PrincipalComponentAnalysis

May 10, 2020

Principle Component Analysis

Implementing PCA on a 2-D Dataset:

Step 1: Normalize the data

First step is to normalize the data that we have so that PCA works properly. This is done by subtracting the respective means from the numbers in the respective column. So if we have two dimensions X and Y, all X become - and all Y become -. This produces a dataset whose mean is zero.

Step 2: Calculate the covariance matrix

Since the dataset we took is 2-dimensional, this will result in a 2x2 Covariance matrix.

Please note that Var[X1] = Cov[X1,X1] and Var[X2] = Cov[X2,X2].

Step 3: Calculate the eigenvalues and eigenvectors

Next step is to calculate the eigenvalues and eigenvectors for the covariance matrix. The same is possible because it is a square matrix. is an eigenvalue for a matrix A if it is a solution of the characteristic equation:

det(I - A) = 0

Where, I is the identity matrix of the same dimension as A which is a required condition for the matrix subtraction as well in this case and 'det' is the determinant of the matrix. For each eigenvalue , a corresponding eigen-vector v, can be found by solving:

(I - A)v = 0

Step 4: Choosing components and forming a feature vector:

We order the eigenvalues from largest to smallest so that it gives us the components in order or significance. Here comes the dimensionality reduction part. If we have a dataset with n variables, then we have the corresponding n eigenvalues and eigenvectors. It turns out that the eigenvector corresponding to the highest eigenvalue is the principal component of the dataset and it is our call as to how many eigenvalues we choose to proceed our analysis with. To reduce the dimensions, we choose the first p eigenvalues and ignore the rest. We do lose out some information in the process, but if the eigenvalues are small, we do not lose much.

Many thanks to :

https://www.kaggle.com/shrutimechlearn/step-by-step-pca-with-iris-dataset

https://sebastianraschka.com/Articles/2015_pca_in_3_steps.html

https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html

https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60

[1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn import datasets
```

```
import seaborn as sns
sns.set()
import matplotlib.pyplot as plt
import os
from sklearn.model_selection import train_test_split

from sklearn.datasets import load_iris
import pandas as pd

dataset=load_iris()
iris_data=pd.DataFrame(dataset['data'],columns=["Petal length","Petal␣
 ↪Width","Sepal Length","Sepal Width"])
iris_data['Species']=dataset['target']
iris_data['Species']=iris_data['Species'].apply(lambda x:␣
 ↪dataset['target_names'][x])
```

```
/home/sanjukta/anaconda/lib/python3.7/site-
packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is
deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

[2]:
```
iris_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Petal length  150 non-null    float64
 1   Petal Width   150 non-null    float64
 2   Sepal Length  150 non-null    float64
 3   Sepal Width   150 non-null    float64
 4   Species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

[3]:
```
iris_data.describe()
```

[3]:

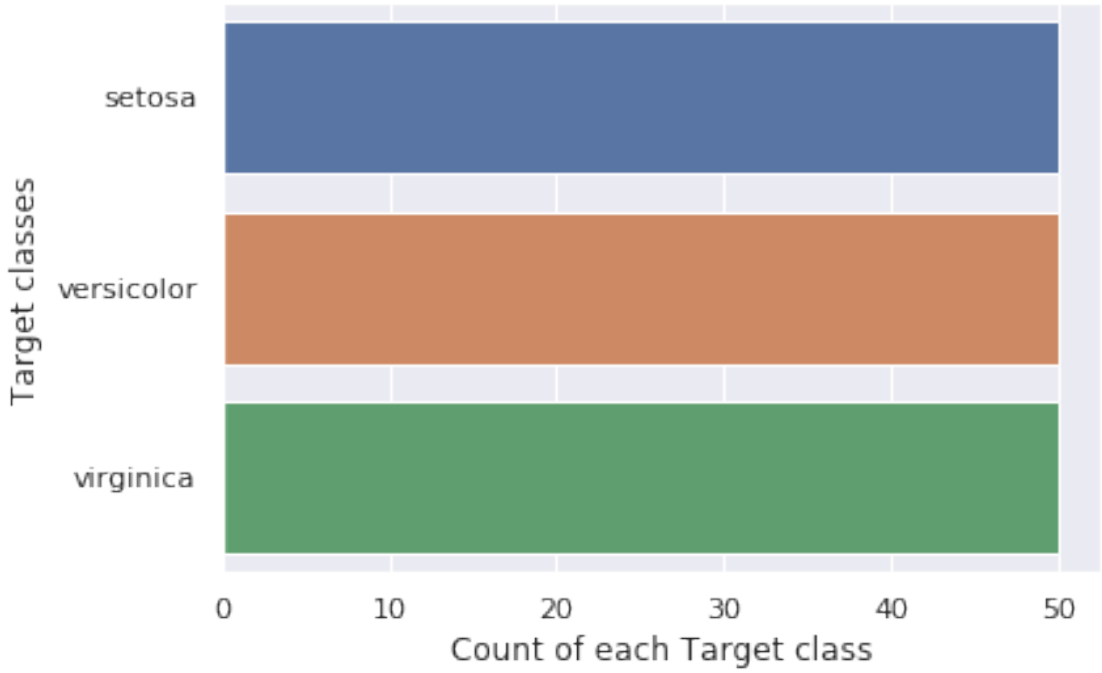
	Petal length	Petal Width	Sepal Length	Sepal Width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

[4]:
```
iris_data.head()
```

[4]:

	Petal length	Petal Width	Sepal Length	Sepal Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

[5]:
```
sns.countplot(y=iris_data.Species ,data=iris_data)
plt.xlabel("Count of each Target class")
plt.ylabel("Target classes")
plt.show()
```

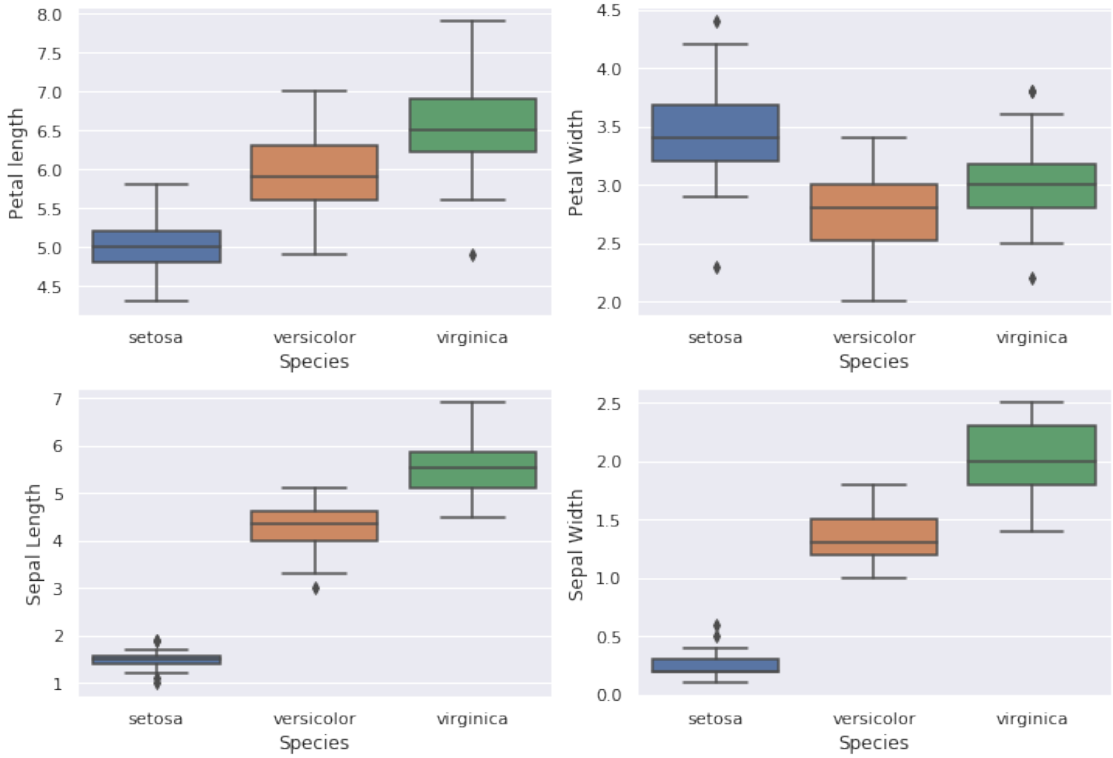


Early Insights :

```
There are total 150 rows
4 Independent variables to act as factors
All have same units of measurement (cm)
No missing data
Three unique target classes namely : 'setosa', 'versicolor' and 'virginica'
It is a balanced data set, all target classes have equal number of rows (50 each).
```

Bivariate Analysis
Box plot

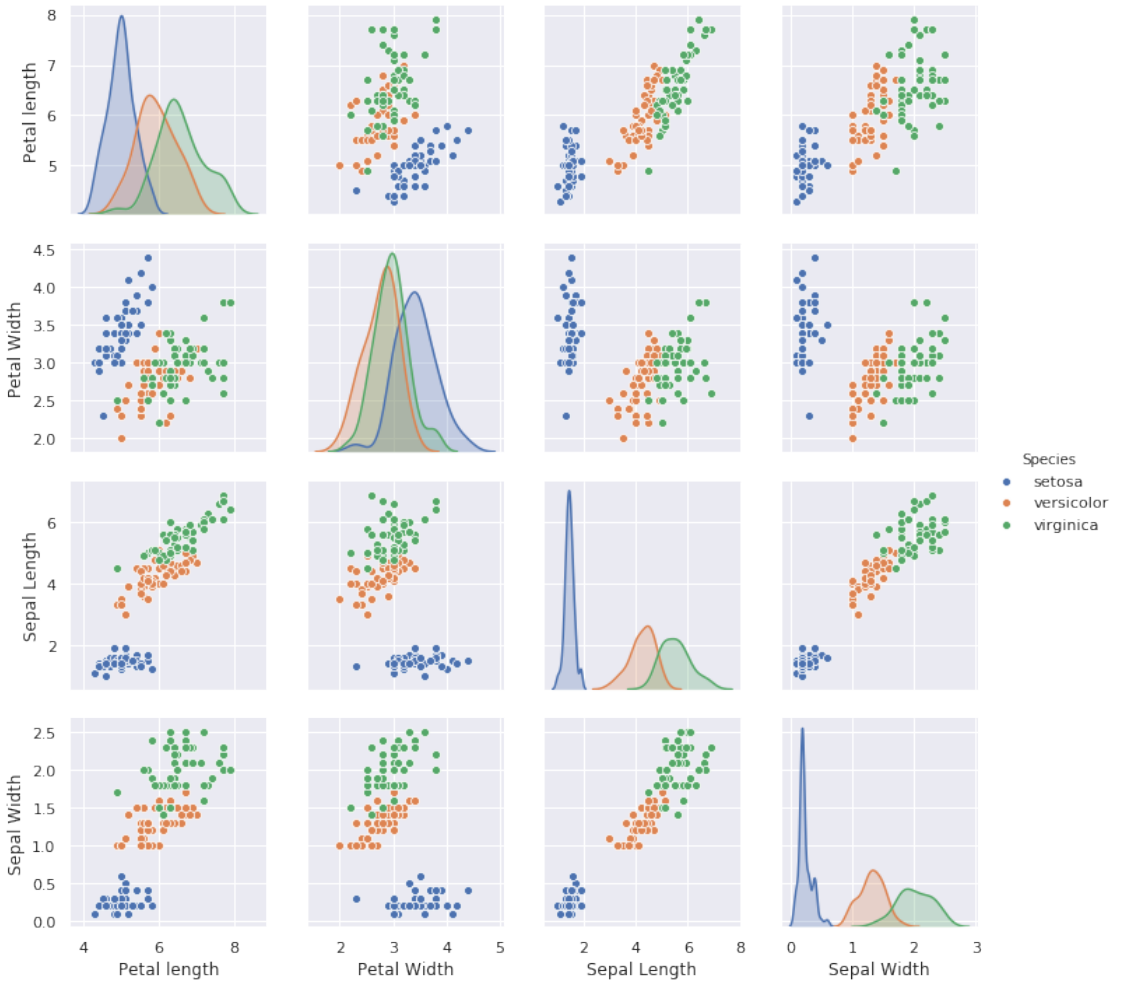
[6]:
```
fig,ax = plt.subplots(nrows = 2, ncols=2, figsize=(10,7))
row = 0
col = 0
for i in range(len(iris_data.columns) -1):
    if col > 1:
        row += 1
        col = 0
    axes = ax[row,col]
    sns.boxplot(x = iris_data['Species'], y = iris_data[iris_data.
 ↪columns[i]],ax = axes)
    col += 1
plt.tight_layout()
# plt.title("Individual Features by Class")
plt.show()
```



Pair Plot

[7]:
```
sns.pairplot(iris_data, hue = 'Species')
```

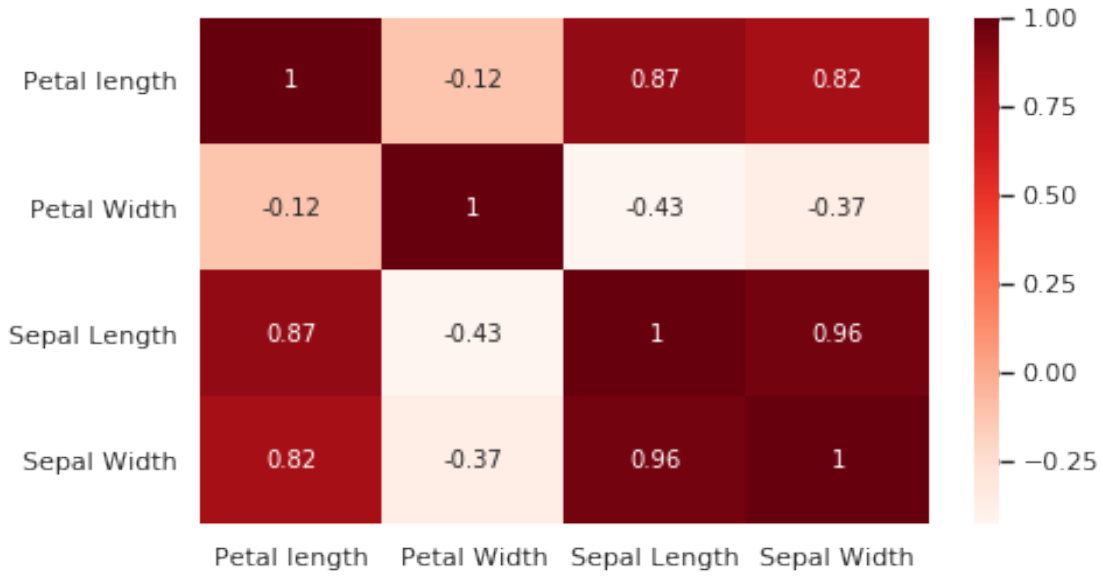
[7]:
```
<seaborn.axisgrid.PairGrid at 0x7f535efc6048>
```



Heat Map

[8]:
```
plt.figure(figsize=(7,4))
sns.heatmap(iris_data.corr(),annot=True,cmap='Reds')
```

[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f535ddfeac8>
```



Conclusion:

One of the biggest aims of these sort of plots and EDAs are to identify features that are not much helpful in explaining the target outcome. The Petal Width feature seems to be less relevant in explaining the target class as compared to the other features

If we consider Sepal Length Feature, it will easily seperate all 3 classes.

Modelling

1. Without PCA

Standardize the Data

PCA is effected by scale so you need to scale the features in your data before applying PCA. Us

[9]:
```
from sklearn.preprocessing import StandardScaler

features = ["Petal length","Petal Width","Sepal Length","Sepal Width"]#␣
 ↪Separating out the features
x = iris_data.loc[:, features].values# Separating out the target
print("Before Standardizing ", x)

y = iris_data.loc[:,['Species']].values# Standardizing the features
x_std = StandardScaler().fit_transform(x)

print("After Standardizing ",x_std)
```

```
Before Standardizing  [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
```

```
[4.6 3.1 1.5 0.2]
[5.  3.6 1.4 0.2]
[5.4 3.9 1.7 0.4]
[4.6 3.4 1.4 0.3]
[5.  3.4 1.5 0.2]
[4.4 2.9 1.4 0.2]
[4.9 3.1 1.5 0.1]
[5.4 3.7 1.5 0.2]
[4.8 3.4 1.6 0.2]
[4.8 3.  1.4 0.1]
[4.3 3.  1.1 0.1]
[5.8 4.  1.2 0.2]
[5.7 4.4 1.5 0.4]
[5.4 3.9 1.3 0.4]
[5.1 3.5 1.4 0.3]
[5.7 3.8 1.7 0.3]
[5.1 3.8 1.5 0.3]
[5.4 3.4 1.7 0.2]
[5.1 3.7 1.5 0.4]
[4.6 3.6 1.  0.2]
[5.1 3.3 1.7 0.5]
[4.8 3.4 1.9 0.2]
[5.  3.  1.6 0.2]
[5.  3.4 1.6 0.4]
[5.2 3.5 1.5 0.2]
[5.2 3.4 1.4 0.2]
[4.7 3.2 1.6 0.2]
[4.8 3.1 1.6 0.2]
[5.4 3.4 1.5 0.4]
[5.2 4.1 1.5 0.1]
[5.5 4.2 1.4 0.2]
[4.9 3.1 1.5 0.2]
[5.  3.2 1.2 0.2]
[5.5 3.5 1.3 0.2]
[4.9 3.6 1.4 0.1]
[4.4 3.  1.3 0.2]
[5.1 3.4 1.5 0.2]
[5.  3.5 1.3 0.3]
[4.5 2.3 1.3 0.3]
[4.4 3.2 1.3 0.2]
[5.  3.5 1.6 0.6]
[5.1 3.8 1.9 0.4]
[4.8 3.  1.4 0.3]
[5.1 3.8 1.6 0.2]
[4.6 3.2 1.4 0.2]
[5.3 3.7 1.5 0.2]
[5.  3.3 1.4 0.2]
[7.  3.2 4.7 1.4]
```

```
[6.4 3.2 4.5 1.5]
[6.9 3.1 4.9 1.5]
[5.5 2.3 4.  1.3]
[6.5 2.8 4.6 1.5]
[5.7 2.8 4.5 1.3]
[6.3 3.3 4.7 1.6]
[4.9 2.4 3.3 1. ]
[6.6 2.9 4.6 1.3]
[5.2 2.7 3.9 1.4]
[5.  2.  3.5 1. ]
[5.9 3.  4.2 1.5]
[6.  2.2 4.  1. ]
[6.1 2.9 4.7 1.4]
[5.6 2.9 3.6 1.3]
[6.7 3.1 4.4 1.4]
[5.6 3.  4.5 1.5]
[5.8 2.7 4.1 1. ]
[6.2 2.2 4.5 1.5]
[5.6 2.5 3.9 1.1]
[5.9 3.2 4.8 1.8]
[6.1 2.8 4.  1.3]
[6.3 2.5 4.9 1.5]
[6.1 2.8 4.7 1.2]
[6.4 2.9 4.3 1.3]
[6.6 3.  4.4 1.4]
[6.8 2.8 4.8 1.4]
[6.7 3.  5.  1.7]
[6.  2.9 4.5 1.5]
[5.7 2.6 3.5 1. ]
[5.5 2.4 3.8 1.1]
[5.5 2.4 3.7 1. ]
[5.8 2.7 3.9 1.2]
[6.  2.7 5.1 1.6]
[5.4 3.  4.5 1.5]
[6.  3.4 4.5 1.6]
[6.7 3.1 4.7 1.5]
[6.3 2.3 4.4 1.3]
[5.6 3.  4.1 1.3]
[5.5 2.5 4.  1.3]
[5.5 2.6 4.4 1.2]
[6.1 3.  4.6 1.4]
[5.8 2.6 4.  1.2]
[5.  2.3 3.3 1. ]
[5.6 2.7 4.2 1.3]
[5.7 3.  4.2 1.2]
[5.7 2.9 4.2 1.3]
[6.2 2.9 4.3 1.3]
[5.1 2.5 3.  1.1]
```

```
[5.7 2.8 4.1 1.3]
[6.3 3.3 6.  2.5]
[5.8 2.7 5.1 1.9]
[7.1 3.  5.9 2.1]
[6.3 2.9 5.6 1.8]
[6.5 3.  5.8 2.2]
[7.6 3.  6.6 2.1]
[4.9 2.5 4.5 1.7]
[7.3 2.9 6.3 1.8]
[6.7 2.5 5.8 1.8]
[7.2 3.6 6.1 2.5]
[6.5 3.2 5.1 2. ]
[6.4 2.7 5.3 1.9]
[6.8 3.  5.5 2.1]
[5.7 2.5 5.  2. ]
[5.8 2.8 5.1 2.4]
[6.4 3.2 5.3 2.3]
[6.5 3.  5.5 1.8]
[7.7 3.8 6.7 2.2]
[7.7 2.6 6.9 2.3]
[6.  2.2 5.  1.5]
[6.9 3.2 5.7 2.3]
[5.6 2.8 4.9 2. ]
[7.7 2.8 6.7 2. ]
[6.3 2.7 4.9 1.8]
[6.7 3.3 5.7 2.1]
[7.2 3.2 6.  1.8]
[6.2 2.8 4.8 1.8]
[6.1 3.  4.9 1.8]
[6.4 2.8 5.6 2.1]
[7.2 3.  5.8 1.6]
[7.4 2.8 6.1 1.9]
[7.9 3.8 6.4 2. ]
[6.4 2.8 5.6 2.2]
[6.3 2.8 5.1 1.5]
[6.1 2.6 5.6 1.4]
[7.7 3.  6.1 2.3]
[6.3 3.4 5.6 2.4]
[6.4 3.1 5.5 1.8]
[6.  3.  4.8 1.8]
[6.9 3.1 5.4 2.1]
[6.7 3.1 5.6 2.4]
[6.9 3.1 5.1 2.3]
[5.8 2.7 5.1 1.9]
[6.8 3.2 5.9 2.3]
[6.7 3.3 5.7 2.5]
[6.7 3.  5.2 2.3]
[6.3 2.5 5.  1.9]
```

```
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
After Standardizing  [[-9.00681170e-01  1.01900435e+00 -1.34022653e+00
-1.31544430e+00]
 [-1.14301691e+00 -1.31979479e-01 -1.34022653e+00 -1.31544430e+00]
 [-1.38535265e+00  3.28414053e-01 -1.39706395e+00 -1.31544430e+00]
 [-1.50652052e+00  9.82172869e-02 -1.28338910e+00 -1.31544430e+00]
 [-1.02184904e+00  1.24920112e+00 -1.34022653e+00 -1.31544430e+00]
 [-5.37177559e-01  1.93979142e+00 -1.16971425e+00 -1.05217993e+00]
 [-1.50652052e+00  7.88807586e-01 -1.34022653e+00 -1.18381211e+00]
 [-1.02184904e+00  7.88807586e-01 -1.28338910e+00 -1.31544430e+00]
 [-1.74885626e+00 -3.62176246e-01 -1.34022653e+00 -1.31544430e+00]
 [-1.14301691e+00  9.82172869e-02 -1.28338910e+00 -1.44707648e+00]
 [-5.37177559e-01  1.47939788e+00 -1.28338910e+00 -1.31544430e+00]
 [-1.26418478e+00  7.88807586e-01 -1.22655167e+00 -1.31544430e+00]
 [-1.26418478e+00 -1.31979479e-01 -1.34022653e+00 -1.44707648e+00]
 [-1.87002413e+00 -1.31979479e-01 -1.51073881e+00 -1.44707648e+00]
 [-5.25060772e-02  2.16998818e+00 -1.45390138e+00 -1.31544430e+00]
 [-1.73673948e-01  3.09077525e+00 -1.28338910e+00 -1.05217993e+00]
 [-5.37177559e-01  1.93979142e+00 -1.39706395e+00 -1.05217993e+00]
 [-9.00681170e-01  1.01900435e+00 -1.34022653e+00 -1.18381211e+00]
 [-1.73673948e-01  1.70959465e+00 -1.16971425e+00 -1.18381211e+00]
 [-9.00681170e-01  1.70959465e+00 -1.28338910e+00 -1.18381211e+00]
 [-5.37177559e-01  7.88807586e-01 -1.16971425e+00 -1.31544430e+00]
 [-9.00681170e-01  1.47939788e+00 -1.28338910e+00 -1.05217993e+00]
 [-1.50652052e+00  1.24920112e+00 -1.56757623e+00 -1.31544430e+00]
 [-9.00681170e-01  5.58610819e-01 -1.16971425e+00 -9.20547742e-01]
 [-1.26418478e+00  7.88807586e-01 -1.05603939e+00 -1.31544430e+00]
 [-1.02184904e+00 -1.31979479e-01 -1.22655167e+00 -1.31544430e+00]
 [-1.02184904e+00  7.88807586e-01 -1.22655167e+00 -1.05217993e+00]
 [-7.79513300e-01  1.01900435e+00 -1.28338910e+00 -1.31544430e+00]
 [-7.79513300e-01  7.88807586e-01 -1.34022653e+00 -1.31544430e+00]
 [-1.38535265e+00  3.28414053e-01 -1.22655167e+00 -1.31544430e+00]
 [-1.26418478e+00  9.82172869e-02 -1.22655167e+00 -1.31544430e+00]
 [-5.37177559e-01  7.88807586e-01 -1.28338910e+00 -1.05217993e+00]
 [-7.79513300e-01  2.40018495e+00 -1.28338910e+00 -1.44707648e+00]
 [-4.16009689e-01  2.63038172e+00 -1.34022653e+00 -1.31544430e+00]
 [-1.14301691e+00  9.82172869e-02 -1.28338910e+00 -1.31544430e+00]
 [-1.02184904e+00  3.28414053e-01 -1.45390138e+00 -1.31544430e+00]
 [-4.16009689e-01  1.01900435e+00 -1.39706395e+00 -1.31544430e+00]
 [-1.14301691e+00  1.24920112e+00 -1.34022653e+00 -1.44707648e+00]
 [-1.74885626e+00 -1.31979479e-01 -1.39706395e+00 -1.31544430e+00]
 [-9.00681170e-01  7.88807586e-01 -1.28338910e+00 -1.31544430e+00]
 [-1.02184904e+00  1.01900435e+00 -1.39706395e+00 -1.18381211e+00]
 [-1.62768839e+00 -1.74335684e+00 -1.39706395e+00 -1.18381211e+00]
 [-1.74885626e+00  3.28414053e-01 -1.39706395e+00 -1.31544430e+00]
 [-1.02184904e+00  1.01900435e+00 -1.22655167e+00 -7.88915558e-01]
```

```
[-9.00681170e-01  1.70959465e+00 -1.05603939e+00 -1.05217993e+00]
[-1.26418478e+00 -1.31979479e-01 -1.34022653e+00 -1.18381211e+00]
[-9.00681170e-01  1.70959465e+00 -1.22655167e+00 -1.31544430e+00]
[-1.50652052e+00  3.28414053e-01 -1.34022653e+00 -1.31544430e+00]
[-6.58345429e-01  1.47939788e+00 -1.28338910e+00 -1.31544430e+00]
[-1.02184904e+00  5.58610819e-01 -1.34022653e+00 -1.31544430e+00]
[ 1.40150837e+00  3.28414053e-01  5.35408562e-01  2.64141916e-01]
[ 6.74501145e-01  3.28414053e-01  4.21733708e-01  3.95774101e-01]
[ 1.28034050e+00  9.82172869e-02  6.49083415e-01  3.95774101e-01]
[-4.16009689e-01 -1.74335684e+00  1.37546573e-01  1.32509732e-01]
[ 7.95669016e-01 -5.92373012e-01  4.78571135e-01  3.95774101e-01]
[-1.73673948e-01 -5.92373012e-01  4.21733708e-01  1.32509732e-01]
[ 5.53333275e-01  5.58610819e-01  5.35408562e-01  5.27406285e-01]
[-1.14301691e+00 -1.51316008e+00 -2.60315415e-01 -2.62386821e-01]
[ 9.16836886e-01 -3.62176246e-01  4.78571135e-01  1.32509732e-01]
[-7.79513300e-01 -8.22569778e-01  8.07091462e-02  2.64141916e-01]
[-1.02184904e+00 -2.43394714e+00 -1.46640561e-01 -2.62386821e-01]
[ 6.86617933e-02 -1.31979479e-01  2.51221427e-01  3.95774101e-01]
[ 1.89829664e-01 -1.97355361e+00  1.37546573e-01 -2.62386821e-01]
[ 3.10997534e-01 -3.62176246e-01  5.35408562e-01  2.64141916e-01]
[-2.94841818e-01 -3.62176246e-01 -8.98031345e-02  1.32509732e-01]
[ 1.03800476e+00  9.82172869e-02  3.64896281e-01  2.64141916e-01]
[-2.94841818e-01 -1.31979479e-01  4.21733708e-01  3.95774101e-01]
[-5.25060772e-02 -8.22569778e-01  1.94384000e-01 -2.62386821e-01]
[ 4.32165405e-01 -1.97355361e+00  4.21733708e-01  3.95774101e-01]
[-2.94841818e-01 -1.28296331e+00  8.07091462e-02 -1.30754636e-01]
[ 6.86617933e-02  3.28414053e-01  5.92245988e-01  7.90670654e-01]
[ 3.10997534e-01 -5.92373012e-01  1.37546573e-01  1.32509732e-01]
[ 5.53333275e-01 -1.28296331e+00  6.49083415e-01  3.95774101e-01]
[ 3.10997534e-01 -5.92373012e-01  5.35408562e-01  8.77547895e-04]
[ 6.74501145e-01 -3.62176246e-01  3.08058854e-01  1.32509732e-01]
[ 9.16836886e-01 -1.31979479e-01  3.64896281e-01  2.64141916e-01]
[ 1.15917263e+00 -5.92373012e-01  5.92245988e-01  2.64141916e-01]
[ 1.03800476e+00 -1.31979479e-01  7.05920842e-01  6.59038469e-01]
[ 1.89829664e-01 -3.62176246e-01  4.21733708e-01  3.95774101e-01]
[-1.73673948e-01 -1.05276654e+00 -1.46640561e-01 -2.62386821e-01]
[-4.16009689e-01 -1.51316008e+00  2.38717193e-02 -1.30754636e-01]
[-4.16009689e-01 -1.51316008e+00 -3.29657076e-02 -2.62386821e-01]
[-5.25060772e-02 -8.22569778e-01  8.07091462e-02  8.77547895e-04]
[ 1.89829664e-01 -8.22569778e-01  7.62758269e-01  5.27406285e-01]
[-5.37177559e-01 -1.31979479e-01  4.21733708e-01  3.95774101e-01]
[ 1.89829664e-01  7.88807586e-01  4.21733708e-01  5.27406285e-01]
[ 1.03800476e+00  9.82172869e-02  5.35408562e-01  3.95774101e-01]
[ 5.53333275e-01 -1.74335684e+00  3.64896281e-01  1.32509732e-01]
[-2.94841818e-01 -1.31979479e-01  1.94384000e-01  1.32509732e-01]
[-4.16009689e-01 -1.28296331e+00  1.37546573e-01  1.32509732e-01]
[-4.16009689e-01 -1.05276654e+00  3.64896281e-01  8.77547895e-04]
[ 3.10997534e-01 -1.31979479e-01  4.78571135e-01  2.64141916e-01]
```

```
[-5.25060772e-02 -1.05276654e+00  1.37546573e-01  8.77547895e-04]
[-1.02184904e+00 -1.74335684e+00 -2.60315415e-01 -2.62386821e-01]
[-2.94841818e-01 -8.22569778e-01  2.51221427e-01  1.32509732e-01]
[-1.73673948e-01 -1.31979479e-01  2.51221427e-01  8.77547895e-04]
[-1.73673948e-01 -3.62176246e-01  2.51221427e-01  1.32509732e-01]
[ 4.32165405e-01 -3.62176246e-01  3.08058854e-01  1.32509732e-01]
[-9.00681170e-01 -1.28296331e+00 -4.30827696e-01 -1.30754636e-01]
[-1.73673948e-01 -5.92373012e-01  1.94384000e-01  1.32509732e-01]
[ 5.53333275e-01  5.58610819e-01  1.27429511e+00  1.71209594e+00]
[-5.25060772e-02 -8.22569778e-01  7.62758269e-01  9.22302838e-01]
[ 1.52267624e+00 -1.31979479e-01  1.21745768e+00  1.18556721e+00]
[ 5.53333275e-01 -3.62176246e-01  1.04694540e+00  7.90670654e-01]
[ 7.95669016e-01 -1.31979479e-01  1.16062026e+00  1.31719939e+00]
[ 2.12851559e+00 -1.31979479e-01  1.61531967e+00  1.18556721e+00]
[-1.14301691e+00 -1.28296331e+00  4.21733708e-01  6.59038469e-01]
[ 1.76501198e+00 -3.62176246e-01  1.44480739e+00  7.90670654e-01]
[ 1.03800476e+00 -1.28296331e+00  1.16062026e+00  7.90670654e-01]
[ 1.64384411e+00  1.24920112e+00  1.33113254e+00  1.71209594e+00]
[ 7.95669016e-01  3.28414053e-01  7.62758269e-01  1.05393502e+00]
[ 6.74501145e-01 -8.22569778e-01  8.76433123e-01  9.22302838e-01]
[ 1.15917263e+00 -1.31979479e-01  9.90107977e-01  1.18556721e+00]
[-1.73673948e-01 -1.28296331e+00  7.05920842e-01  1.05393502e+00]
[-5.25060772e-02 -5.92373012e-01  7.62758269e-01  1.58046376e+00]
[ 6.74501145e-01  3.28414053e-01  8.76433123e-01  1.44883158e+00]
[ 7.95669016e-01 -1.31979479e-01  9.90107977e-01  7.90670654e-01]
[ 2.24968346e+00  1.70959465e+00  1.67215710e+00  1.31719939e+00]
[ 2.24968346e+00 -1.05276654e+00  1.78583195e+00  1.44883158e+00]
[ 1.89829664e-01 -1.97355361e+00  7.05920842e-01  3.95774101e-01]
[ 1.28034050e+00  3.28414053e-01  1.10378283e+00  1.44883158e+00]
[-2.94841818e-01 -5.92373012e-01  6.49083415e-01  1.05393502e+00]
[ 2.24968346e+00 -5.92373012e-01  1.67215710e+00  1.05393502e+00]
[ 5.53333275e-01 -8.22569778e-01  6.49083415e-01  7.90670654e-01]
[ 1.03800476e+00  5.58610819e-01  1.10378283e+00  1.18556721e+00]
[ 1.64384411e+00  3.28414053e-01  1.27429511e+00  7.90670654e-01]
[ 4.32165405e-01 -5.92373012e-01  5.92245988e-01  7.90670654e-01]
[ 3.10997534e-01 -1.31979479e-01  6.49083415e-01  7.90670654e-01]
[ 6.74501145e-01 -5.92373012e-01  1.04694540e+00  1.18556721e+00]
[ 1.64384411e+00 -1.31979479e-01  1.16062026e+00  5.27406285e-01]
[ 1.88617985e+00 -5.92373012e-01  1.33113254e+00  9.22302838e-01]
[ 2.49201920e+00  1.70959465e+00  1.50164482e+00  1.05393502e+00]
[ 6.74501145e-01 -5.92373012e-01  1.04694540e+00  1.31719939e+00]
[ 5.53333275e-01 -5.92373012e-01  7.62758269e-01  3.95774101e-01]
[ 3.10997534e-01 -1.05276654e+00  1.04694540e+00  2.64141916e-01]
[ 2.24968346e+00 -1.31979479e-01  1.33113254e+00  1.44883158e+00]
[ 5.53333275e-01  7.88807586e-01  1.04694540e+00  1.58046376e+00]
[ 6.74501145e-01  9.82172869e-02  9.90107977e-01  7.90670654e-01]
[ 1.89829664e-01 -1.31979479e-01  5.92245988e-01  7.90670654e-01]
[ 1.28034050e+00  9.82172869e-02  9.33270550e-01  1.18556721e+00]
```

```
[ 1.03800476e+00  9.82172869e-02  1.04694540e+00  1.58046376e+00]
[ 1.28034050e+00  9.82172869e-02  7.62758269e-01  1.44883158e+00]
[-5.25060772e-02 -8.22569778e-01  7.62758269e-01  9.22302838e-01]
[ 1.15917263e+00  3.28414053e-01  1.21745768e+00  1.44883158e+00]
[ 1.03800476e+00  5.58610819e-01  1.10378283e+00  1.71209594e+00]
[ 1.03800476e+00 -1.31979479e-01  8.19595696e-01  1.44883158e+00]
[ 5.53333275e-01 -1.28296331e+00  7.05920842e-01  9.22302838e-01]
[ 7.95669016e-01 -1.31979479e-01  8.19595696e-01  1.05393502e+00]
[ 4.32165405e-01  7.88807586e-01  9.33270550e-01  1.44883158e+00]
[ 6.86617933e-02 -1.31979479e-01  7.62758269e-01  7.90670654e-01]]
```

PCA Projection to 2D

The original data has 4 columns (sepal length, sepal width, petal length, and petal width). In this section, the code projects the original data which is 4 dimensional into 2 dimensions.

[25]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])

principalDf.head()
```

[25]:

	principal component 1	principal component 2
0	-2.684126	0.319397
1	-2.714142	-0.177001
2	-2.888991	-0.144949
3	-2.745343	-0.318299
4	-2.728717	0.326755

Concatenating DataFrame along axis = 1. finalDf is the final DataFrame before plotting the data.

[26]:
```
finalDf = pd.concat([principalDf, iris_data[['Species']]], axis = 1)

finalDf.head()
```

[26]:

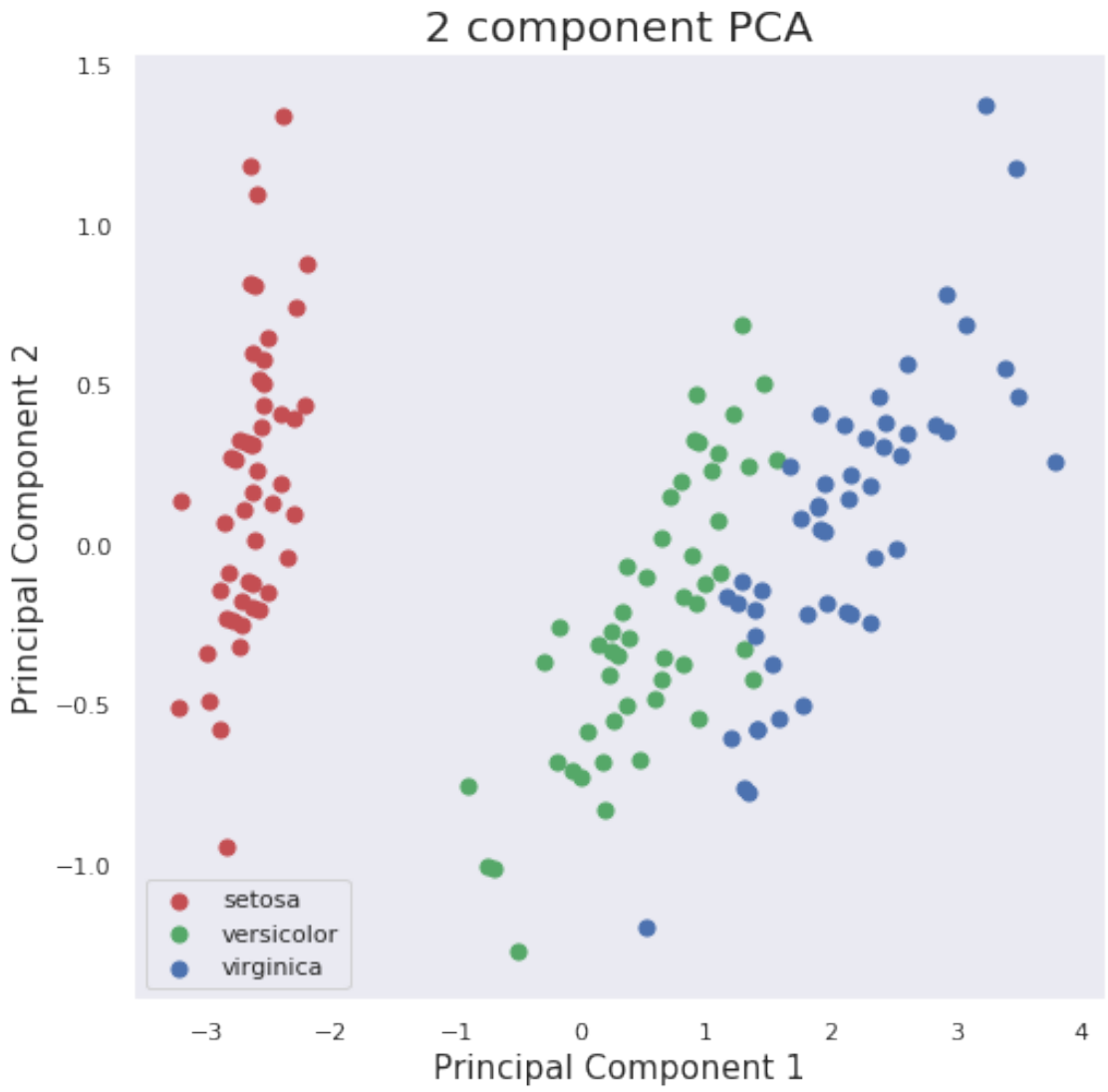
	principal component 1	principal component 2	Species
0	-2.684126	0.319397	setosa
1	-2.714142	-0.177001	setosa
2	-2.888991	-0.144949	setosa
3	-2.745343	-0.318299	setosa
4	-2.728717	0.326755	setosa

Visualize 2D Projection

This section is just plotting 2 dimensional data. Notice on the graph below that the classes seem well separated from each other.

[12]:
```
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
```

```
ax.set_title('2 component PCA', fontsize = 20)
targets = ['setosa', 'versicolor', 'virginica']
colors = ['r', 'g', 'b']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['Species'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
```



Explained Variance
Covariance Matrix

The classic approach to PCA is to perform the eigendecomposition on the covariance matrix , which is a dŒd

matrix where each element represents the covariance between two features. The covariance between two features is calculated as follows: jk=1n1i=1n(xijxŕj)(xikxŕk).

We can summarize the calculation of the covariance matrix via the following matrix equation: =1n1((Xxŕ)T(Xxŕ))

where xŕ is the mean vector xŕ=1ni=1nxi. The mean vector is a d-dimensional vector where each value in this vector represents the sample mean of a feature column in the dataset.

[13]:
```
# simply used for demonstration purposes, equivalently, we could have used the␣
 ↪numpy cov function:

print('NumPy covariance matrix: \n%s' %np.cov(x_std.T))
```

```
NumPy covariance matrix:
[[ 1.00671141 -0.11835884  0.87760447  0.82343066]
 [-0.11835884  1.00671141 -0.43131554 -0.36858315]
 [ 0.87760447 -0.43131554  1.00671141  0.96932762]
 [ 0.82343066 -0.36858315  0.96932762  1.00671141]]
```

[14]:
```
# Next, we perform an eigendecomposition on the covariance matrix:

cov_mat = np.cov(x_std.T)

eig_vals, eig_vecs = np.linalg.eig(cov_mat)

print('Eigenvectors \n%s' %eig_vecs)
print('\nEigenvalues \n%s' %eig_vals)
```

```
Eigenvectors
[[ 0.52106591 -0.37741762 -0.71956635  0.26128628]
 [-0.26934744 -0.92329566  0.24438178 -0.12350962]
 [ 0.5804131  -0.02449161  0.14212637 -0.80144925]
 [ 0.56485654 -0.06694199  0.63427274  0.52359713]]

Eigenvalues
[2.93808505 0.9201649  0.14774182 0.02085386]
```

1 Correlation Matrix

Eigendecomposition of the standardized data based on the correlation matrix:

[15]:
```
cor_mat1 = np.corrcoef(x_std.T)

eig_vals, eig_vecs = np.linalg.eig(cor_mat1)

print('Eigenvectors \n%s' %eig_vecs)
print('\nEigenvalues \n%s' %eig_vals)
```

```
Eigenvectors
[[ 0.52106591 -0.37741762 -0.71956635  0.26128628]
 [-0.26934744 -0.92329566  0.24438178 -0.12350962]
 [ 0.5804131  -0.02449161  0.14212637 -0.80144925]
 [ 0.56485654 -0.06694199  0.63427274  0.52359713]]

Eigenvalues
[2.91849782 0.91403047 0.14675688 0.02071484]
```

Eigendecomposition of the raw data based on the correlation matrix:

[16]:
```
cor_mat2 = np.corrcoef(x.T)

eig_vals, eig_vecs = np.linalg.eig(cor_mat2)

print('Eigenvectors \n%s' %eig_vecs)
print('\nEigenvalues \n%s' %eig_vals)
```

```
Eigenvectors
[[ 0.52106591 -0.37741762 -0.71956635  0.26128628]
 [-0.26934744 -0.92329566  0.24438178 -0.12350962]
 [ 0.5804131  -0.02449161  0.14212637 -0.80144925]
 [ 0.56485654 -0.06694199  0.63427274  0.52359713]]

Eigenvalues
[2.91849782 0.91403047 0.14675688 0.02071484]
```

We can clearly see that all three approaches yield the same eigenvectors and eigenvalue pairs:

```
Eigendecomposition of the covariance matrix after standardizing the data.
Eigendecomposition of the correlation matrix.
Eigendecomposition of the correlation matrix after standardizing the data.
```

2 Singular Value Decomposition

While the eigendecomposition of the covariance or correlation matrix may be more intuitiuve, most PCA implementations perform a Singular Value Decomposition (SVD) to improve the computational efficiency. So, let us perform an SVD to confirm that the result are indeed the same:

[17]:
```
u,s,v = np.linalg.svd(x_std.T)
u
```

[17]:
```
array([[-0.52106591, -0.37741762,  0.71956635,  0.26128628],
       [ 0.26934744, -0.92329566, -0.24438178, -0.12350962],
       [-0.5804131 , -0.02449161, -0.14212637, -0.80144925],
       [-0.56485654, -0.06694199, -0.63427274,  0.52359713]])
```

3 Selecting Principal Components

In order to decide which eigenvector(s) can dropped without losing too much information for the construction of lower-dimensional subspace, we need to inspect the corresponding eigenvalues: The eigenvectors with the lowest eigenvalues bear the least information about the distribution of the data; those are the ones can be dropped. In order to do so, the common approach is to rank the eigenvalues from highest to lowest in order choose the top k eigenvectors.

[18]:
```
# Make a list of (eigenvalue, eigenvector) tuples
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]

# Sort the (eigenvalue, eigenvector) tuples from high to low
eig_pairs.sort(key=lambda x: x[0], reverse=True)

# Visually confirm that the list is correctly sorted by decreasing eigenvalues
print('Eigenvalues in descending order:')
for i in eig_pairs:
    print(i[0])
```

```
Eigenvalues in descending order:
2.918497816531998
0.9140304714680696
0.1467568755713156
0.02071483642861873
```

4 Explained Variance

After sorting the eigenpairs, the next question is "how many principal components are we going to choose for our new feature subspace?" A useful measure is the so-called "explained variance," which can be calculated from the eigenvalues. The explained variance tells us how much information (variance) can be attributed to each of the principal components.

[19]:
```
tot = sum(eig_vals)
var_exp = [(i / tot)*100 for i in sorted(eig_vals, reverse=True)]
cum_var_exp = np.cumsum(var_exp)

cum_var_exp
```

[19]:
```
array([ 72.96244541,  95.8132072 ,  99.48212909, 100.        ])
```

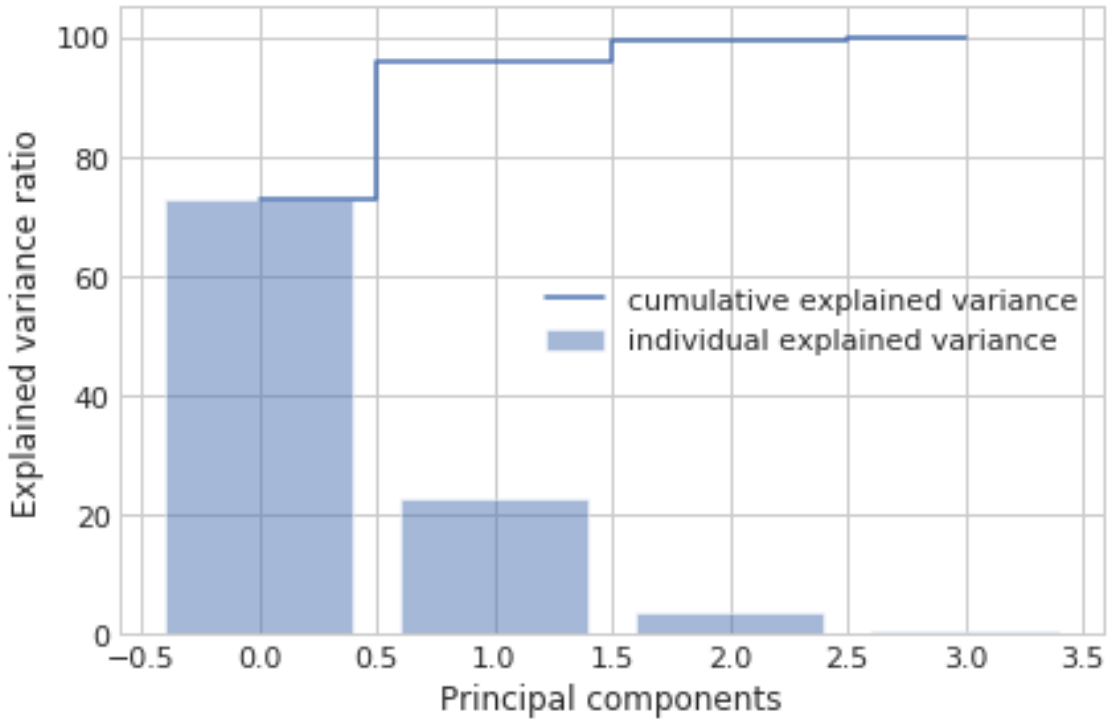
Observation :

you can see that the first principal component contains 72.77% of the variance and together, the two components contain 95.80% of the information and so on.

[20]:
```
with plt.style.context('seaborn-whitegrid'):
    plt.figure(figsize=(6, 4))

    plt.bar(range(4), var_exp, alpha=0.5, align='center',
            label='individual explained variance')
    plt.step(range(4), cum_var_exp, where='mid',
```

```
            label='cumulative explained variance')
    plt.ylabel('Explained variance ratio')
    plt.xlabel('Principal components')
    plt.legend(loc='best')
    plt.tight_layout()
```



The plot above clearly shows that most of the variance (72.77% of the variance to be precise) can be explained by the first principal component alone. The second principal component still bears some information (23.03%) while the third and fourth principal components can safely be dropped without losing to much information. Together, the first two principal components contain 95.8% of the information.

5 Projection Matrix

The construction of the projection matrix that will be used to transform the Iris data onto the new feature subspace. Although, the name "projection matrix" has a nice ring to it, it is basically just a matrix of our concatenated top k eigenvectors.

Here, we are reducing the 4-dimensional feature space to a 2-dimensional feature subspace, by choosing the "top 2" eigenvectors with the highest eigenvalues to construct our dŒk -dimensional eigenvector matrix W.

[21]:
```
matrix_w = np.hstack((eig_pairs[0][1].reshape(4,1),
                      eig_pairs[1][1].reshape(4,1)))
```

```
print('Matrix W:\n', matrix_w)
```

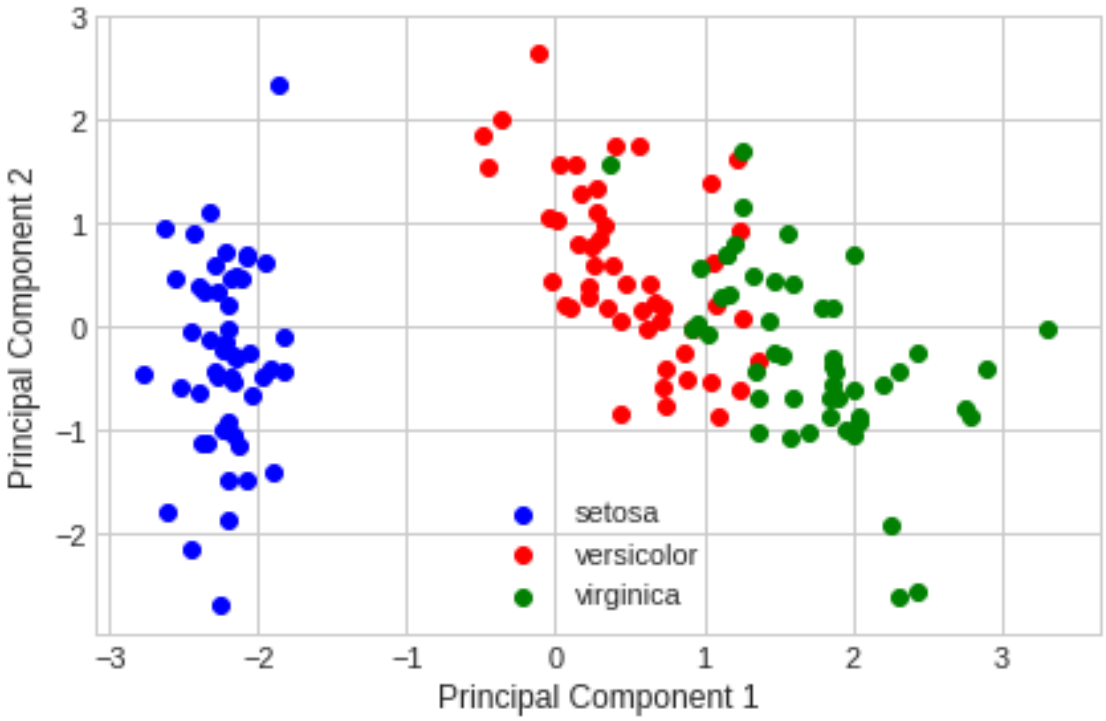
```
Matrix W:
 [[ 0.52106591 -0.37741762]
 [-0.26934744 -0.92329566]
 [ 0.5804131  -0.02449161]
 [ 0.56485654 -0.06694199]]
```

6 Projection Onto the New Feature Space

In this last step we will use the 4Œ2-dimensional projection matrix W to transform our samples onto the new subspace via the equation Y=XŒW, where Y is a 150Œ2 matrix of our transformed samples.

[22]:
```
Y = x_std.dot(matrix_w)
```

[23]:
```
with plt.style.context('seaborn-whitegrid'):
    plt.figure(figsize=(6, 4))
    for lab, col in zip(('setosa', 'versicolor', 'virginica'),
                        ('blue', 'red', 'green')):
        plt.scatter(Y[iris_data['Species']==lab, 0],
                    Y[iris_data['Species']==lab, 1],
                    label=lab,
                    c=col)
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.legend(loc='lower center')
    plt.tight_layout()
    plt.show()
```



7 Conclusion:

In the Explained Variance section we come to a conclusion that the first principal component contains 72.96% of the variance and the second principal component contains 22.85% of the variance. Together, the two components contain 95.81% of the information.

[ ]: