```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sbn
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.impute import SimpleImputer
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

tr=pd.read_csv('/content/heart failur classification dataset.csv')
tr.head()
tr.shape
tr.isnull().sum()
tr=tr[tr['ejection_fraction'].notnull()]
tr.shape
from sklearn.impute import SimpleImputer
impute=SimpleImputer(missing_values=np.nan,strategy='mean')
impute.fit(tr[['diabetes']])
tr[['diabetes']]=impute.transform(tr[['diabetes']])
impute.fit(tr[['serum_sodium']])
tr[['serum_sodium']]=impute.transform(tr[['serum_sodium']])
impute.fit(tr[['time']])
tr[['time']]=impute.transform(tr[['time']])
impute.fit(tr[['serum_creatinine']])
tr[['serum_creatinine']]=impute.transform(tr[['serum_creatinine']])
```

```
tr = tr.drop(['Unnamed: 0'], axis = 1)

tr.isnull().sum()

tr.info()

print(tr['anaemia'].unique())

print(tr['DEATH_EVENT'].unique())

tr['sex']=tr['sex'].map({'Male':1,'Female':0})

tr['smoking']=tr['smoking'].map({'Yes':1,'No':0})

tr.info()

xcor=tr.corr()

sbn.heatmap(xcor,cmap='YlGnBu')

features=['age','anaemia','creatinine_phosphokinase','diabetes','ejection_fraction','high_blood_pressure','pl
atelets','serum_creatinine','serum_sodium','sex','smoking','time','DEATH_EVENT']

label=['DEATH_EVENT']

x=tr[features]

y=tr[label]

from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=1)

print(x_train.shape)

print(x_test.shape)

from sklearn.preprocessing import MinMaxScaler

scaler=MinMaxScaler(copy=True, feature_range=(0,1))

scaler.fit(x_train)

x_train_scaled = scaler.transform(x_train)

x_test_scaled = scaler.transform(x_test)


svm = SVC(kernel="linear")

svm.fit(x_train, y_train)

prePcaSvmTrainScore = round(svm.score(x_train_scaled, y_train)*100,2)

prePcaSvmTestScore = round(svm.score(x_test_scaled, y_test)*100,2)

print("Training accuracy of pre PCA SVM is " + str(prePcaSvmTrainScore))
```

```
print("Testing accuracy of pre PCA SVM is "+ str(prePcaSvmTestScore))


rfc = RandomForestClassifier(n_estimators=50)
rfc.fit(x_train, y_train)
prePcaRfcTrainScore = round(rfc.score(x_train, y_train)*100,2)
prePcaRfcTestScore = round(rfc.score(x_test, y_test)*100,2)
print("Training accuracy of pre PCA Random Forest is " + str(prePcaRfcTrainScore))
print("Testing accuracy of pre PCA Random Forest is "+ str(prePcaRfcTestScore))


nnc=MLPClassifier(hidden_layer_sizes=(10), activation="relu", max_iter=10000)
nnc.fit(x_train, y_train)
prePcaNncTrainScore = round(nnc.score(x_train, y_train)*100,2)
prePcaNncTestScore = round(nnc.score(x_test, y_test)*100,2)
print("Training accuracy of pre PCA of NNC is " + str(prePcaNncTrainScore))
print("Testing accuracy of pre PCA NNC is "+ str(prePcaNncTestScore))


pd.options.mode.chained_assignment = None  # default='warn'
scaler= StandardScaler()
scaledData = scaler.fit_transform(tr)
n = int (tr.shape[1]/2)
pca = PCA(n_components=n)
principal_components= pca.fit_transform(scaledData)
principal_df = pd.DataFrame(data=principal_components, columns=["principle component 1", "principle component 2","principle component 3","principle component 4","principle component 5","principle component 6"])
main_df=pd.concat([principal_df, tr[["DEATH_EVENT"]]], axis=1)
main_df.head()


X= main_df.drop("DEATH_EVENT" , axis=1)
y= main_df["DEATH_EVENT"]
y=y.astype('int')
```

```
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=4)
svm = SVC(kernel="linear")
svm.fit(x_train, y_train)
postPcaSvmTrainScore = round(svm.score(x_train, y_train)*100,2)
postPcaSvmTestScore = round(svm.score(x_test, y_test)*100,2)
print("Training accuracy of SVM post PCA is " + str(postPcaSvmTrainScore))
print("Testing accuracy of SVM post PCA is "+ str(postPcaSvmTestScore))

rfc = RandomForestClassifier(n_estimators=50)
rfc.fit(x_train, y_train)
postPcaRfcTrainScore = round(rfc.score(x_train, y_train)*100,2)
postPcaRfcTestScore = round(rfc.score(x_test, y_test)*100,2)
print("Training accuracy of Random Forest post PCA is " + str(postPcaRfcTrainScore))
print("Testing accuracy of Random Forest post PCA is "+ str(postPcaRfcTestScore))

nnc=MLPClassifier(hidden_layer_sizes=(10), activation="relu", max_iter=10000)
nnc.fit(x_train, y_train)
postPcaNncTrainScore = round(nnc.score(x_train, y_train)*100,2)
postPcaNncTestScore = round(nnc.score(x_test, y_test)*100,2)
print("Training accuracy of NNC post PCA is " + str(postPcaNncTrainScore))
print("Testing accuracy of NNC post PCA is "+ str(postPcaNncTestScore))

tr = [[prePcaSvmTestScore, prePcaRfcTestScore, prePcaNncTestScore],
[postPcaSvmTestScore, postPcaRfcTestScore, postPcaNncTestScore],
[prePcaSvmTrainScore, prePcaRfcTrainScore, prePcaNncTrainScore],
[postPcaSvmTrainScore, postPcaRfcTrainScore, postPcaNncTrainScore]]
X = np.arange(3)

fig = plt.figure()
```

```
ax = fig.add_axes([0,0,1,1])

ax.bar(X + 0.00, tr[0], color = 'blue', width = 0.20)
ax.bar(X + 0.20, tr[1], color = 'green', width = 0.20)
ax.bar(X + 0.40, tr[2], color = 'red', width = 0.20)
ax.bar(X + 0.60, tr[3], color = 'maroon', width = 0.20)

ax.legend(labels=['Pre PCA Test', 'Post PCA Test','Pre PCA Train','Post PCA Train'])

plt.ylabel('Percentage')
plt.xlabel('SVM (left), Random Forest (middle), NNC (right)')
plt.title('Test Score')
```