#Importing necessary libraries

import pandas as pd

import numpy as np

import sklearn

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import LabelEncoder , MinMaxScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import classification\_report

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.neural\_network import MLPClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

#Load the dataset (breast cancer data)

breast\_cancer = pd.read\_csv('/content/breast cancer classification dataset.csv')

#Droping highest null value's and unique values column (here it is-->id,Unnamed: 32)

breast\_cancer = breast\_cancer.drop(['id','Unnamed: 32'],axis=1)

#Imputing Missing values

impute = SimpleImputer(missing\_values=np.nan,strategy='mean')

impute.fit(breast\_cancer[['radius\_mean']])

breast\_cancer['radius\_mean'] = impute.transform(breast\_cancer[['radius\_mean']])

impute.fit(breast\_cancer[['fractal\_dimension\_worst']])

breast\_cancer['fractal\_dimension\_worst'] = impute.transform(breast\_cancer[['fractal\_dimension\_worst']])

#Encode categorical feature

labelling = LabelEncoder()

breast\_cancer['diagnosis'] = labelling.fit\_transform(breast\_cancer['diagnosis'])

#Feature and Label selection

features = breast\_cancer.iloc[:,1:31]

label = breast\_cancer.iloc[:,0]

#Split train-test (8:2)

y = pd.DataFrame(label)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(features, label, test\_size=0.2, random\_state=0, stratify = y)

#support vector before PCA

svm = SVC(kernel="linear")

svm.fit(x\_train, y\_train)

y\_predict\_svm = svm.predict(x\_test)

accuracy\_svm\_pre = accuracy\_score(y\_predict\_svm,y\_test)

#Neural Network before PCA

nnc=MLPClassifier(hidden\_layer\_sizes=(7), activation="relu", max\_iter=100000)

nnc.fit(x\_train, y\_train)

y\_predict\_nnc = nnc.predict(x\_test)

accuracy\_nnc\_pre = accuracy\_score(y\_predict\_nnc,y\_test)

#Random Forest before PCA

rfc = RandomForestClassifier(n\_estimators=50)

rfc.fit(x\_train, y\_train)

y\_predict\_rfc = rfc.predict(x\_test)

accuracy\_rfc\_pre = accuracy\_score(y\_predict\_rfc,y\_test)

#performing PCA

features\_temp = features

label\_temp = label

scaler= StandardScaler()

scaler.fit(features\_temp)

features\_temp = scaler.transform(features\_temp)

pca = PCA(n\_components=15)

principal\_components= pca.fit\_transform(features\_temp)

#split again into train test after pca (8:2)

y = pd.DataFrame(label\_temp)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(principal\_components, label\_temp, test\_size=0.2, random\_state=0, stratify = y)

#support vector after PCA

svm.fit(x\_train, y\_train)

y\_predict\_svm\_pca = svm.predict(x\_test)

accuracy\_svm\_post = accuracy\_score(y\_predict\_svm\_pca,y\_test)

#Neural Network after PCA

nnc.fit(x\_train, y\_train)

y\_predict\_nnc\_pca = nnc.predict(x\_test)

accuracy\_nnc\_post = accuracy\_score(y\_predict\_nnc\_pca,y\_test)

#Random Forest after PCA

rfc.fit(x\_train, y\_train)

y\_predict\_rfc\_pca = rfc.predict(x\_test)

accuracy\_rfc\_post = accuracy\_score(y\_predict\_rfc\_pca,y\_test)

#comparison

print('For Support Vector Machine(svm) Pre-Pca Accuracy :',accuracy\_svm\_pre,', Post-PCA Accuracy:',accuracy\_svm\_post)

if(accuracy\_svm\_pre>accuracy\_svm\_post):

  print('So PRE-PCA svm has higher accuracy')

elif(accuracy\_svm\_pre<accuracy\_svm\_post):

  print('So POST-PCA svm has higher accuracy')

else:

  print('Both PRE-PCA and POST-PCA svm have equal accuracy')

print('.......................................................................................')

print('For Neural Network(nnc) Pre-Pca Accuracy :',accuracy\_nnc\_pre,', Post-PCA Accuracy:',accuracy\_nnc\_post)

if(accuracy\_nnc\_pre>accuracy\_nnc\_post):

  print('So PRE-PCA nnc has higher accuracy')

elif(accuracy\_nnc\_pre<accuracy\_nnc\_post):

  print('So POST-PCA nnc has higher accuracy')

else:

  print('Both PRE-PCA and POST-PCA nnc have equal accuracy')

print('.......................................................................................')

print('For Random Forest(rfc) Pre-Pca Accuracy :',accuracy\_rfc\_pre,', Post-PCA Accuracy:',accuracy\_rfc\_post)

if(accuracy\_rfc\_pre>accuracy\_rfc\_post):

  print('So PRE-PCA rfc has higher accuracy')

elif(accuracy\_rfc\_pre<accuracy\_rfc\_post):

  print('So POST-PCA rfc has higher accuracy')

else:

  print('Both PRE-PCA and POST-PCA rfc have equal accuracy')

print('.......................................................................................')

#bar graph

index = np.arange(3)

bar\_width = 0.35

fig, ax = plt.subplots(figsize =(8, 12))

ax.bar(index, [accuracy\_svm\_pre,accuracy\_nnc\_pre,accuracy\_rfc\_pre], bar\_width,label="PRE-PCA")

ax.bar(index+bar\_width, [accuracy\_svm\_post,accuracy\_nnc\_post,accuracy\_rfc\_post],bar\_width, label="POST-PCA")

ax.set\_xlabel('Model Name')

ax.set\_ylabel('Accuracy')

ax.set\_title('Accuracy Comparison before and after PCA')

ax.set\_xticks(index + bar\_width/2 )

ax.set\_xticklabels(["SVM", "NNC", "RFC"])

ax.legend()

plt.show()