14_Feb_EDA

February 14, 2025

1 Name: Tufan Kundu

2 Reg no: 24MDT0184

3 EDA lab

3.1 14 February

3.2 PCA on Iris Dataset

3.2.1 Importing the necessary libraries

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

3.2.2 Loading the dataset

[2]:
```
iris = load_iris()

## Creating dataframe with feature names
df = pd.DataFrame(iris.data,columns = iris.feature_names)
df['target'] = iris.target

df
```

[2]:
```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                  5.1               3.5                1.4               0.2
1                  4.9               3.0                1.4               0.2
2                  4.7               3.2                1.3               0.2
3                  4.6               3.1                1.5               0.2
4                  5.0               3.6                1.4               0.2
..                 …                 …                  …                 …
```

```
145                6.7               3.0                5.2               2.3
146                6.3               2.5                5.0               1.9
147                6.5               3.0                5.2               2.0
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8

     target
0         0
1         0
2         0
3         0
4         0
..      …
145       2
146       2
147       2
148       2
149       2

[150 rows x 5 columns]
```

[3]:
```python
## Checking the shape of the data
df.shape
```

[3]: (150, 5)

[4]:
```python
df.info()
```

```
&lt;class 'pandas.core.frame.DataFrame'&gt;
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
 4   target             150 non-null    int32
dtypes: float64(4), int32(1)
memory usage: 5.4 KB
```

[5]:
```python
df.describe()
```

[5]:
```
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.057333           3.758000
std             0.828066          0.435866           1.765298
min             4.300000          2.000000           1.000000
```

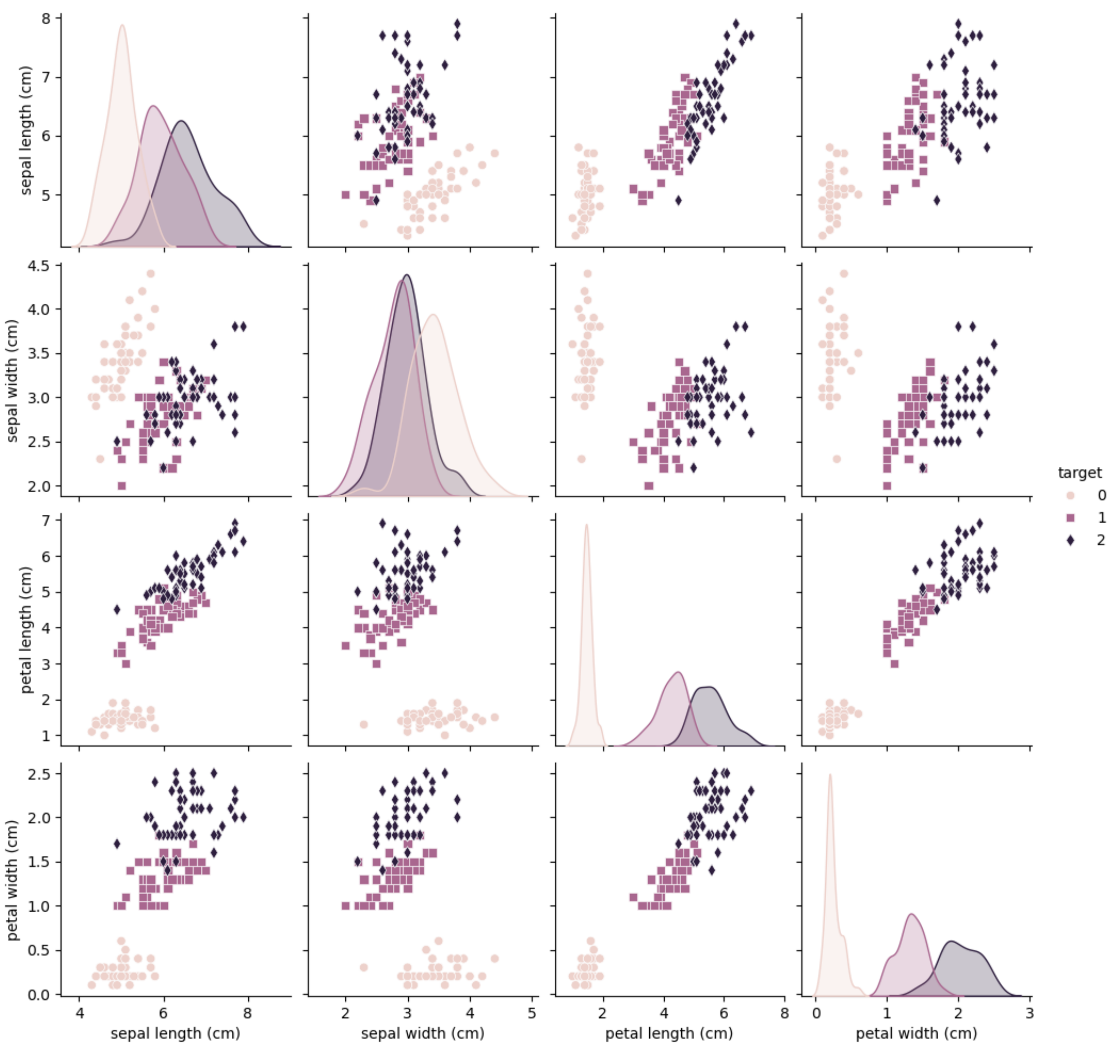
```
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
max             7.900000          4.400000           6.900000

       petal width (cm)      target
count        150.000000  150.000000
mean           1.199333    1.000000
std            0.762238    0.819232
min            0.100000    0.000000
25%            0.300000    0.000000
50%            1.300000    1.000000
75%            1.800000    2.000000
max            2.500000    2.000000
```

3.2.3 Visualizing the dataset with pairplot

[6]:
```
plt.figure(figsize=(8,6))
sns.pairplot(df,hue='target',markers = ["o","s","d"])
plt.suptitle("Pairplot of the Iris dataset",y = 1.02)
plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```

Pairplot of the Iris dataset



3.2.4 extracting the feature and target

[7]:
```
features = iris.feature_names
x = df.loc[:,features].values
y = df.loc[:,['target']].values

## Standardising the features
scaler = StandardScaler()
x_std = scaler.fit_transform(x)

print("Mean of standardized features(should be close to 0):",np.mean(x_std,axis␣
↪= 0))
print("Standard deviation (should be 1):",np.std(x_std,axis = 0))
```

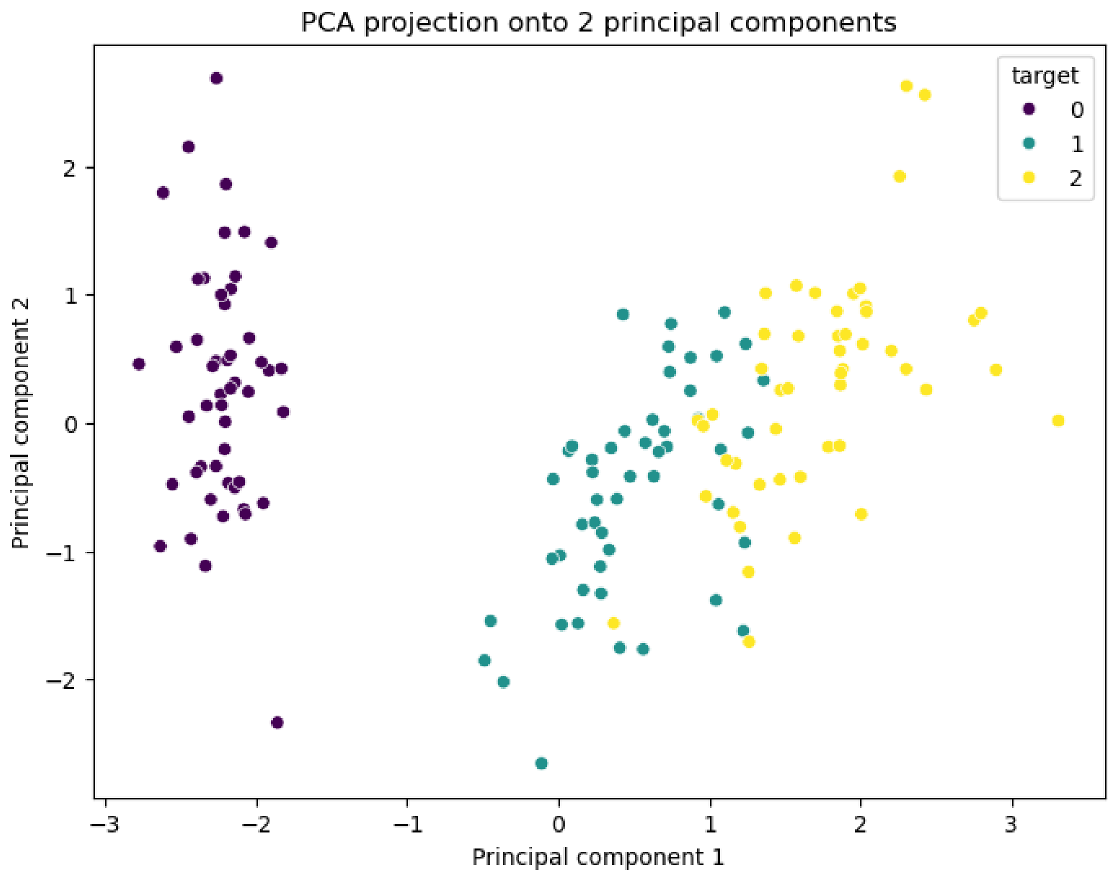
```
Mean of standardized features(should be close to 0): [-1.69031455e-15
-1.84297022e-15 -1.69864123e-15 -1.40924309e-15]
Standard deviation (should be 1): [1. 1. 1. 1.]
```

[8]:
```python
### Initializing PCA to reduce the data to 2 components

pca = PCA(n_components=2)
principalcomponents = pca.fit_transform(x_std)

## Creating dataframe for the two principal components
principaldf = pd.DataFrame(data = principalcomponents, columns=['PC1','PC2'])

## Concatenate the target variable for plotting

final_df = pd.concat([principaldf,df[['target']]],axis = 1)

final_df
```

[8]:
```
          PC1       PC2  target
0   -2.264703  0.480027       0
1   -2.080961 -0.674134       0
2   -2.364229 -0.341908       0
3   -2.299384 -0.597395       0
4   -2.389842  0.646835       0
..        ...       ...     ...
145  1.870503  0.386966       2
146  1.564580 -0.896687       2
147  1.521170  0.269069       2
148  1.372788  1.011254       2
149  0.960656 -0.024332       2

[150 rows x 3 columns]
```

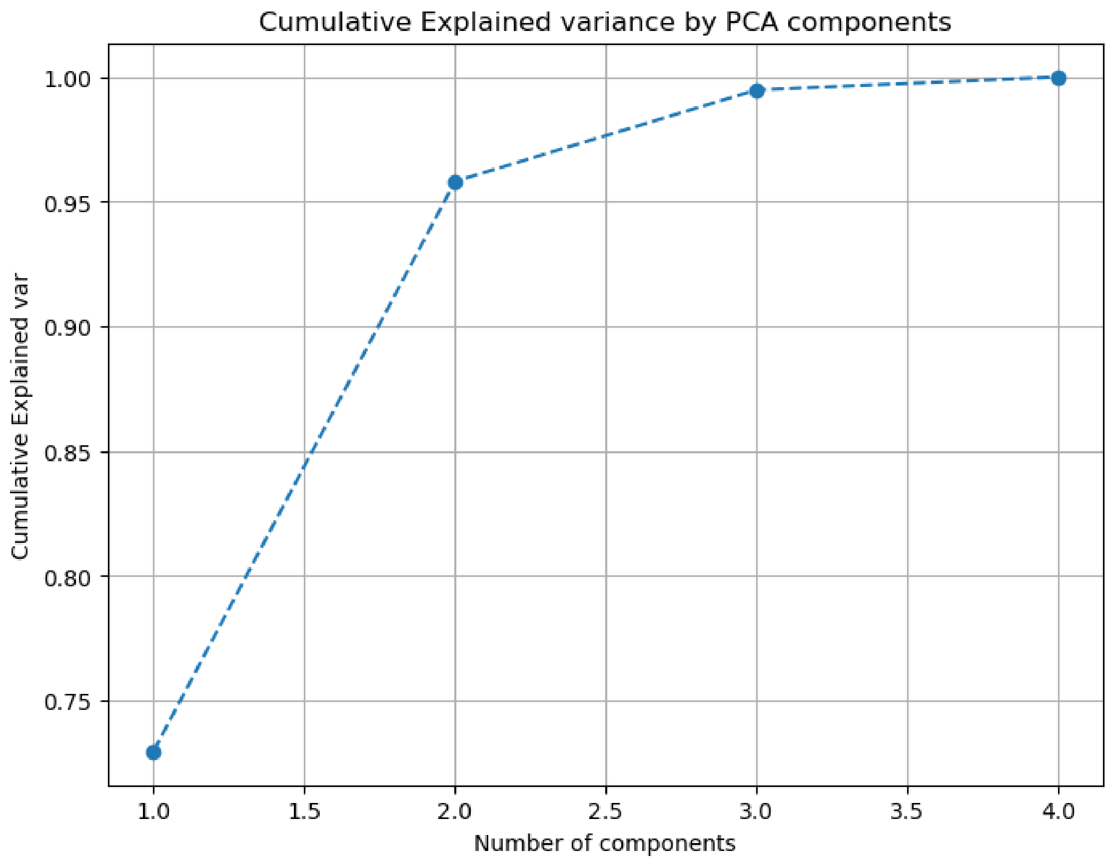
3.2.5 Visualizing the PCA results

[9]:
```python
plt.figure(figsize=(8,6))
sns.scatterplot(x='PC1',y = 'PC2', hue = 'target', data = final_df, palette =␣
↪'viridis')
plt.title("PCA projection onto 2 principal components")
plt.xlabel("Principal component 1")
plt.ylabel("Principal component 2")
plt.legend(title='target')
plt.show()
```



[10]:
```
## Analyzing the explained variance

print("Explained variance ratio for 2 components:")
print(pca.explained_variance_ratio_)

## cumulative explained variance with all components
pca_full = PCA().fit(x_std)
cum_var = np.cumsum(pca_full.explained_variance_ratio_)

plt.figure(figsize = (8,6))
plt.plot(np.arange(1,len(cum_var)+1),cum_var,marker = 'o', linestyle = '--')
plt.xlabel("Number of components")
plt.ylabel("Cumulative Explained var")
plt.title("Cumulative Explained variance by PCA components")
plt.grid(True)
plt.show()
```

```
Explained variance ratio for 2 components:
[0.72962445 0.22850762]
```

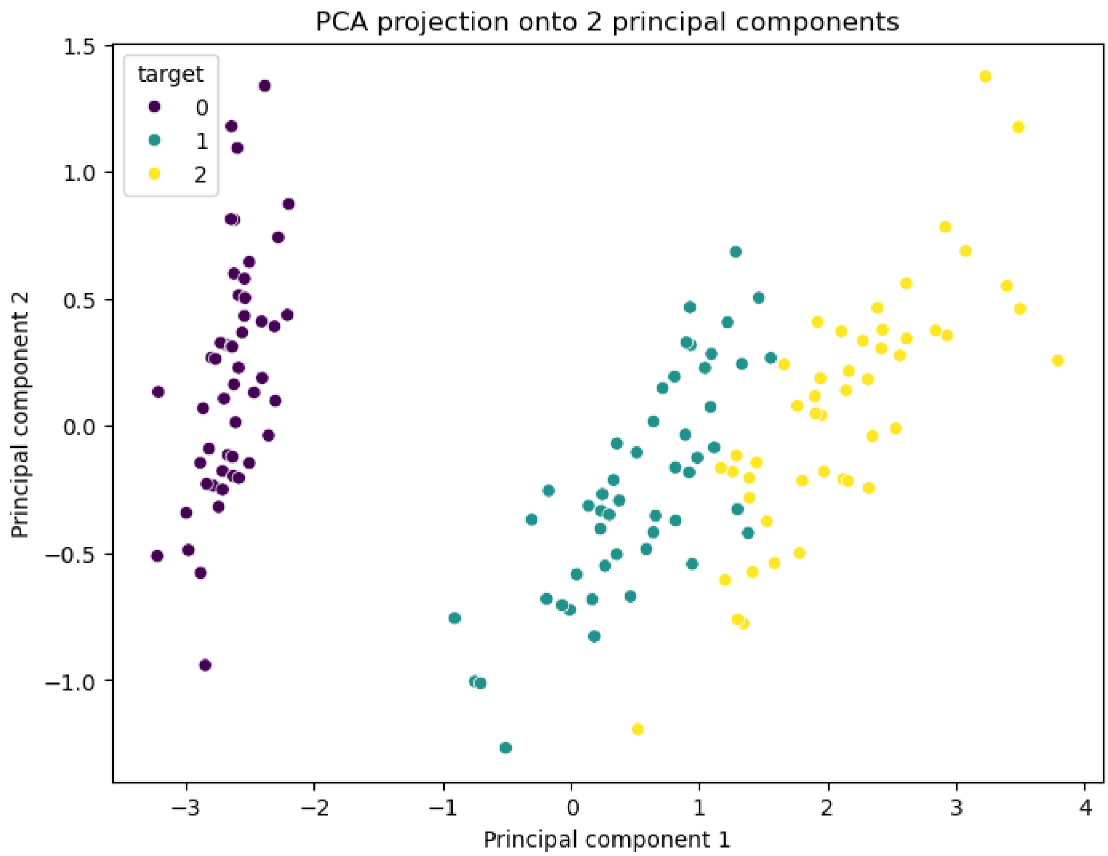


3.2.6 Without scaling the data

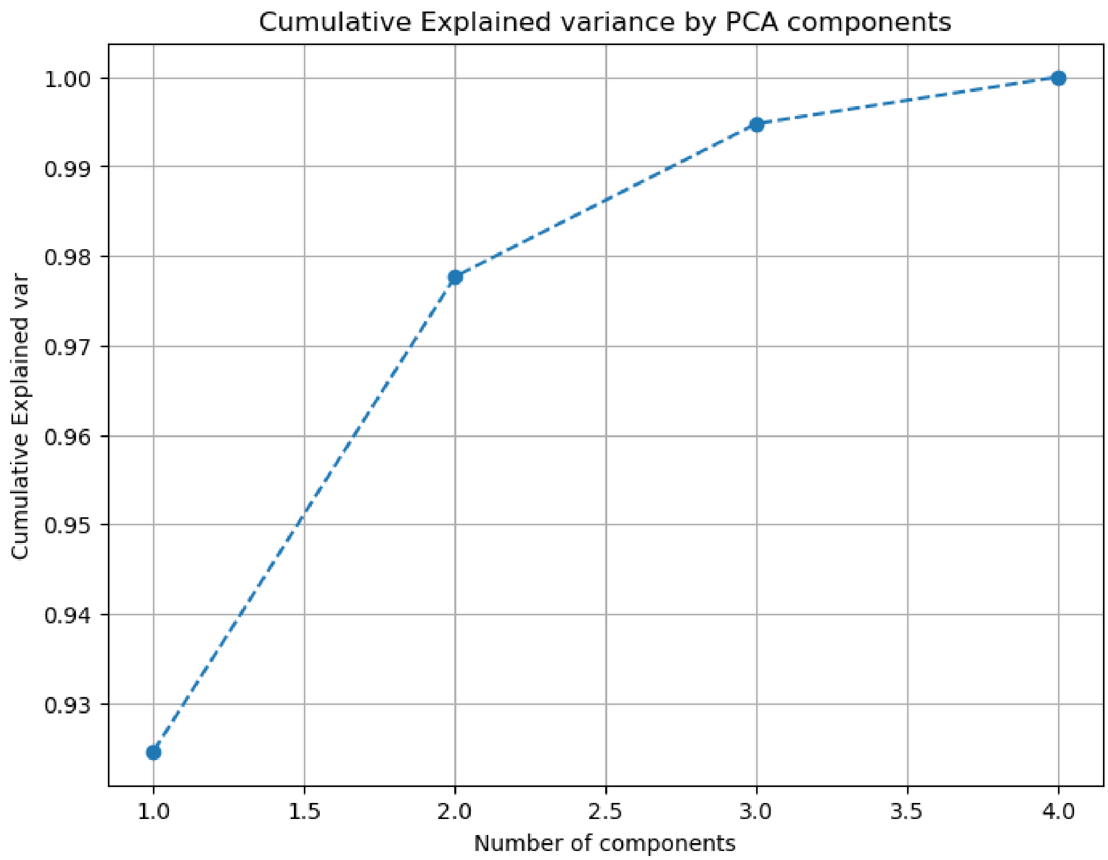
[11]:
```
features = iris.feature_names
x = df.loc[:,features].values
y = df.loc[:,['target']].values

### Initializing PCA to reduce the data to 2 components

pca = PCA(n_components=2)
principalcomponents = pca.fit_transform(x)

## Creating dataframe for the two principal components
principaldf = pd.DataFrame(data = principalcomponents, columns=['PC1','PC2'])

## Concatenate the target variable for plotting

final_df = pd.concat([principaldf,df[['target']]],axis = 1)

print(final_df)
```

```python
plt.figure(figsize=(8,6))
sns.scatterplot(x='PC1',y = 'PC2', hue = 'target', data = final_df, palette =␣
 ↪'viridis')
plt.title("PCA projection onto 2 principal components")
plt.xlabel("Principal component 1")
plt.ylabel("Principal component 2")
plt.legend(title='target')
plt.show()

## Analyzing the explained variance

print("Explained variance ratio for 2 components:")
print(pca.explained_variance_ratio_)

## cumulative explained variance with all components
pca_full = PCA().fit(x)
cum_var = np.cumsum(pca_full.explained_variance_ratio_)

plt.figure(figsize = (8,6))
plt.plot(np.arange(1,len(cum_var)+1),cum_var,marker = 'o', linestyle = '--')
plt.xlabel("Number of components")
plt.ylabel("Cumulative Explained var")
plt.title("Cumulative Explained variance by PCA components")
plt.grid(True)
plt.show()
```

```
          PC1       PC2  target
0   -2.684126  0.319397       0
1   -2.714142 -0.177001       0
2   -2.888991 -0.144949       0
3   -2.745343 -0.318299       0
4   -2.728717  0.326755       0
..        …         …     …
145  1.944110  0.187532       2
146  1.527167 -0.375317       2
147  1.764346  0.078859       2
148  1.900942  0.116628       2
149  1.390189 -0.282661       2

[150 rows x 3 columns]
```




```
Explained variance ratio for 2 components:
[0.92461872 0.05306648]
```



3.3 PCA on breast cancer dataset

[29]:
```
# importing the dataset
from sklearn.datasets import load_breast_cancer
bc = load_breast_cancer()

# creating dataframe with feature names
df_bc = pd.DataFrame(bc.data,columns = bc.feature_names)
df_bc['target'] = bc.target

df_bc
```

[29]:
```
     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0          17.99         10.38          122.80     1001.0          0.11840
1          20.57         17.77          132.90     1326.0          0.08474
2          19.69         21.25          130.00     1203.0          0.10960
3          11.42         20.38           77.58      386.1          0.14250
4          20.29         14.34          135.10     1297.0          0.10030
..           …             …               …          …                …
564        21.56         22.39          142.00     1479.0          0.11100
```

```
565      20.13        28.25        131.20    1261.0          0.09780
566      16.60        28.08        108.30     858.1          0.08455
567      20.60        29.33        140.10    1265.0          0.11780
568       7.76        24.54         47.92     181.0          0.05263

     mean compactness  mean concavity  mean concave points  mean symmetry  \
0             0.27760         0.30010              0.14710         0.2419
1             0.07864         0.08690              0.07017         0.1812
2             0.15990         0.19740              0.12790         0.2069
3             0.28390         0.24140              0.10520         0.2597
4             0.13280         0.19800              0.10430         0.1809
..                …              …                    …             …
564           0.11590         0.24390              0.13890         0.1726
565           0.10340         0.14400              0.09791         0.1752
566           0.10230         0.09251              0.05302         0.1590
567           0.27700         0.35140              0.15200         0.2397
568           0.04362         0.00000              0.00000         0.1587

     mean fractal dimension  …  worst texture  worst perimeter  worst area  \
0                   0.07871  …          17.33           184.60      2019.0
1                   0.05667  …          23.41           158.80      1956.0
2                   0.05999  …          25.53           152.50      1709.0
3                   0.09744  …          26.50            98.87       567.7
4                   0.05883  …          16.67           152.20      1575.0
..                      …  …            …              …           …
564                 0.05623  …          26.40           166.10      2027.0
565                 0.05533  …          38.25           155.00      1731.0
566                 0.05648  …          34.12           126.70      1124.0
567                 0.07016  …          39.42           184.60      1821.0
568                 0.05884  …          30.37            59.16       268.6

     worst smoothness  worst compactness  worst concavity  \
0             0.16220            0.66560           0.7119
1             0.12380            0.18660           0.2416
2             0.14440            0.42450           0.4504
3             0.20980            0.86630           0.6869
4             0.13740            0.20500           0.4000
..                …                 …               …
564           0.14100            0.21130           0.4107
565           0.11660            0.19220           0.3215
566           0.11390            0.30940           0.3403
567           0.16500            0.86810           0.9387
568           0.08996            0.06444           0.0000

     worst concave points  worst symmetry  worst fractal dimension  target
0                  0.2654          0.4601                  0.11890       0
1                  0.1860          0.2750                  0.08902       0
```

```
2               0.2430       0.3613                   0.08758      0
3               0.2575       0.6638                   0.17300      0
4               0.1625       0.2364                   0.07678      0
..                 ...          ...                      ...    ...
564             0.2216       0.2060                   0.07115      0
565             0.1628       0.2572                   0.06637      0
566             0.1418       0.2218                   0.07820      0
567             0.2650       0.4087                   0.12400      0
568             0.0000       0.2871                   0.07039      1

[569 rows x 31 columns]
```

[30]:

```
## checking the value count of each class
df_bc['target'].value_counts()
```

[30]:

```
target
1    357
0    212
Name: count, dtype: int64
```

[31]:

```
bc.target_names
```

[31]:

```
array(['malignant', 'benign'], dtype='<U9')
```

3.4 1 –> malignant

3.5 0 –> benign

[32]:

```
## Checking the shape of the data
df_bc.shape
```

[32]:

```
(569, 31)
```

[33]:

```
df_bc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
```

```
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  target                   569 non-null    int32
dtypes: float64(30), int32(1)
memory usage: 135.7 KB
```

[34]:
```
df_bc.describe()
```

[34]:
```
       mean radius  mean texture  mean perimeter    mean area  \
count   569.000000    569.000000      569.000000   569.000000
mean     14.127292     19.289649       91.969033   654.889104
std       3.524049      4.301036       24.298981   351.914129
min       6.981000      9.710000       43.790000   143.500000
25%      11.700000     16.170000       75.170000   420.300000
50%      13.370000     18.840000       86.240000   551.100000
75%      15.780000     21.800000      104.100000   782.700000
max      28.110000     39.280000      188.500000  2501.000000

       mean smoothness  mean compactness  mean concavity  mean concave points  \
count       569.000000        569.000000      569.000000           569.000000
mean          0.096360          0.104341        0.088799             0.048919
std           0.014064          0.052813        0.079720             0.038803
min           0.052630          0.019380        0.000000             0.000000
25%           0.086370          0.064920        0.029560             0.020310
50%           0.095870          0.092630        0.061540             0.033500
75%           0.105300          0.130400        0.130700             0.074000
max           0.163400          0.345400        0.426800             0.201200
```

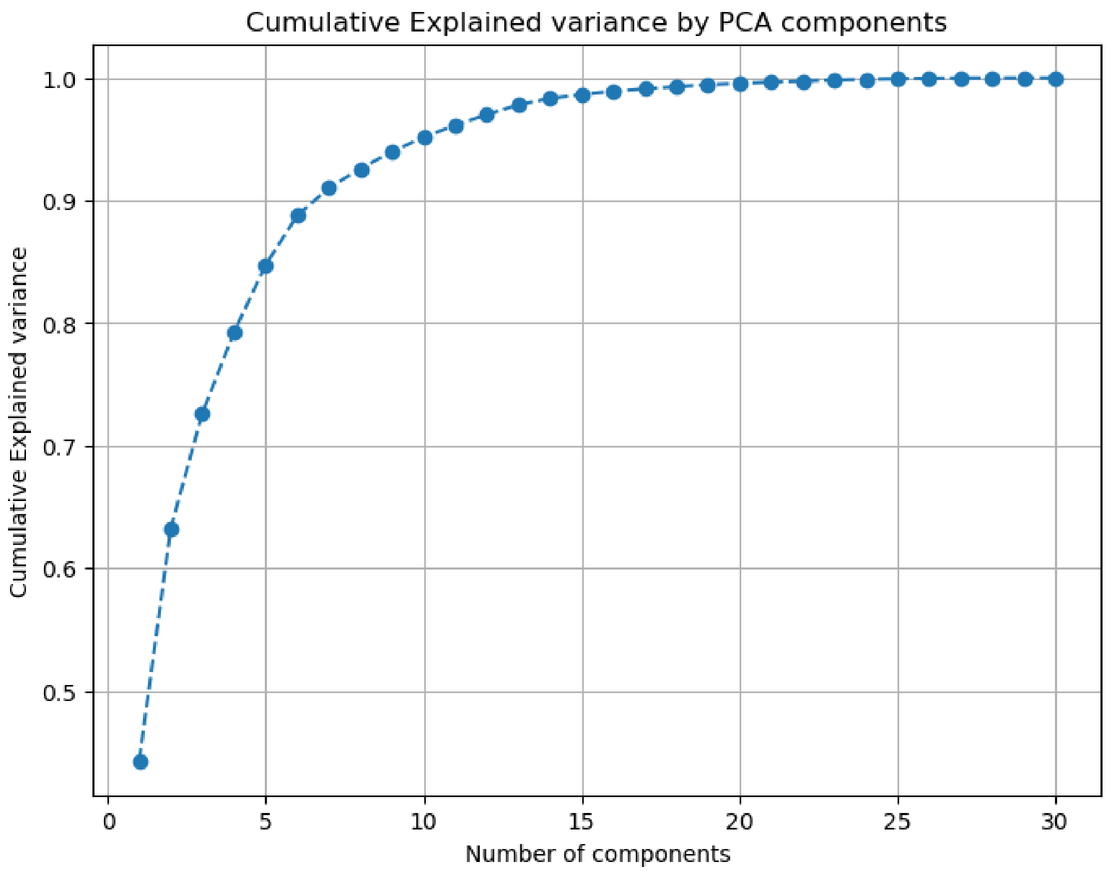
```
       mean symmetry  mean fractal dimension  …  worst texture  \
count     569.000000              569.000000  …     569.000000
mean        0.181162                0.062798  …      25.677223
std         0.027414                0.007060  …       6.146258
min         0.106000                0.049960  …      12.020000
25%         0.161900                0.057700  …      21.080000
50%         0.179200                0.061540  …      25.410000
75%         0.195700                0.066120  …      29.720000
max         0.304000                0.097440  …      49.540000

       worst perimeter   worst area  worst smoothness  worst compactness  \
count       569.000000   569.000000        569.000000         569.000000
mean        107.261213   880.583128          0.132369           0.254265
std          33.602542   569.356993          0.022832           0.157336
min          50.410000   185.200000          0.071170           0.027290
25%          84.110000   515.300000          0.116600           0.147200
50%          97.660000   686.500000          0.131300           0.211900
75%         125.400000  1084.000000          0.146000           0.339100
max         251.200000  4254.000000          0.222600           1.058000

       worst concavity  worst concave points  worst symmetry  \
count       569.000000            569.000000      569.000000
mean          0.272188              0.114606        0.290076
std           0.208624              0.065732        0.061867
min           0.000000              0.000000        0.156500
25%           0.114500              0.064930        0.250400
50%           0.226700              0.099930        0.282200
75%           0.382900              0.161400        0.317900
max           1.252000              0.291000        0.663800

       worst fractal dimension      target
count               569.000000  569.000000
mean                  0.083946    0.627417
std                   0.018061    0.483918
min                   0.055040    0.000000
25%                   0.071460    0.000000
50%                   0.080040    1.000000
75%                   0.092080    1.000000
max                   0.207500    1.000000

[8 rows x 31 columns]
```

3.5.1 Extracting feature and target

[35]:
```
features = bc.feature_names
x = df_bc.loc[:,features].values
y = df_bc.loc[:,['target']].values

## Standardising the features
scaler = StandardScaler()
x_std = scaler.fit_transform(x)

print("Mean of standardized features(should be close to 0):",np.mean(x_std,axis␣
↪= 0))
print("Standard deviation (should be 1):",np.std(x_std,axis = 0))
```

```
Mean of standardized features(should be close to 0): [-3.16286735e-15
-6.53060890e-15 -7.07889127e-16 -8.79983452e-16
  6.13217737e-15 -1.12036918e-15 -4.42138027e-16  9.73249991e-16
 -1.97167024e-15 -1.45363120e-15 -9.07641468e-16 -8.85349205e-16
  1.77367396e-15 -8.29155139e-16 -7.54180940e-16 -3.92187747e-16
  7.91789988e-16 -2.73946068e-16 -3.10823423e-16 -3.36676596e-16
 -2.33322442e-15  1.76367415e-15 -1.19802625e-15  5.04966114e-16
 -5.21317026e-15 -2.17478837e-15  6.85645643e-16 -1.41265636e-16
 -2.28956670e-15  2.57517109e-15]
Standard deviation (should be 1): [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1.]
```

[36]:
```
## cumulative explained variance with all components
pca_full = PCA().fit(x_std)
cum_var = np.cumsum(pca_full.explained_variance_ratio_)

plt.figure(figsize = (8,6))
plt.plot(np.arange(1,len(cum_var)+1),cum_var,marker = 'o', linestyle = '--')
plt.xlabel("Number of components")
plt.ylabel("Cumulative Explained variance")
plt.title("Cumulative Explained variance by PCA components")
plt.grid(True)
plt.show()
```



[49]:
```
### Initializing PCA to reduce the data to 2 components

pca = PCA(n_components=2)
principalcomponents = pca.fit_transform(x_std)

## Creating dataframe for the two principal components
principaldf = pd.DataFrame(data = principalcomponents, columns=['PC1','PC2'])

## Concatenate the target variable for plotting

final_df = pd.concat([principaldf,df_bc[['target']]],axis = 1)

final_df
```

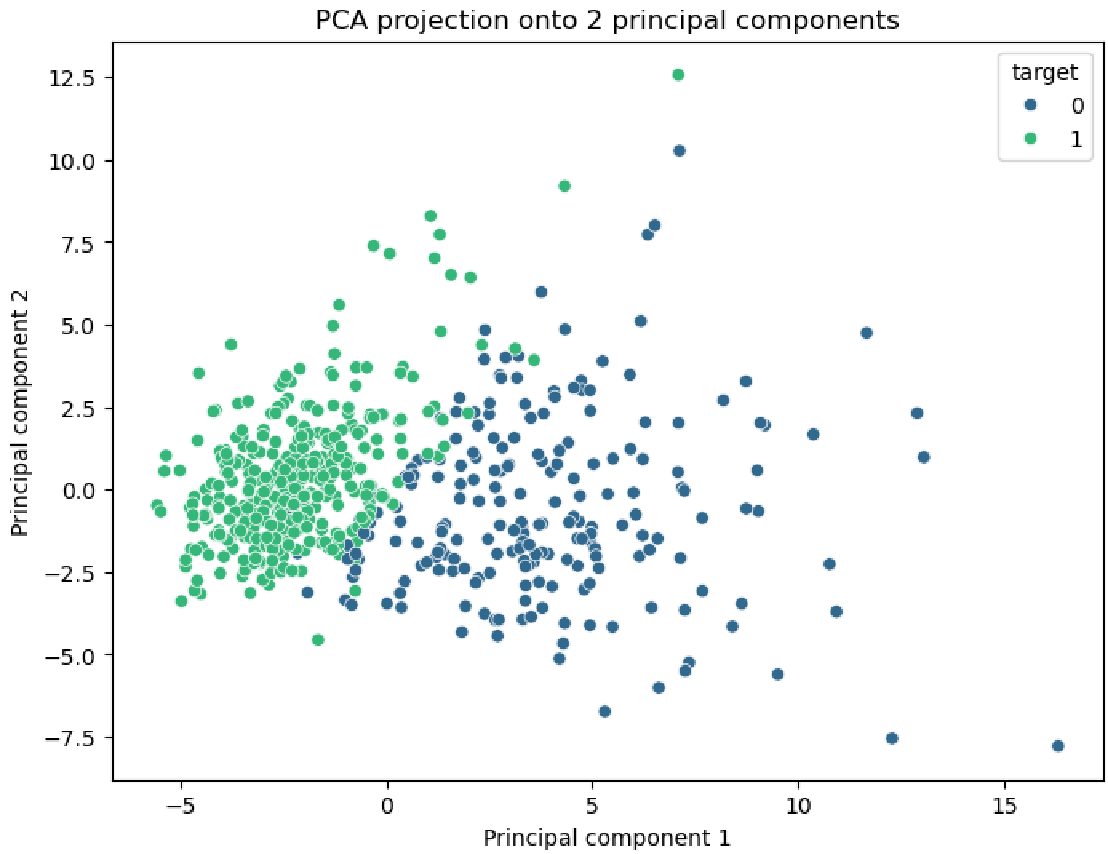
[49]:

	PC1	PC2	target
0	9.192837	1.948583	0
1	2.387802	-3.768172	0
2	5.733896	-1.075174	0
3	7.122953	10.275589	0

```
4     3.935302  -1.948072       0
..         …          …      …
564   6.439315  -3.576817       0
565   3.793382  -3.584048       0
566   1.256179  -1.902297       0
567  10.374794   1.672010       0
568  -5.475243  -0.670637       1

[569 rows x 3 columns]
```

[50]:

```python
plt.figure(figsize=(8,6))
sns.scatterplot(x='PC1',y = 'PC2', hue = 'target', data = final_df, palette =␣
 ↪'viridis')
plt.title("PCA projection onto 2 principal components")
plt.xlabel("Principal component 1")
plt.ylabel("Principal component 2")
plt.legend(title='target')
plt.show()
```



[51]:
```
print("Explained variance ratio for 2 components:")
print(pca.explained_variance_ratio_)
```

```
Explained variance ratio for 2 components:
[0.44272026 0.18971182]
```

[52]:
```
sum(pca.explained_variance_ratio_)
```

[52]:
```
0.6324320765155944
```

3.6 so 2 components are not enough to explain the other features

[56]:
```
### Initializing PCA to reduce the data to 10 components(as elbow begins after␣
↪10th feature)

pca = PCA(n_components=10)
principalcomponents = pca.fit_transform(x_std)

## Creating dataframe for the two principal components
principaldf = pd.DataFrame(data = principalcomponents,␣
↪columns=['PC1','PC2','PC3','PC4','PC5','PC6','PC7','PC8','PC9','PC10'])

## Concatenate the target variable for plotting

final_df = pd.concat([principaldf,df_bc[['target']]],axis = 1)

print(final_df)
print("Explained variance ratio for 10 components:")
print(pca.explained_variance_ratio_)
```

```
          PC1        PC2       PC3       PC4       PC5       PC6       PC7  \
0    9.192837   1.948583 -1.123166  3.633731 -1.195110  1.411424  2.159372
1    2.387802  -3.768172 -0.529293  1.118264  0.621775  0.028656  0.013357
2    5.733896  -1.075174 -0.551748  0.912083 -0.177086  0.541452 -0.668166
3    7.122953  10.275589 -3.232790  0.152547 -2.960878  3.053422  1.429911
4    3.935302  -1.948072  1.389767  2.940639  0.546747 -1.226495 -0.936212
..        …          …         …         …         …         …         …
564  6.439315  -3.576817  2.459487  1.177314 -0.074824 -2.375193 -0.596131
565  3.793382  -3.584048  2.088476 -2.506028 -0.510723 -0.246710 -0.716328
566  1.256179  -1.902297  0.562731 -2.089227  1.809991 -0.534447 -0.192759
567 10.374794   1.672010 -1.877029 -2.356031 -0.033742  0.567936  0.223084
568 -5.475243  -0.670637  1.490443 -2.299157 -0.184703  1.617837  1.698954

          PC8       PC9      PC10  target
0   -0.398400 -0.157113 -0.877447       0
1    0.240985 -0.711909  1.107010       0
2    0.097375  0.024069  0.454282       0
3    1.059566 -1.405435 -1.116957       0
```

```
4    0.636377 -0.263805  0.377699       0
..        …        …        …     …
564 -0.035476  0.987927  0.257027       0
565 -1.113363 -0.105209 -0.108608       0
566  0.341885  0.393915  0.520891       0
567 -0.280233 -0.542029 -0.089325       0
568  1.046362  0.374108 -0.047771       1

[569 rows x 11 columns]
Explained variance ratio for 10 components:
[0.44272026 0.18971182 0.09393163 0.06602135 0.05495768 0.04024522
 0.02250734 0.01588724 0.01389649 0.01168978]
```

[57]:
```
sum(pca.explained_variance_ratio_)
```

[57]:
```
0.9515688143209542
```