Python Implementation

In [1]:

```
#Import necessory Libraries

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import matplotlib.pyplot as plt
#import plotly

import warnings
warnings.filterwarnings('ignore')
```

In [14]:

```
# Get the CSV data here and print head
df=pd.read_csv('https://raw.githubusercontent.com/training-ml/Files/main/breast%20cancer.cs
df.head(25)
```

Out[14]:

id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mea
842302	M	17.990	10.38	122.80	1001.0	0.118
842517	M	20.570	17.77	132.90	1326.0	0.084
84300903	M	19.690	21.25	130.00	1203.0	0.109
84348301	M	11.420	20.38	77.58	386.1	0.142
84358402	M	20.290	14.34	135.10	1297.0	0.100
843786	M	12.450	15.70	82.57	477.1	0.127
844359	M	18.250	19.98	119.60	1040.0	0.094
84458202	M	13.710	20.83	90.20	577.9	0.118
844981	M	13.000	21.82	87.50	519.8	0.127
84501001	M	12.460	24.04	83.97	475.9	0.118
845636	M	16.020	23.24	102.70	797.8	0.082
84610002	M	15.780	17.89	103.60	781.0	0.097
846226	M	19.170	24.80	132.40	1123.0	0.097
846381	M	15.850	23.95	103.70	782.7	0.084
84667401	M	13.730	22.61	93.60	578.3	0.113
84799002	M	14.540	27.54	96.73	658.8	0.113
848406	M	14.680	20.13	94.74	684.5	0.098
84862001	M	16.130	20.68	108.10	798.8	0.117
849014	M	19.810	22.15	130.00	1260.0	0.098
8510426	B	13.540	14.36	87.46	566.3	0.097
8510653	B	13.080	15.71	85.63	520.0	0.107
8510824	B	9.504	12.44	60.34	273.9	0.102
8511133	M	15.340	14.26	102.50	704.4	0.107
851509	M	21.160	23.04	137.20	1404.0	0.094
852552	M	16.650	21.38	110.00	904.6	0.112

25 rows × 32 columns

In [3]:

```
#print summary
print('shape     ------------->',df.shape)
print('Each column and data type and its count','\n')
print(df.info())
```

```
shape     -------------> (569, 32)
Each column and data type and its count

<class 'pandas.core.frame.DataFrame'>
Int64Index: 569 entries, 842302 to 92751
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   diagnosis                569 non-null    object 
 1   radius_mean              569 non-null    float64
 2   texture_mean             569 non-null    float64
 3   perimeter_mean           569 non-null    float64
 4   area_mean                569 non-null    float64
 5   smoothness_mean          569 non-null    float64
 6   compactness_mean         569 non-null    float64
 7   concavity_mean           569 non-null    float64
 8   concave points_mean      569 non-null    float64
 9   symmetry_mean            569 non-null    float64
 10  fractal_dimension_mean   569 non-null    float64
 11  radius_se                569 non-null    float64
 12  texture_se               569 non-null    float64
 13  perimeter_se             569 non-null    float64
 14  area_se                  569 non-null    float64
 15  smoothness_se            569 non-null    float64
 16  compactness_se           569 non-null    float64
 17  concavity_se             569 non-null    float64
 18  concave points_se        569 non-null    float64
 19  symmetry_se              569 non-null    float64
 20  fractal_dimension_se     569 non-null    float64
 21  radius_worst             569 non-null    float64
 22  texture_worst            569 non-null    float64
 23  perimeter_worst          569 non-null    float64
 24  area_worst               569 non-null    float64
 25  smoothness_worst         569 non-null    float64
 26  compactness_worst        569 non-null    float64
 27  concavity_worst          569 non-null    float64
 28  concave points_worst     569 non-null    float64
 29  symmetry_worst           569 non-null    float64
 30  fractal_dimension_worst  569 non-null    float64
 31  Unnamed: 32              0 non-null      float64
dtypes: float64(31), object(1)
memory usage: 146.7+ KB
None
```

In [7]:

```
#DROP ALERT 1 : Unnamed :32 column has all nulls.safe to remove the column.
df=df.drop(['Unnamed: 32'],axis=1)
```

In [8]:

```
df.shape
```

Out[8]:

```
(569, 31)
```

In [9]:

```
#Dataframe statistics
df.describe()
```

Out[9]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactne
count	569.000000	569.000000	569.000000	569.000000	569.000000	5(
mean	14.127292	19.289649	91.969033	654.889104	0.096360	
std	3.524049	4.301036	24.298981	351.914129	0.014064	
min	6.981000	9.710000	43.790000	143.500000	0.052630	
25%	11.700000	16.170000	75.170000	420.300000	0.086370	
50%	13.370000	18.840000	86.240000	551.100000	0.095870	
75%	15.780000	21.800000	104.100000	782.700000	0.105300	
max	28.110000	39.280000	188.500000	2501.000000	0.163400	

8 rows × 30 columns

seems no other cols have nulls. It's safe to poceed

As we can see each feature data scaled differently. Let's go ahead and scale the data

In [10]:

```
scaler= StandardScaler()
x=df.drop('diagnosis',axis=1)
x_scaled=scaler.fit_transform(x)
```

Principal component Analysis (PCA)

PCA is dimension reduction technique (Not feature selection technique)

PCA can be applied only on Features (not on target)

PCA can be applied when you have too many features and their correlation is not that significant with target.

PCA will also takes care of multicollinearity problem

In [11]:

```
pca=PCA()
pca.fit_transform(x_scaled)
```

Out[11]:

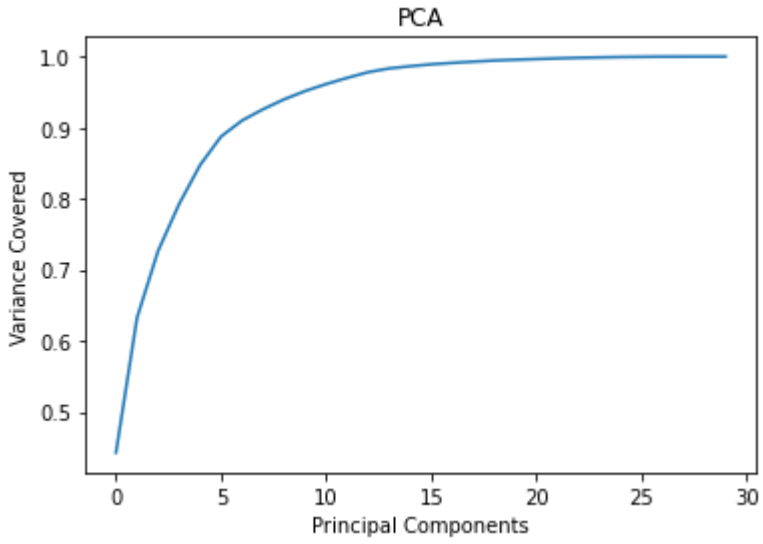
```
array([[ 9.19283683e+00,  1.94858307e+00, -1.12316616e+00, ...,
        -3.39144536e-02,  4.56477199e-02, -4.71692081e-02],
       [ 2.38780180e+00, -3.76817174e+00, -5.29292687e-01, ...,
         3.26241827e-02, -5.68742432e-03, -1.86787626e-03],
       [ 5.73389628e+00, -1.07517380e+00, -5.51747593e-01, ...,
         4.70258247e-02,  3.14589659e-03,  7.50534755e-04],
       ...,
       [ 1.25617928e+00, -1.90229671e+00,  5.62730526e-01, ...,
        -2.57775589e-03,  6.70621179e-03,  3.77041667e-03],
       [ 1.03747941e+01,  1.67201011e+00, -1.87702933e+00, ...,
        -6.80863833e-02, -8.41632764e-02, -2.37828222e-02],
       [-5.47524330e+00, -6.70636791e-01,  1.49044308e+00, ...,
        -9.51587894e-03, -6.09131090e-02, -1.94755854e-02]])
```

In [12]:

```
 # Let's plot scree plot to check the best component
plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Principal Components')
plt.ylabel('Variance Covered')
plt.title('PCA')
plt.show
```

Out[12]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Around 13 Principal components are able to explain >95% variance. Its safe to consider starting 13 PC's

In [13]:

```
pca=PCA(n_components=13)
new_pcomp=pca.fit_transform(x_scaled)
princi_comp=pd.DataFrame(new_pcomp,columns=['PC1','PC2','PC3','PC4','PC5','PC6','PC7','PC8'
princi_comp
```

Out[13]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	
0	9.192837	1.948583	-1.123166	3.633731	-1.195110	1.411424	2.159370	-0.398406	-0.1
1	2.387802	-3.768172	-0.529293	1.118264	0.621775	0.028656	0.013358	0.240991	-0.7
2	5.733896	-1.075174	-0.551748	0.912083	-0.177086	0.541452	-0.668166	0.097372	0.0
3	7.122953	10.275589	-3.232790	0.152547	-2.960878	3.053422	1.429911	1.059566	-1.4
4	3.935302	-1.948072	1.389767	2.940639	0.546747	-1.226495	-0.936213	0.636376	-0.2
...	...	...	...	...	...	...	...	...	
564	6.439315	-3.576817	2.459487	1.177314	-0.074824	-2.375193	-0.596130	-0.035472	0.9
565	3.793382	-3.584048	2.088476	-2.506028	-0.510723	-0.246710	-0.716326	-1.113359	-0.1
566	1.256179	-1.902297	0.562731	-2.089227	1.809991	-0.534447	-0.192758	0.341887	0.3
567	10.374794	1.672010	-1.877029	-2.356031	-0.033742	0.567936	0.223081	-0.280243	-0.5
568	-5.475243	-0.670637	1.490443	-2.299157	-0.184703	1.617837	1.698952	1.046354	0.3

569 rows × 13 columns

In [15]:

```
# Replace label column (diagnosis) into binary codes
df['diagnosis']=df['diagnosis'].replace({'M':1,'B':0})
y=df['diagnosis']
```

In [16]:

```
# Data split into train and test
x_train,x_test,y_train,y_test=train_test_split(princi_comp,y,test_size=0.25,random_state=35
```

In [19]:

```
def print_score(clf,x_train,x_test,y_tain,y_test,train=True):
    if train:
        y_pred=clf.predict(x_train)

        print("\n===============Train Result=====================")

        print(f"Accuracy score:{accuracy_score(y_train,y_pred)* 100:.2f}%")


    elif train==False:
        pred=clf.predict(x_test)

        print('\n==================Test Result=================================')
        print(f"Accuracy score:{accuracy_score(y_test,pred)*100:.2f}%")

        print('\n \n Test classification Report \n' ,classification_report(y_test,pred,digi
```

In [20]:

```
from sklearn.svm import SVC
svc=SVC()
#svc model training and printing train and test score
svc.fit(x_train,y_train)
# call the function and pass dataset to check train and test score

print_score(svc,x_train,x_test,y_train,y_test,train=True)
print_score(svc,x_train,x_test,y_train,y_test,train=False)
```

```
===============Train Result=====================
Accuracy score:98.12%

===================Test Result================================
Accuracy score:97.90%


 Test classification Report
              precision    recall  f1-score   support

           0       0.97      1.00      0.98        93
           1       1.00      0.94      0.97        50

    accuracy                           0.98       143
   macro avg       0.98      0.97      0.98       143
weighted avg       0.98      0.98      0.98       143
```

In [21]:

```
from sklearn.ensemble import GradientBoostingClassifier
gbdt=GradientBoostingClassifier()
#GBDT model training and printing train and test score
gbdt.fit(x_train,y_train)

# call the function and pass dataset to check train and test score
print_score(gbdt,x_train,x_test,y_train,y_test,train=True)
print_score(gbdt,x_train,x_test,y_train,y_test,train=False)
```

```
===============Train Result=====================
Accuracy score:100.00%

===================Test Result================================
Accuracy score:94.41%


 Test classification Report
              precision    recall  f1-score   support

           0       0.95      0.97      0.96        93
           1       0.94      0.90      0.92        50

    accuracy                           0.94       143
   macro avg       0.94      0.93      0.94       143
weighted avg       0.94      0.94      0.94       143
```

In [22]:

```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()

#Random Forest model training and printing train and test score
rf.fit(x_train,y_train)

# call the function and pass dataset to check train and test score
print_score(rf,x_train,x_test,y_train,y_test,train=True)
print_score(rf,x_train,x_test,y_train,y_test,train=False)
```

```
==============Train Result=====================
Accuracy score:100.00%

==================Test Result================================
Accuracy score:96.50%


 Test classification Report
              precision    recall  f1-score   support

           0       0.98      0.97      0.97        93
           1       0.94      0.96      0.95        50

    accuracy                           0.97       143
   macro avg       0.96      0.96      0.96       143
weighted avg       0.97      0.97      0.97       143
```

In [23]:

```
from sklearn.model_selection import GridSearchCV
```

In [24]:

```
# C     = It is a hypermeter in SVM to cntrol error. How much error we can allow.
#         Low C means allowing less number of error/s and
#          Large C means allowing more nuber of errors.

# gamma =Gamma decides that how much curvature we want in a decision boundary.Gamma high me
#        Gamma Low means Less curvature.

param_grid={'C' : [1,5,10,20],
            'gamma': [0.001,0.01,0.02,0.002]
            }
```

In [25]:

```
gridsearch=GridSearchCV(svc,param_grid)
gridsearch.fit(x_train,y_train)

# best parms
gridsearch.best_params_
```

Out[25]:

```
{'C': 5, 'gamma': 0.01}
```

In [26]:

```
# SVC Model Training and printing train and test score(post param update)

svc = SVC(C= 7, gamma =0.001)
svc.fit(x_train,y_train)

#Call the function and pass dataset to check train and test score

print_score(svc,x_train,x_test,y_train,y_test,train=True)
print_score(svc,x_train,x_test,y_train,y_test,train=False)
```

```
==============Train Result=====================
Accuracy score:97.65%

==================Test Result================================
Accuracy score:97.90%


 Test classification Report
              precision    recall  f1-score   support

           0       0.97      1.00      0.98        93
           1       1.00      0.94      0.97        50

    accuracy                           0.98       143
   macro avg       0.98      0.97      0.98       143
weighted avg       0.98      0.98      0.98       143
```

Creating Pipeline

In real world the final model is built with popeline. We work on all preprocessing steps, do EDA, make analysis etc.Once we find all the hyperparameter and feature selection techniquesetc, we use the main techniques and create pipeline.This will be clean and better flow of data through series of sequences.

In [28]:

```
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
```

In [29]:

```
df=pd.read_csv('https://raw.githubusercontent.com/training-ml/Files/main/breast%20cancer.cs
```

In [30]:

```
x=df.drop(['Unnamed: 32','diagnosis'],axis=1)
y=df.diagnosis
x_train,x_test,y_train,y_test=train_test_split(
x,y,test_size=0.25,random_state=355)
```

In [32]:

```
pipe=Pipeline([('Scaler',StandardScaler()),    # fit_transform
               ('PCA' ,PCA(n_components=13)),    # fit_transform
               ('SVM',SVC(C=7,gamma=0.01))])    # ONLY fit
```

In [33]:

```
pipe.fit(x_train,y_train)
```

Out[33]:

```
Pipeline(steps=[('Scaler', StandardScaler()), ('PCA', PCA(n_components=13)),
                ('SVM', SVC(C=7, gamma=0.01))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [34]:

```
y_pred=pipe.predict(x_test)
```

In [35]:

```
accuracy_score(y_test,y_pred)
```

Out[35]:

```
0.9790209790209791
```

- key points . you need to know the execution sequence(example- Imputation techniques should be applied before standard scaler and then PCA

. you cannot use pipeline for plotting graphs and analysis.

. Analysis can be done before creating a pipeline

. Do not use unnecessary methods in the pipeline

you can also use ny encoding/imputation techniques in the pipeline like.

.('Simple Imputer',SinpleImputer(strategy='mean')),#fit_transform

.('Ohe',OneHotEncoder(handle_unknown='ignore')), #fit_transform

In [ ]: