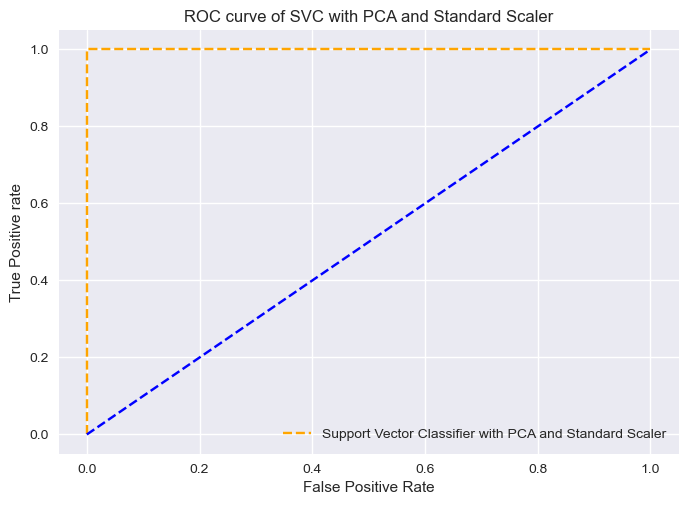
scores = cross\_val\_score(svc, x\_train, y\_train, cv=cv, scoring='accuracy')

y\_pred\_svc\_pca\_sclr = svc.predict(x\_test)

#### Let's plot the ROC-AUC curves, display the confusion matrix and print the classification report which displays the Precision, recall and the F1-score to evaluate the model. The subsequent cells represent the balanced accuracy score for the particular model [2]

# [2]  
probs = svc.predict\_proba(x\_test)  
  
fpr1\_svc\_pca\_sclr, tpr1\_svc\_pca\_sclr, thresh1\_svc = roc\_curve(y\_test, probs[:,1], pos\_label=1)  
# roc curve for tpr = fpr   
random\_probs = [0 for i in range(len(y\_test))]  
p\_fpr\_svc\_pca\_sclr, p\_tpr\_svc\_pca\_sclr, \_ = roc\_curve(y\_test, random\_probs, pos\_label=1)  
  
##matplotlib  
import matplotlib.pyplot as plt  
plt.style.use('seaborn')  
  
# plot roc curves  
plt.plot(fpr1\_svc\_pca\_sclr, tpr1\_svc\_pca\_sclr, linestyle='--',color='orange', label='Support Vector Classifier with PCA and Standard Scaler')  
plt.plot(p\_fpr\_svc\_pca\_sclr, p\_tpr\_svc\_pca\_sclr, linestyle='--', color='blue')  
# title  
plt.title('ROC curve of SVC with PCA and Standard Scaler')  
# x label  
plt.xlabel('False Positive Rate')  
# y label  
plt.ylabel('True Positive rate')  
plt.legend(loc='best')

<matplotlib.legend.Legend at 0x2172b5cdb20>



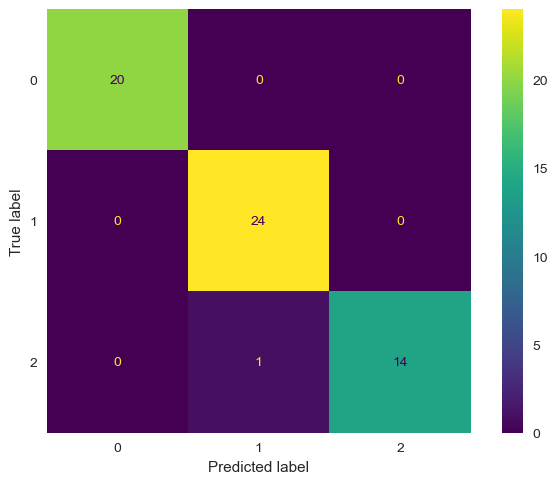
#### ROC AUC score

print('the ROC-AUC score of SVC with PCA and Standard Scaler', roc\_auc\_score(y\_test, svc.predict\_proba(x\_test), multi\_class='ovr'))

the ROC-AUC score of SVC with PCA and Standard Scaler 1.0

cm=confusion\_matrix(y\_test,y\_pred\_svc\_pca\_sclr,labels=svc.classes\_)  
disp = ConfusionMatrixDisplay(confusion\_matrix=cm,display\_labels=svc.classes\_)  
print(' \033[1mConfusion Matrix of SVC with PCA and Standard Scaler033[0m')  
disp.plot()  
plt.grid(False)  
plt.show()

Confusion Matrix of SVC with PCA and Standard Scaler033[0m



print("\033[1mThe classification report of SVC with PCA and Standard Scaler is:\n\n------------------Classification Report------------------\033[0m\n",classification\_report(y\_test, y\_pred\_svc\_pca\_sclr))

The classification report of SVC with PCA and Standard Scaler is:  
  
------------------Classification Report------------------  
 precision recall f1-score support  
  
 0 1.00 1.00 1.00 20  
 1 0.96 1.00 0.98 24  
 2 1.00 0.93 0.97 15  
  
 accuracy 0.98 59  
 macro avg 0.99 0.98 0.98 59  
weighted avg 0.98 0.98 0.98 59

print("\033[1mThe Balanced accuracy score obtained using SVC with PCA and Standard Scaler\033[0m is:\033[1m", balanced\_accuracy\_score(y\_test, y\_pred\_svc\_pca\_sclr))

The Balanced accuracy score obtained using SVC with PCA and Standard Scaler is: 0.9777777777777779

# K Nearest Neighbours Classification [10]

knn = KNeighborsClassifier(n\_neighbors=5)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.33, random\_state=42)

knn.fit(x\_train,y\_train)

KNeighborsClassifier()

#### Cross validate and measure the scores and calculate the mean accuracy of the model

scores = cross\_val\_score(knn, x\_train, y\_train, cv=cv, scoring='accuracy')

y\_pred\_knn = knn.predict(x\_test)

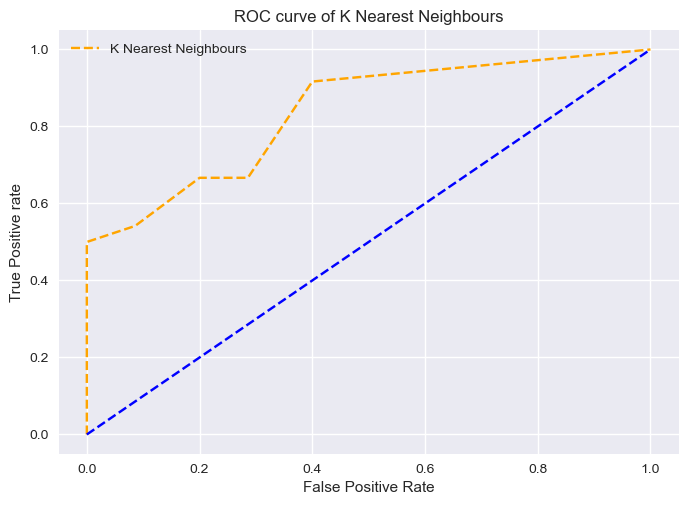
print("The mean accuracy obtained using KNN is: ",scores.mean())

The mean accuracy obtained using KNN is: 0.7321428571428572

#### Let's plot the ROC-AUC curves, display the confusion matrix and print the classification report which displays the Precision, recall and the F1-score to evaluate the model. The subsequent cells represent the balanced accuracy score for the particular model [2]

# [2]  
probs = knn.predict\_proba(x\_test)  
  
fpr1\_knn, tpr1\_knn, thresh1\_svc = roc\_curve(y\_test, probs[:,1], pos\_label=1)  
# roc curve for tpr = fpr   
random\_probs = [0 for i in range(len(y\_test))]  
p\_fpr\_knn, p\_tpr\_knn, \_ = roc\_curve(y\_test, random\_probs, pos\_label=1)  
  
##matplotlib  
import matplotlib.pyplot as plt  
plt.style.use('seaborn')  
  
# plot roc curves  
plt.plot(fpr1\_knn, tpr1\_knn, linestyle='--',color='orange', label='K Nearest Neighbours')  
plt.plot(p\_fpr\_knn, p\_tpr\_knn, linestyle='--', color='blue')  
# title  
plt.title('ROC curve of K Nearest Neighbours')  
# x label  
plt.xlabel('False Positive Rate')  
# y label  
plt.ylabel('True Positive rate')  
plt.legend(loc='best')

<matplotlib.legend.Legend at 0x2172a49a640>



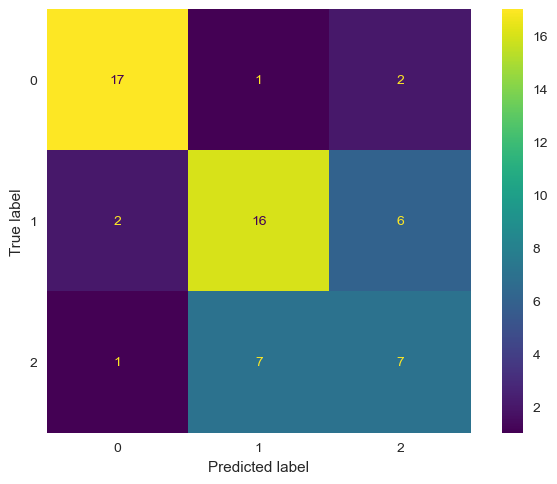
#### ROC AUC score

print('the ROC-AUC score of K Nearest Neighbours', roc\_auc\_score(y\_test, knn.predict\_proba(x\_test), multi\_class='ovr'))

the ROC-AUC score of K Nearest Neighbours 0.875467587967588

cm=confusion\_matrix(y\_test,y\_pred\_knn,labels=knn.classes\_)  
disp = ConfusionMatrixDisplay(confusion\_matrix=cm,display\_labels=knn.classes\_)  
print(' \033[1mConfusion Matrix of K Nearest Neighbours\033[0m')  
disp.plot()  
plt.grid(False)  
plt.show()

Confusion Matrix of K Nearest Neighbours



print("\033[1mThe classification report of K Nearest Neighbours Classification is:\n\n------------------Classification Report------------------\033[0m\n",classification\_report(y\_test, y\_pred\_knn))

The classification report of K Nearest Neighbours Classification is:  
  
------------------Classification Report------------------  
 precision recall f1-score support  
  
 0 0.85 0.85 0.85 20  
 1 0.67 0.67 0.67 24  
 2 0.47 0.47 0.47 15  
  
 accuracy 0.68 59  
 macro avg 0.66 0.66 0.66 59  
weighted avg 0.68 0.68 0.68 59

print("\033[1mThe Balanced accuracy score obtained using K Nearest Neighbours\033[0m is:\033[1m", balanced\_accuracy\_score(y\_test, y\_pred\_knn))

The Balanced accuracy score obtained using K Nearest Neighbours is: 0.6611111111111111

### K Nearest Neighbours with Standard Scaler and PCA

x\_pca = PCA(n\_components=7).fit\_transform(sclr.fit\_transform(X))

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_pca, Y, test\_size=0.33, random\_state=42)

knn.fit(x\_train, y\_train)

KNeighborsClassifier()

#### Cross validate and measure the scores and calculate the mean accuracy of the model

score = cross\_val\_score(knn, x\_train, y\_train, cv = cv, scoring='accuracy')

y\_pred\_knn\_pca\_sclr = knn.predict(x\_test)

print("The mean accuracy obtained using KNN with Standard Scaler and PCA is: ",score.mean())

The mean accuracy obtained using KNN with Standard Scaler and PCA is: 0.9494047619047619

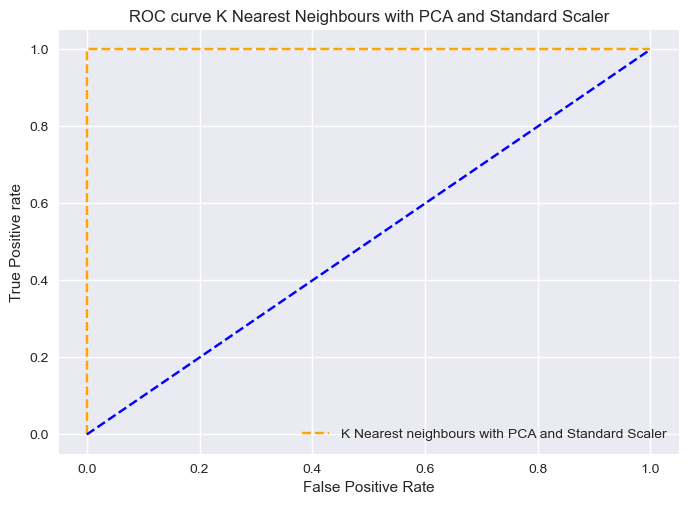
score

array([0.93333333, 1. , 0.93333333, 0.93333333, 0.86666667,  
 1. , 1. , 0.92857143])

#### Let's plot the ROC-AUC curves, display the confusion matrix and print the classification report which displays the Precision, recall and the F1-score to evaluate the model. The subsequent cells represent the balanced accuracy score for the particular model [2]

# [2]  
probs = knn.predict\_proba(x\_test)  
  
fpr1\_knn\_pca\_sclr, tpr1\_knn\_pca\_sclr, thresh1\_svc = roc\_curve(y\_test, probs[:,1], pos\_label=1)  
# roc curve for tpr = fpr   
random\_probs = [0 for i in range(len(y\_test))]  
p\_fpr\_knn\_pca\_sclr, p\_tpr\_knn\_pca\_sclr, \_ = roc\_curve(y\_test, random\_probs, pos\_label=1)  
  
##matplotlib  
import matplotlib.pyplot as plt  
plt.style.use('seaborn')  
  
# plot roc curves  
plt.plot(fpr1\_knn\_pca\_sclr, tpr1\_knn\_pca\_sclr, linestyle='--',color='orange', label='K Nearest neighbours with PCA and Standard Scaler')  
plt.plot(p\_fpr\_knn\_pca\_sclr, p\_tpr\_knn\_pca\_sclr, linestyle='--', color='blue')  
# title  
plt.title('ROC curve K Nearest Neighbours with PCA and Standard Scaler')  
# x label  
plt.xlabel('False Positive Rate')  
# y label  
plt.ylabel('True Positive rate')  
plt.legend(loc='best')

<matplotlib.legend.Legend at 0x2172d400e20>



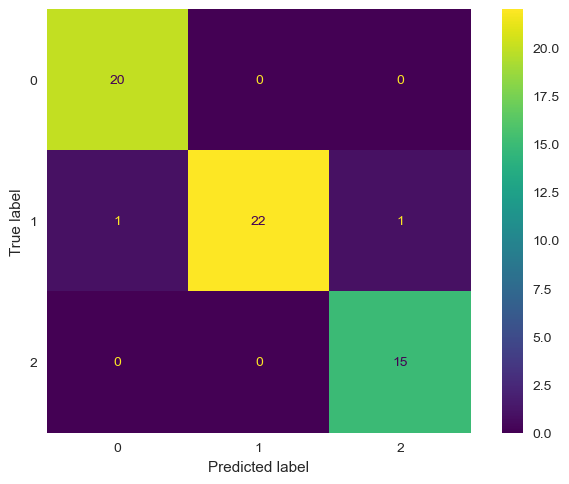
#### ROC AUC score

print('the ROC-AUC score of KNN with PCA and Standard Scaler', roc\_auc\_score(y\_test, knn.predict\_proba(x\_test), multi\_class='ovr'))

the ROC-AUC score of KNN with PCA and Standard Scaler 1.0

cm=confusion\_matrix(y\_test,y\_pred\_knn\_pca\_sclr,labels=knn.classes\_)  
disp = ConfusionMatrixDisplay(confusion\_matrix=cm,display\_labels=knn.classes\_)  
print(' \033[1mConfusion Matrix of K Nearest Neighbours with PCA and Standard Scaler\033[0m')  
disp.plot()  
plt.grid(False)  
plt.show()

Confusion Matrix of K Nearest Neighbours with PCA and Standard Scaler



print("\033[1mThe classification report of K Nearest Neighbours with Standard Scaler and PCA is:\n\n------------------Classification Report------------------\033[0m\n",classification\_report(y\_test, y\_pred\_knn\_pca\_sclr))

The classification report of K Nearest Neighbours with Standard Scaler and PCA is:  
  
------------------Classification Report------------------  
 precision recall f1-score support  
  
 0 0.95 1.00 0.98 20  
 1 1.00 0.92 0.96 24  
 2 0.94 1.00 0.97 15  
  
 accuracy 0.97 59  
 macro avg 0.96 0.97 0.97 59  
weighted avg 0.97 0.97 0.97 59

print("\033[1mThe Balanced accuracy score obtained using K Nearest Neighbours with Standard Scaler and PCA\033[0m is:\033[1m", balanced\_accuracy\_score(y\_test, y\_pred\_knn\_pca\_sclr))

The Balanced accuracy score obtained using K Nearest Neighbours with Standard Scaler and PCA is: 0.9722222222222222

# Ada Boost Classifier [11]

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.33, random\_state=42)  
ada\_boost = AdaBoostClassifier(n\_estimators=100)  
ada\_boost.fit(x\_train, y\_train)

AdaBoostClassifier(n\_estimators=100)

#### Cross validate and measure the scores and calculate the mean accuracy of the model

cross\_val\_score(ada\_boost, X, Y, cv=cv, scoring='accuracy')

array([0.86956522, 0.73913043, 0.90909091, 0.72727273, 0.95454545,  
 0.95454545, 0.81818182, 0.81818182])

y\_pred\_ada = ada\_boost.predict(x\_test)

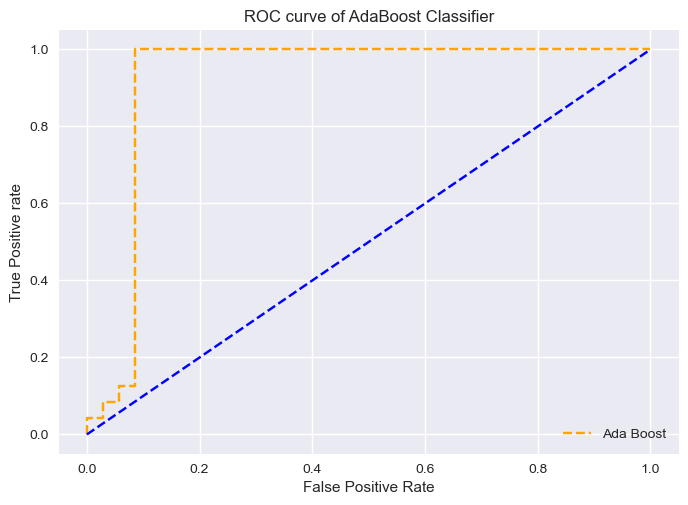
print("The accuracy obtained using Ada Boost classifier is: ",accuracy\_score(y\_test,y\_pred\_ada))

The accuracy obtained using Ada Boost classifier is: 0.9152542372881356

#### Let's plot the ROC-AUC curves, display the confusion matrix and print the classification report which displays the Precision, recall and the F1-score to evaluate the model. The subsequent cells represent the balanced accuracy score for the particular model [2]

# [2]  
probs = ada\_boost.predict\_proba(x\_test)  
  
fpr1\_ada, tpr1\_ada, thresh1\_svc = roc\_curve(y\_test, probs[:,1], pos\_label=1)  
# roc curve for tpr = fpr   
random\_probs = [0 for i in range(len(y\_test))]  
p\_fpr\_ada, p\_tpr\_ada, \_ = roc\_curve(y\_test, random\_probs, pos\_label=1)  
  
##matplotlib  
import matplotlib.pyplot as plt  
plt.style.use('seaborn')  
  
# plot roc curves  
plt.plot(fpr1\_ada, tpr1\_ada, linestyle='--',color='orange', label='Ada Boost')  
plt.plot(p\_fpr\_ada, p\_tpr\_ada, linestyle='--', color='blue')  
# title  
plt.title('ROC curve of AdaBoost Classifier')  
# x label  
plt.xlabel('False Positive Rate')  
# y label  
plt.ylabel('True Positive rate')  
plt.legend(loc='best')

<matplotlib.legend.Legend at 0x2172d68e070>



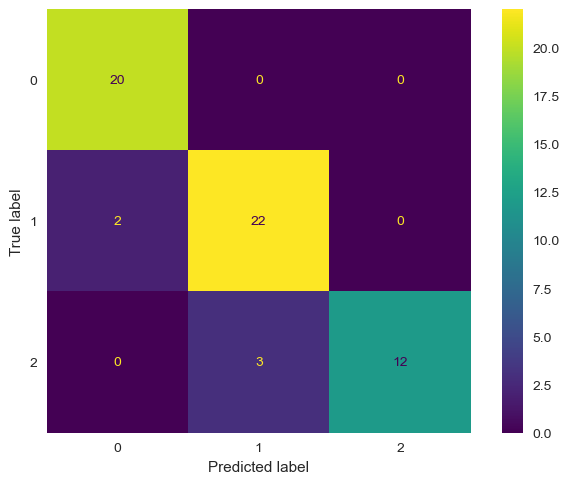
#### ROC AUC score

print('the ROC-AUC score of Ada Boost Classifer', roc\_auc\_score(y\_test, ada\_boost.predict\_proba(x\_test), multi\_class='ovr'))

the ROC-AUC score of Ada Boost Classifer 0.9738095238095238

cm=confusion\_matrix(y\_test,y\_pred\_ada,labels=ada\_boost.classes\_)  
disp = ConfusionMatrixDisplay(confusion\_matrix=cm,display\_labels=ada\_boost.classes\_)  
print(' \033[1mConfusion Matrix of AdaBoost Classifier\033[0m')  
disp.plot()  
plt.grid(False)  
plt.show()

Confusion Matrix of AdaBoost Classifier



print("\033[1mThe classification report of AdaBoost Classifier is:\n\n------------------Classification Report------------------\033[0m\n",classification\_report(y\_test, y\_pred\_ada))

The classification report of AdaBoost Classifier is:  
  
------------------Classification Report------------------  
 precision recall f1-score support  
  
 0 0.91 1.00 0.95 20  
 1 0.88 0.92 0.90 24  
 2 1.00 0.80 0.89 15  
  
 accuracy 0.92 59  
 macro avg 0.93 0.91 0.91 59  
weighted avg 0.92 0.92 0.91 59

print("\033[1mThe Balanced accuracy score obtained using AdaBoost Classifier\033[0m is:\033[1m", balanced\_accuracy\_score(y\_test, y\_pred\_ada))

The Balanced accuracy score obtained using AdaBoost Classifier is: 0.9055555555555556

### Ada Boost with Standard Scaler and PCA

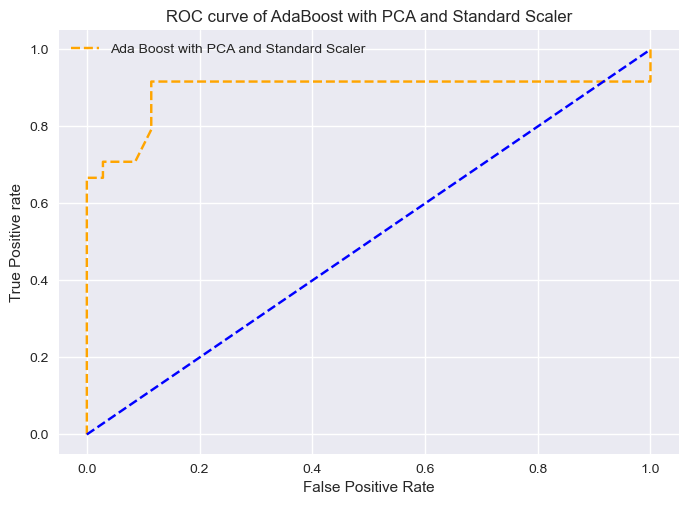
#### Cross validate and measure the scores and calculate the mean accuracy of the model

sclr = StandardScaler()  
x\_transform = sclr.fit\_transform(X)  
x\_pca = PCA(n\_components=7).fit\_transform(sclr.fit\_transform(X))  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_pca,Y,test\_size=0.33,random\_state=42)  
ada\_boost.fit(x\_train,y\_train)  
scores = cross\_val\_score(ada\_boost, x\_train, y\_train, cv=cv, scoring='accuracy')  
y\_pred\_ada\_sclr\_pca = ada\_boost.predict(x\_test)

#### Let's plot the ROC-AUC curves, display the confusion matrix and print the classification report which displays the Precision, recall and the F1-score to evaluate the model. The subsequent cells represent the balanced accuracy score for the particular model [2]

# [2]  
probs = ada\_boost.predict\_proba(x\_test)  
  
fpr1\_ada\_sclr\_pca, tpr1\_ada\_sclr\_pca, thresh1\_svc = roc\_curve(y\_test, probs[:,1], pos\_label=1)  
# roc curve for tpr = fpr   
random\_probs = [0 for i in range(len(y\_test))]  
p\_fpr\_ada\_sclr\_pca, p\_tpr\_ada\_sclr\_pca, \_ = roc\_curve(y\_test, random\_probs, pos\_label=1)  
  
##matplotlib  
import matplotlib.pyplot as plt  
plt.style.use('seaborn')  
  
# plot roc curves  
plt.plot(fpr1\_ada\_sclr\_pca, tpr1\_ada\_sclr\_pca, linestyle='--',color='orange', label='Ada Boost with PCA and Standard Scaler')  
plt.plot(p\_fpr\_ada\_sclr\_pca, p\_tpr\_ada\_sclr\_pca, linestyle='--', color='blue')  
# title  
plt.title('ROC curve of AdaBoost with PCA and Standard Scaler')  
# x label  
plt.xlabel('False Positive Rate')  
# y label  
plt.ylabel('True Positive rate')  
plt.legend(loc='best')

<matplotlib.legend.Legend at 0x2172d6730a0>



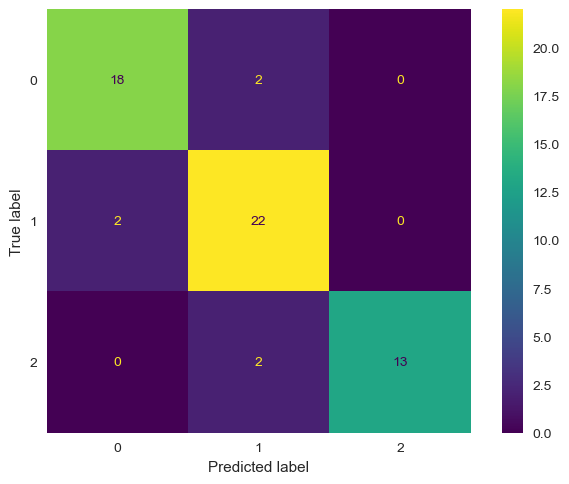
#### ROC AUC score

print('the ROC-AUC score of Ada Boost with PCA and Standard Scaler', roc\_auc\_score(y\_test, ada\_boost.predict\_proba(x\_test), multi\_class='ovr'))

the ROC-AUC score of Ada Boost with PCA and Standard Scaler 0.9335164835164834

cm=confusion\_matrix(y\_test,y\_pred\_ada\_sclr\_pca,labels=ada\_boost.classes\_)  
disp = ConfusionMatrixDisplay(confusion\_matrix=cm,display\_labels=ada\_boost.classes\_)  
print(' \033[1mConfusion Matrix of AdaBoost with PCA and Standard Scaler\033[0m')  
disp.plot()  
plt.grid(False)  
plt.show()

Confusion Matrix of AdaBoost with PCA and Standard Scaler



print("\033[1mThe classification report of AdaBoost with PCA and Standard Scaler is:\n\n------------------Classification Report------------------\033[0m\n",classification\_report(y\_test, y\_pred\_ada\_sclr\_pca))

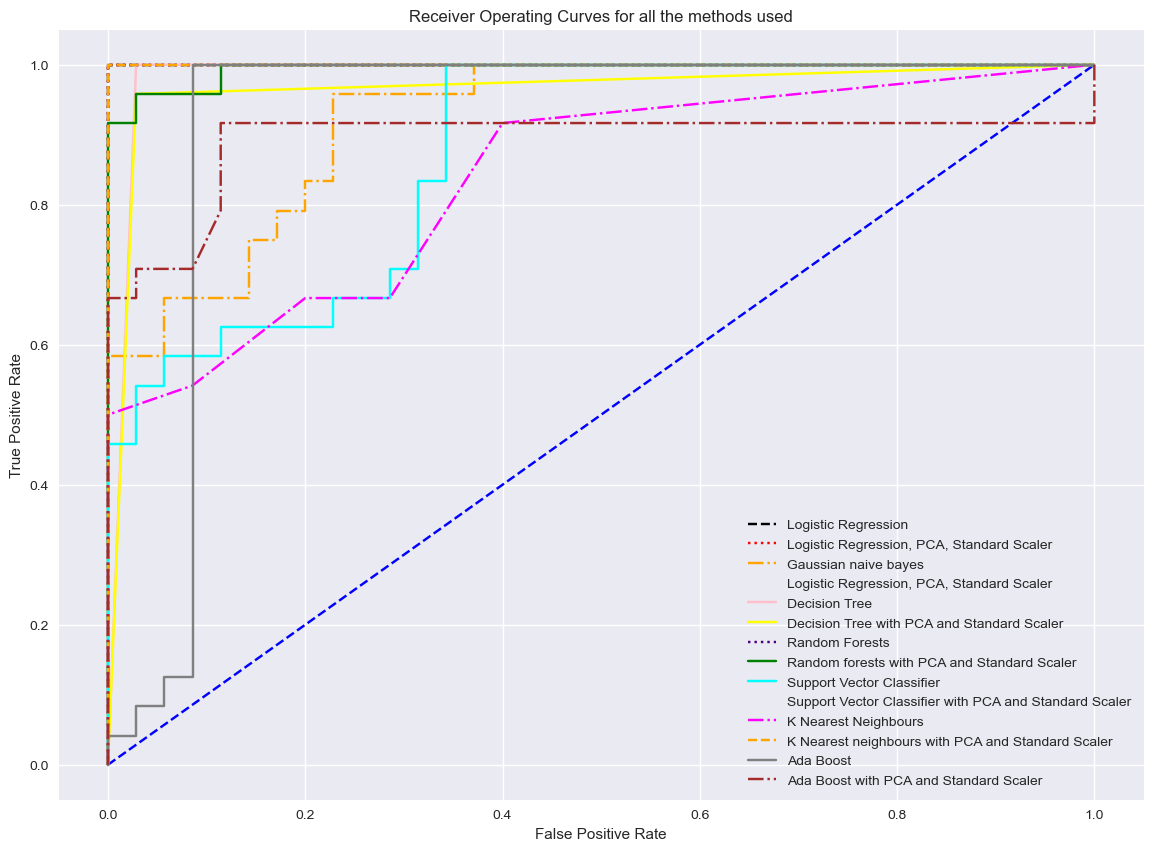
The classification report of AdaBoost with PCA and Standard Scaler is:  
  
------------------Classification Report------------------  
 precision recall f1-score support  
  
 0 0.90 0.90 0.90 20  
 1 0.85 0.92 0.88 24  
 2 1.00 0.87 0.93 15  
  
 accuracy 0.90 59  
 macro avg 0.92 0.89 0.90 59  
weighted avg 0.90 0.90 0.90 59

print("\033[1mThe Balanced accuracy score obtained using AdaBoost Classifier\033[0m is:\033[1m", balanced\_accuracy\_score(y\_test, y\_pred\_ada\_sclr\_pca))

The Balanced accuracy score obtained using AdaBoost Classifier is: 0.8944444444444445

## Plotting all the ROC and AUC curves in the same graph to get an overview of every model used [2]

##supported values are '-', '--', '-.', ':', 'None', ' ', '', 'solid', 'dashed', 'dashdot', 'dotted'  
plt.figure(figsize=(14,10))  
  
#reference line  
plt.plot(p\_fpr\_full, p\_tpr\_full, linestyle='--', color='blue')  
  
#logistic regression with Kfold  
plt.plot(fpr1\_full, tpr1\_full, linestyle='--', color='black', label='Logistic Regression')  
  
#logistic regression without kfold  
plt.plot(fpr1\_lr\_pca\_sclr, tpr1\_lr\_pca\_sclr, linestyle=':', color='red', label='Logistic Regression, PCA, Standard Scaler')  
  
#gaussian naive bayes  
plt.plot(fpr1\_gnb, tpr1\_gnb, linestyle='-.',color='orange', label='Gaussian naive bayes')  
  
#logistic regression with standard scaler and PCA  
plt.plot(fpr1\_lr\_pca\_sclr, tpr1\_lr\_pca\_sclr, linestyle='None', color='violet', label='Logistic Regression, PCA, Standard Scaler')  
  
#Decision Tree  
plt.plot(fpr1\_dt, tpr1\_dt, linestyle='solid',color='pink', label='Decision Tree')  
  
#Decision Tree with Scaler and PCA  
plt.plot(fpr1\_dt\_sclr\_pca, tpr1\_dt\_sclr\_pca, linestyle='solid',color='yellow', label='Decision Tree with PCA and Standard Scaler')  
  
#Random Forest  
plt.plot(fpr1\_rf, tpr1\_rf, linestyle=':',color='indigo', label='Random Forests')  
  
#Random Forest with PCA and Standard Scaler  
plt.plot(fpr1\_rf\_sclr\_pca, tpr1\_rf\_sclr\_pca, linestyle='-',color='green', label='Random forests with PCA and Standard Scaler')  
  
#SVC   
plt.plot(fpr1\_svc, tpr1\_svc, linestyle='solid',color='cyan', label='Support Vector Classifier')  
  
#SVC with standard scaler and PCA  
plt.plot(fpr1\_svc\_pca\_sclr, tpr1\_svc\_pca\_sclr, linestyle=' ',color='indigo', label='Support Vector Classifier with PCA and Standard Scaler')  
  
#KNN  
plt.plot(fpr1\_knn, tpr1\_knn, linestyle='-.',color='magenta', label='K Nearest Neighbours')  
  
#KNN with standard scaler and PCA  
plt.plot(fpr1\_knn\_pca\_sclr, tpr1\_knn\_pca\_sclr, linestyle='--',color='orange', label='K Nearest neighbours with PCA and Standard Scaler')  
  
#AdaBoost  
plt.plot(fpr1\_ada, tpr1\_ada, linestyle='-',color='gray', label='Ada Boost')  
  
#AdaBoost with Standard Scaler and PCA  
plt.plot(fpr1\_ada\_sclr\_pca, tpr1\_ada\_sclr\_pca, linestyle='-.',color='brown', label='Ada Boost with PCA and Standard Scaler')  
  
  
plt.xlabel('False Positive Rate')  
plt.ylabel('True Positive Rate')  
plt.legend(loc='best')  
plt.title('Receiver Operating Curves for all the methods used')  
plt.show()



# References

#### [1] MachineLearningMastery.com, 'How to configure k-fold cross-validation' [online].

#### Available: <https://machinelearningmastery.com/how-to-configure-k-fold-cross-validation/> . [Accessed 13-Feb-2023].

#### [2] analyticsvidhya.com, 'Guide to AUC ROC Curve in Machine Learning : What Is Specificity?'

#### Available: <https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/> . [Accessed 18-Feb-2023].

#### [3] wellsr.com, 'Python Agglomerative Clustering with sklearn'

#### Available: <https://wellsr.com/python/python-agglomerative-clustering-with-sklearn/> . [Accessed 8-Feb-2023]

#### [4] towardsdatascience.com, 'Visualizing Clusters with Python’s Matplotlib'

#### Available: <https://towardsdatascience.com/visualizing-clusters-with-pythons-matplolib-35ae03d87489> . [Accessed 8-Feb-2023]

#### [5] scikit-learn.org, ' DBSCAN'

#### Available: <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html> . [Accessed on 15-Feb-2023]

#### [6] scikit-learn.org, ' Birch'

#### Available: <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.Birch.html> . [Accessed on 15-Feb-2023]

#### [7] scikit-learn.org, ' AffinityPropagation'

#### Available: <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AffinityPropagation.html> . [Accessed on 16-Feb-2023]

#### [8] scikit-learn.org, 'MeanShift'

#### <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MeanShift.html> . [Accessed on 17-Feb-2023]

#### [9] scikit-learn.org, 'gaussian-naive-bayes'

#### Available: <https://scikit-learn.org/stable/modules/naive_bayes.html#gaussian-naive-bayes> . [Accessed on 18-Feb-2023]

#### [10] scikit-learn.org, 'KNeighborsClassifier'

#### Available: <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html> . [Accessed on 18-Feb-2023]

#### [11] scikit-learn.org, 'AdaBoostClassifier'

#### Available: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html> .[Accessed on 19-Feb-2023]