Lecture 17: Unsupervised Learning: Principal Component Analysis

Data Science I

Elizabeth Sweeney [ems4003@med.cornell.edu]

Lab #5

- Lab #5 will be due on Monday, 12/07 @ 6:00 pm on Canvas
- Lab #5 is graded for correctness and is worth 5% of you grade!
- When you submit the lab you will submit it as an html that contains a link to the GitHub repository with the code for your lab
- Make sure your repo is private. Make sure you add your TAs (Nick and Anjile) and I as collaborators to your repo. There are instructions for this in Lecture 2. You will lose points if you do not.
- There will be 5 participation points for the lab!

Data Challenge #6

Data Challenge #6

- The assignment is due on Monday, 12/16 @ 6:00 pm on Canvas
- A rubric for grading of the data challenge is provide on the assignment
- When you submit the data challenge you will submit it as an html that contains a link to the GitHub repository with the code for your data challenge
- Make sure your repo is private. Make sure you add your TAs (Nick and Anjile) and I as collaborators to your repo. There are instructions for this in Lecture 2. You will lose points if you do not.
- Make sure the html file shows your code.

Data Challenge #6

Data Challenge #6 Hint! Paste this into the first code cell to turn code on / off in Jupyter notebook.

```
# code to show/hide python code in a report
from IPython.display import HTML

HTML('''
<script src='//code.jquery.com/jquery-3.3.1.min.js'></script>
<script>
code_show=true;
function code_toggle() {
if (code_show){
$('div.input').hide();
$('div .jp-CodeCell .jp-Cell-inputWrapper').hide();
} else {
$('div.input').show();
$('div .jp-CodeCell .jp-Cell-inputWrapper').show();
}
code_show = !code_show
}
$( document ).ready(code_toggle);
</script>
<form action="javascript:code_toggle()">
<input type="submit" value="Code on/off"></form>''')
```

Just For Fun!

https://adventofcode.com

Required Reading

Chapter 10.2 from An Introduction to Statistical Learning (http://faculty.marshall.usc.edu/gareth-james/ISL/)

Chapter 5, In Depth: Principal Component Analysis from the Python Data Science Handbook

Recall: Unsupervised Learning

Unsupervised vs Supervised Learning:

- In Data Science II you will learn about supervised learning methods such as regression and classification.
- In that setting you will observe both a set of features $X_1, X_2, \cdots, X_p$ for each object, as well as a response or outcome variable Y . The goal is then to predict Y using $X_1, X_2, \ldots, X_p$.
- In Data Science I, we focus on **unsupervised learning**, where we observe only the features $X_1, X_2, \ldots, X_p$. We are not interested in prediction, because we do not have an associated response variable Y.

Goals and Challenges of Unsupervised Learning

Goals:

- The goal is to discover interesting things about the measurements: is there an informative way to visualize the data? Can we discover subgroups among the variables or among the observations?

Challenges:

- Unsupervised learning is more subjective than supervised learning, as there is no simple goal for the analysis, such as prediction of a response.

Principal Component Analysis

- PCA produces a low-dimensional representation of a dataset. It finds a sequence of linear combinations of the variables that have maximal variance, and are mutually uncorrelated.
- Apart from producing derived variables for use in supervised learning problems, PCA also serves as a tool for data visualization.

PCA vs Clustering

- PCA looks for a low-dimensional representation of the observations that explains a good fraction of the variance.
- Clustering looks for homogeneous subgroups among the observations.

Principal Components Analysis: Details

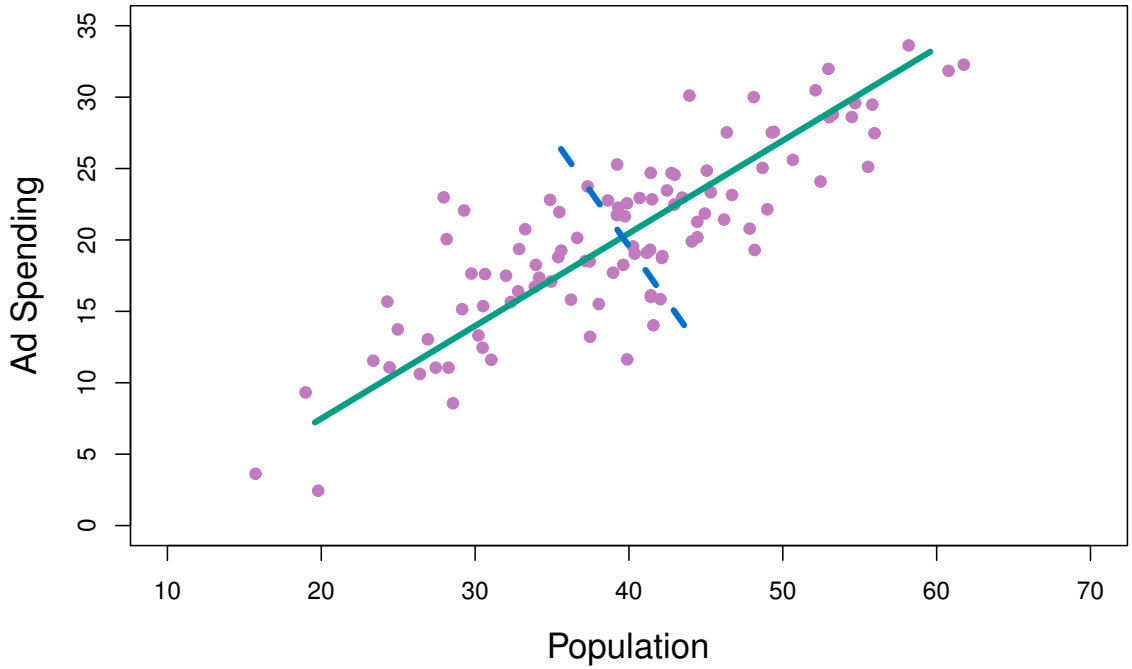
- The **first principal component** of a set of features $X_1, X_2, \ldots, X_p$ is the normalized linear combination of the features:

$$Z_1 = \phi_{11} X_1 + \phi_{21} X_2 + \ldots + \phi_{p1} X_p$$

that has the largest variance. By normalized, we mean that $\sum_{j=1}^{p} \phi_{j1}^2 = 1$.

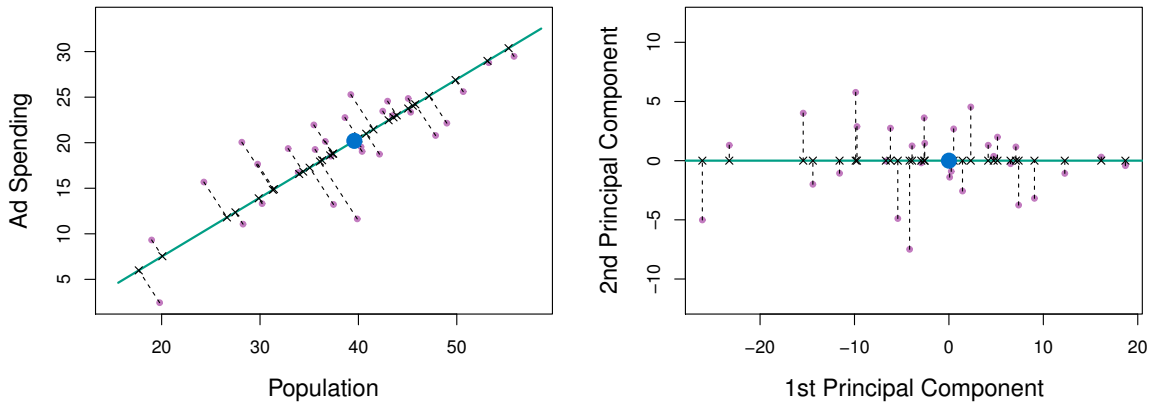
- We refer to the elements $\phi_{11}, \ldots, \phi_{p1}$ as the **loadings** of the first principal component; together, the loadings make up the **principal component loading vector**, $\phi_1 = (\phi_{11}, \phi_{21}, \ldots, \phi_{p1})^T$.
- We constrain the loadings so that their sum of squares is equal to one, since otherwise setting these elements to be arbitrarily large in absolute value could result in an arbitrarily large variance.

PCA: Advertisement Data Example



The population size (**pop**) and ad spending (**ad**) for 100 different cities are shown as purple circles. The green solid line indicates the first principal component, and the blue dashed line indicates the second principal component.

Pictures of PCA: Continued



A subset of the advertising data. **Left**: The first principal component, chosen to minimize the sum of the squared perpendicular distances to each point, is shown in green. These distances are represented using the black dashed line segments. **Right**: The left-hand panel has been rotated so that the first principal component lies on the x-axis.

Computation of Principal Components

- Suppose we have a $n \times p$ data set X. Since we are only interested in variance, we assume that each of the variables in X has been centered to have mean zero (that is, the column means of X are zero).
- We then look for the linear combination of the sample feature values of the form

$$z_{i1} = \phi_{11} x_{i1} + \phi_{21} x_{i2} + \ldots + \phi_{p1} x_{ip} \tag{1}$$

for $i = 1, \ldots, n$ that has largest sample variance, subject to the constraint that $\sum_{j=1}^{p} \phi_{j1}^2 = 1$.

- Since each of the x_{ij} has mean zero, then so does z_{i1} (for any values of ϕ_{j1}).
- Hence the sample variance of the z_{i1} can be written as $\frac{1}{n} \sum_{i=1}^{n} z_{i1}^2$

Computation: continued

- Plugging in (1) the first principal component loading vector solves the optimization problem

$$\underset{\phi_{11},\ldots,\phi_{p1}}{\text{maximize}} \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{p}\phi_{j1}x_{ij}\right)^2 \text{ subject to } \sum_{j=1}^{p}\phi_{j1}^2 = 1$$

- Why maximize? The goal is to get a lower dimensional representation of the data that capture the variance of the original data. So went want to maximize the variance of the component scores.
- This problem can be solved via a **singular-value decomposition** of the matrix X, a standard technique in linear algebra.
- We refer to Z_1 as the first principal component, with realized values $z_{11}, \ldots, z_{n1}$

Geometry of PCA

- The loading vector ϕ_1 with elements $\phi_{11}, \phi_{21}, \ldots, \phi_{p1}$ defines a **direction in feature space** along which the data vary the most.
- If we project the n data points $x_1, \ldots, x_n$ onto this direction, the projected values are the **principal component scores** $z_{11}, \ldots, z_{n1}$ themselves.

Second principal component

- The second principal component is the linear combination of $X_1, X_2, \ldots, X_p$ that has maximal variance among all linear combinations that are **uncorrelated** with Z_1.
- The second principal component scores $z_{12}, z_{22}, \ldots, z_{n2}$ take the form

$$z_{21} = \phi_{12} x_{21} + \phi_{22} x_{22} + \ldots + \phi_{p2} x_{2p}$$

- where ϕ_2 is the second principal component loading vector, with elements $\phi_{12}, \phi_{22}, \ldots, \phi_{p2}$

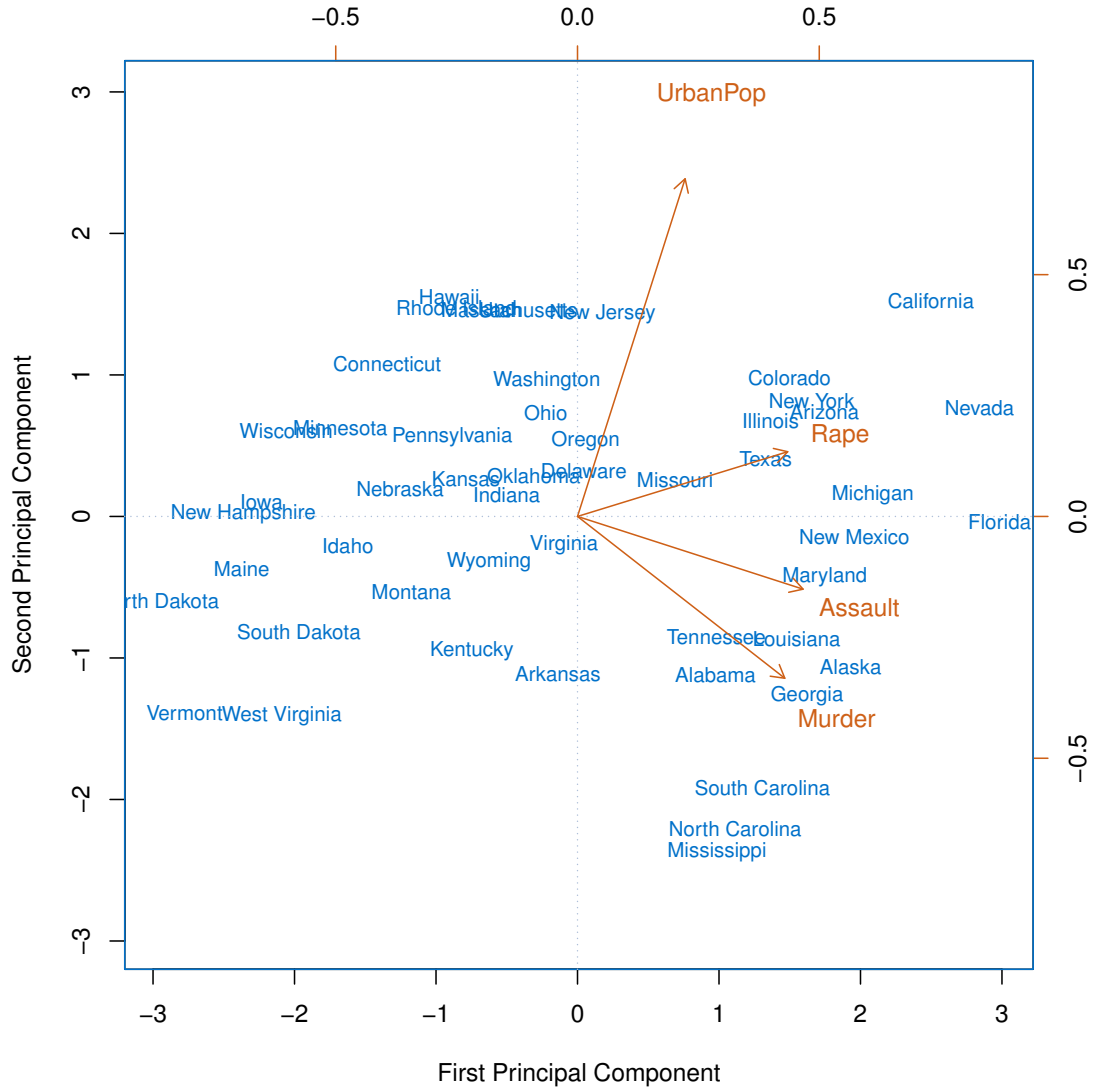
PCA continued

- It turns out that constraining Z_2 to be uncorrelated with Z_1 is equivalent to constraining the direction ϕ_2 to be **orthogonal** (perpendicular) to the direction ϕ_1. And so on.
- The principal component directions $\phi_1, \phi_2, \phi_3, ...$ are the ordered sequence of right singular vectors of the matrix X, and the variances of the components are $\frac{1}{n}$ times the squares of the singular values.
- There are at most $min(n-1, p)$ principal components. When $p > n-1$, the rest of the principal components are zeros. When $p < n-1$ there are p eigenvalues. The comparison is with $n-1$ and not n because one degree of freedom is lost due to the estimation of the mean.

Example - `USArrests`

- We illustrate the use of PCA on the `USArrests` data set in the `base` package of `R`.
- For each of the 50 states in the United States, the data set contains the number of arrests per 100,000 residents for each of three crimes: `Assault`, `Murder`, and `Rape`.
- We also record `UrbanPop` (the percent of the population in each state living in urban areas).
- The principal component score vectors have length $n = 50$, and the principal component loading vectors have length $p = 4$.
- PCA was performed after standardizing each variable to have mean zero and standard deviation one.

USArrests data: PCA plot (biplot)



Details of Figure 1

The first two principal components for the `USArrests` data.

- The blue state names represent the scores for the first two principal components.
- The orange arrows indicate the first two principal component loading vectors (with axes on the top and right).
  - For example, the loading for `Rape` on the first component is `0.54`, and its loading on the second principal component `0.17` [the word `Rape` is centered at the point `(0.54, 0.17)`.
- This figure is known as a **biplot**, because it displays both the principal component scores and the principal component loadings.

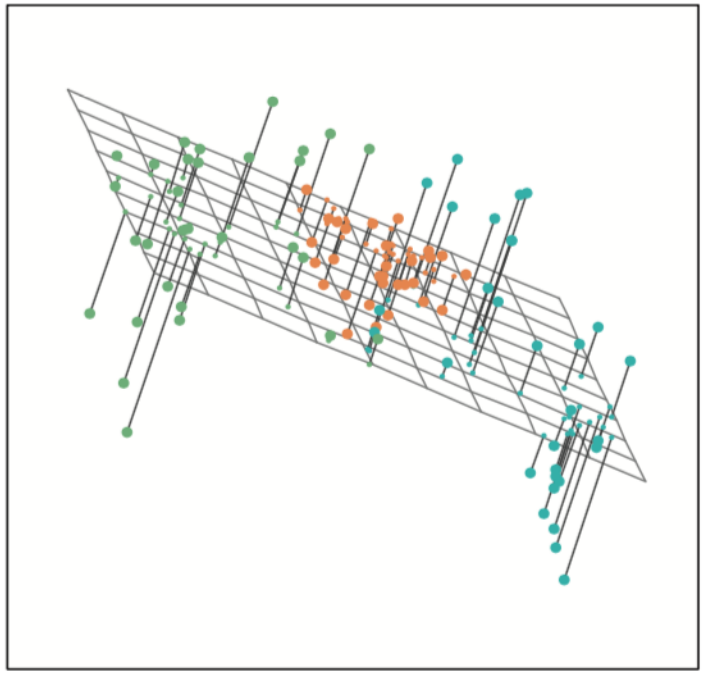
Interpretation of Figure 1

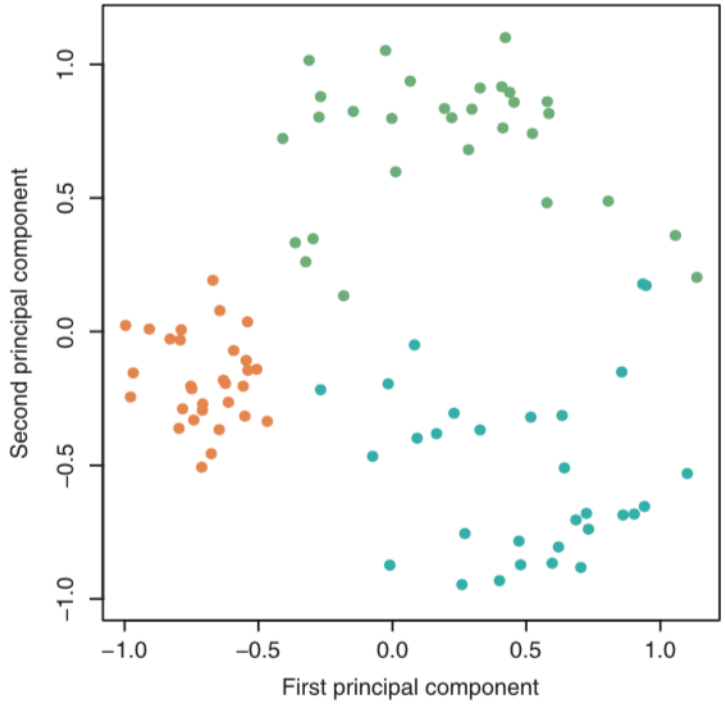
- The first loading vector places approximately equal weight on Assault, Murder, and Rape, with much less weight on UrbanPop. Hence this component roughly corresponds to a measure of overall rates of serious crimes.
- The second loading vector places most of its weight on UrbanPop and much less weight on the other three features. Hence, this component roughly corresponds to the level of urbanization of the state.
- Overall, we see that the crime-related variables (Murder, Assault, and Rape) are located close to each other, and that the UrbanPop variable is far from the other three. This indicates that the crime-related variables are correlated with each other—states with high murder rates tend to have high assault and rape rates—and that the UrbanPop variable is less correlated with the other three.

Interpretation of Figure 1

- States with large positive scores on the first component, such as California, Nevada and Florida, have high crime rates, while states like North Dakota, with negative scores on the first component, have low crime rates.
- California also has a high score on the second component, indicating a high level of urbanization, while the opposite is true for states like Mississippi.
- States close to zero on both components, such as Indiana, have approximately average levels of both crime and urbanization.

Another interpretation of Principal Components



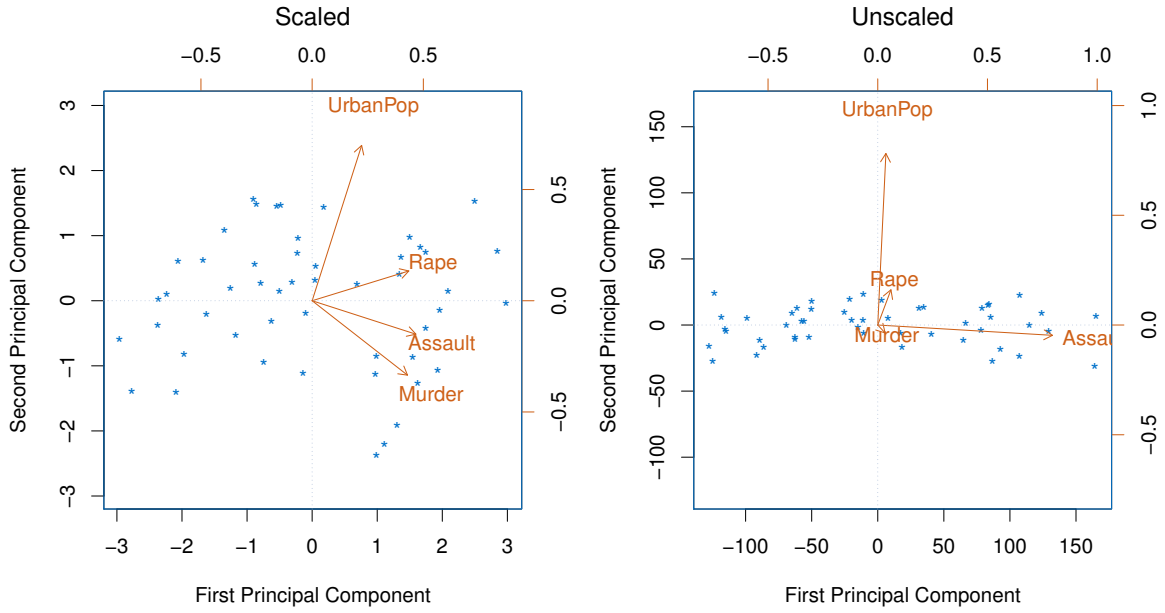


PCA finds the hyperplane closest to the observations

- The first principal component loading vector has a very special property: it defines the line in p-dimensional space that is closest to the n observations (using average squared Euclidean distance as a measure of closeness)
- The notion of principal components as the dimensions that are closest to the n observations extends beyond just the first principal component.
- For instance, the first two principal components of a data set span the plane that is closest to the n observations, in terms of average squared Euclidean distance.
- *Details of Figure 2*: **Left panel**: A two-dimensional hyperplane closest to the observations. The colors help us identify whats going on but do not correspond to any variable. **Right Panel**: A two-dimensional (lower dimensional) projection of the data.
- *Similarities with least square regression*: Least squares also fit a hyperplane to the data and the errors are evaluated by looking distance of the y's from the hyperplane. Since there are no y's in PCA, the **shortest distance** or the line

Scaling of the Variables Matter

- If the variables are in different units, scaling each to have standard deviation equal to one is recommended.
- If they are in the same units, you might or might not scale the variables.



Proportion Variance Explained

- To understand the strength of each component, we are interested in knowing the proportion of variance explained (PVE) by each one.
- The total variance present in a data set (assuming that the variables have been centered to have mean zero) is defined as

$$\sum_{j=1}^{p} Var(X_j) = \sum_{j=1}^{p} \frac{1}{n} \sum_{i=1}^{n} x_{ij}^2$$

and the variance explained by the m^{th} principal component is:

$$Var(Z_m) = \frac{1}{n} \sum_{i=1}^{n} z_{im}^2$$

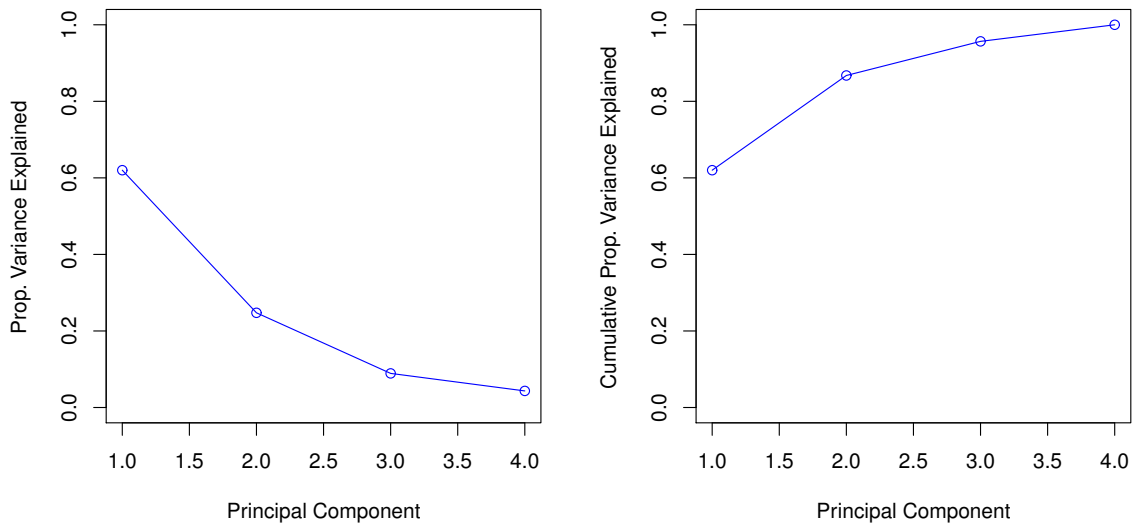
- It can be shown that $\sum_{j=1}^{p} Var(X_j) = \sum_{m=1}^{M} Var(Z_m)$ with $M = min(n-1, p)$.

Proportion Variance Explained: Continued

- Therefore, the PVE of the m^{th} principal component is given by the positive quantity between 0 and 1

$$\frac{\sum_{i=1}^{n} z_{im}^2}{\sum_{j=1}^{p} \sum_{i=1}^{n} x_{ij}^2}$$

- The PVEs sum to one. We sometimes display the cumulative PVEs.



How Many Principal Components Should We Use?

If we use principal components as a summary of our data, how many components are sufficient?

- the **scree plot** on the previous slide can be used as a guide: we look for an **elbow**.

The Iris Dataset

- The Iris Dataset was introduced by R.A. Fisher.
- The dataset consists of 50 samples from 3 species of Iris flowers, for each of the flowers the length and width of the petals and sepals is measured.

The Iris Dataset

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data
df = pd.read_csv(url, names=['sepal length',
                             'sepal width',
                             'petal length',
                             'petal width',
                             'target'])
df.head()
```

```
##    sepal length  sepal width  petal length  petal width        target
## 0           5.1          3.5           1.4          0.2  Iris-setosa
## 1           4.9          3.0           1.4          0.2  Iris-setosa
## 2           4.7          3.2           1.3          0.2  Iris-setosa
## 3           4.6          3.1           1.5          0.2  Iris-setosa
## 4           5.0          3.6           1.4          0.2  Iris-setosa
```

The Iris Dataset

```
features = ['sepal length', 'sepal width', 'petal length', 'petal width']
# Separating out the features
x = df[features]
y = df['target']
x.head()
```

```
##    sepal length  sepal width  petal length  petal width
## 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
```

Correlation Between Variables

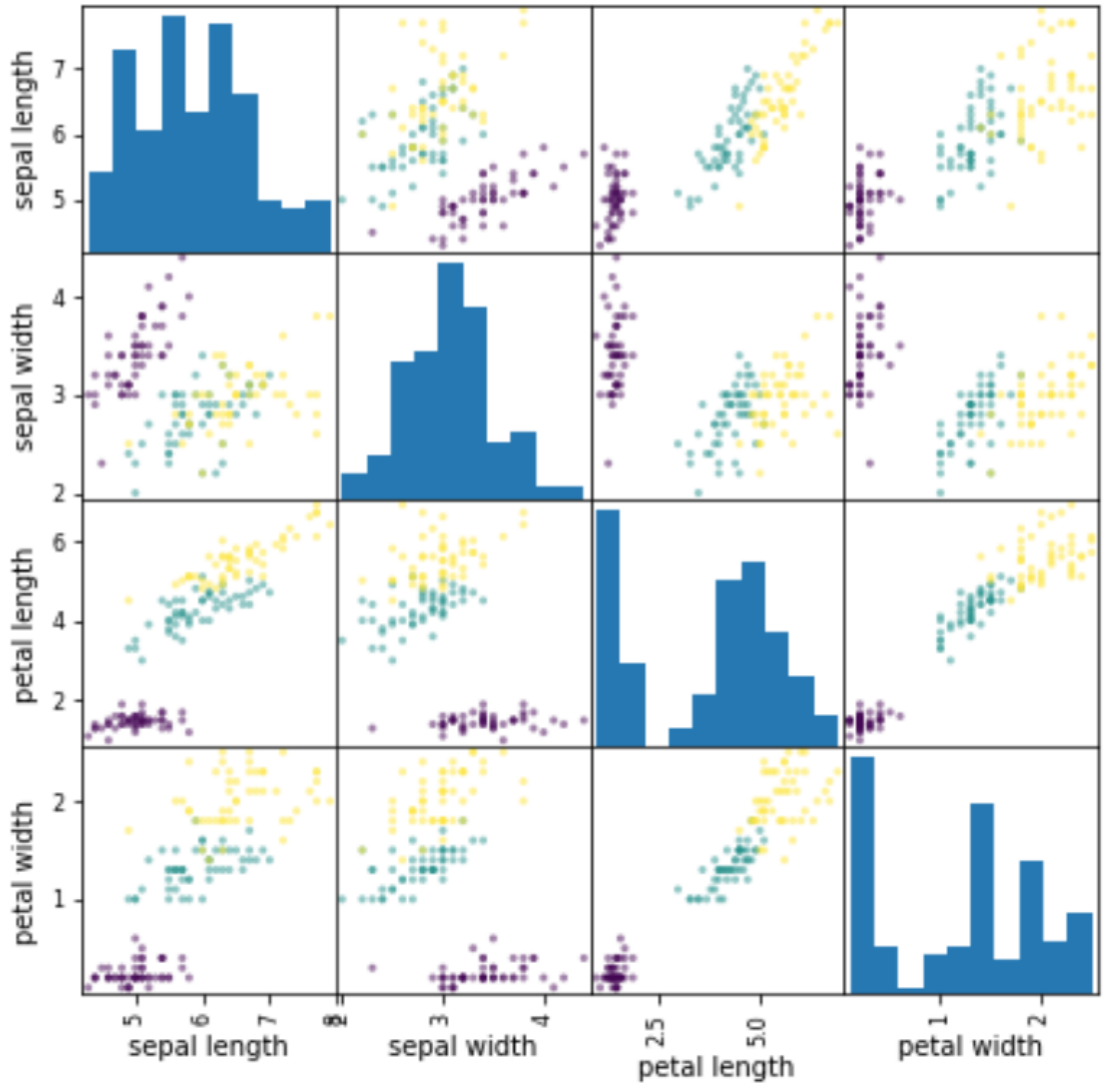
```
corr = x.corr()
corr
```

```
##                sepal length  sepal width  petal length  petal width
## sepal length       1.000000    -0.109369      0.871754     0.817954
## sepal width       -0.109369     1.000000     -0.420516    -0.356544
## petal length       0.871754    -0.420516      1.000000     0.962757
## petal width        0.817954    -0.356544      0.962757     1.000000
```

Correlation Between Variables

```
from sklearn.preprocessing import LabelEncoder
LE = LabelEncoder()
pd.plotting.scatter_matrix(x, c = LE.fit_transform(y))
```

Correlation Between Variables



Correlation Between Variables

- The variables petal length, petal width, and sepal length have high positive correlations.
- The variable sepal length is less correlated with the other variables; this correlation is in the other direction
- Becasue the variables are highly correlation we can use PCA to reduce the dimmensionality of our data.

Scale

The first thing we need to do before performing PCA is to scale the data. We will use StandardScaler() from sklearn, which subtracts the mean from a variable and divides by the standard deviation. This leaves us with a variable with zero mean and unit variance.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(x)
```

```
## StandardScaler(copy=True, with_mean=True, with_std=True)
```

```
x_scaled=scaler.transform(x)
```

Scale

```
sepal_length = x['sepal length']
sepal_length.mean()
```

```
## 5.843333333333335
```

```
sepal_length.std()
```

```
## 0.8280661279778629
```

```
scaled_sepal_length = x_scaled[:,0]
scaled_sepal_length.mean()
```

```
## -4.736951571734001e-16
```

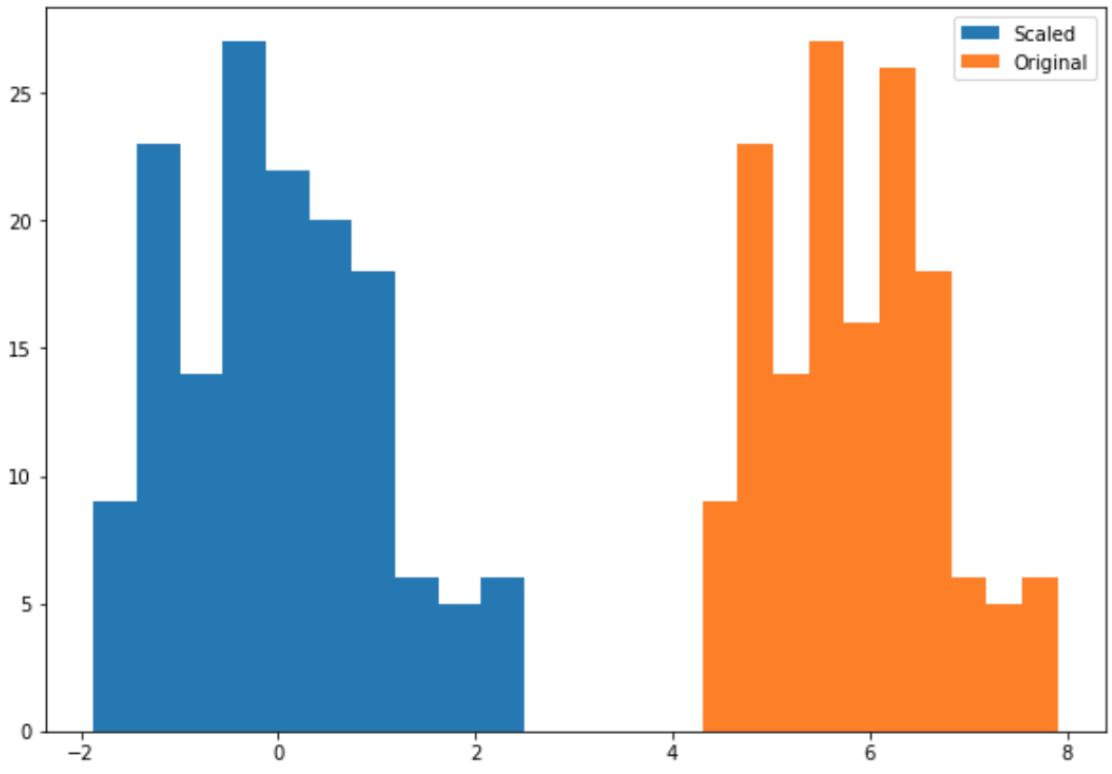
```
scaled_sepal_length.std()
```

```
## 1.0
```

Scale

```
plt.hist(scaled_sepal_length, label = 'Scaled')
plt.hist(sepal_length, label = 'Original')
plt.legend()
```

Scale



PCA

```
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(x_scaled)
```

```
## PCA(copy=True, iterated_power='auto', n_components=None, random_state=None,
##   svd_solver='auto', tol=0.0, whiten=False)
```

Variance Explained / Scree Plot

The first thing we will figure out is how many components we need. We can do this by looking at the variance explained by the first x components.

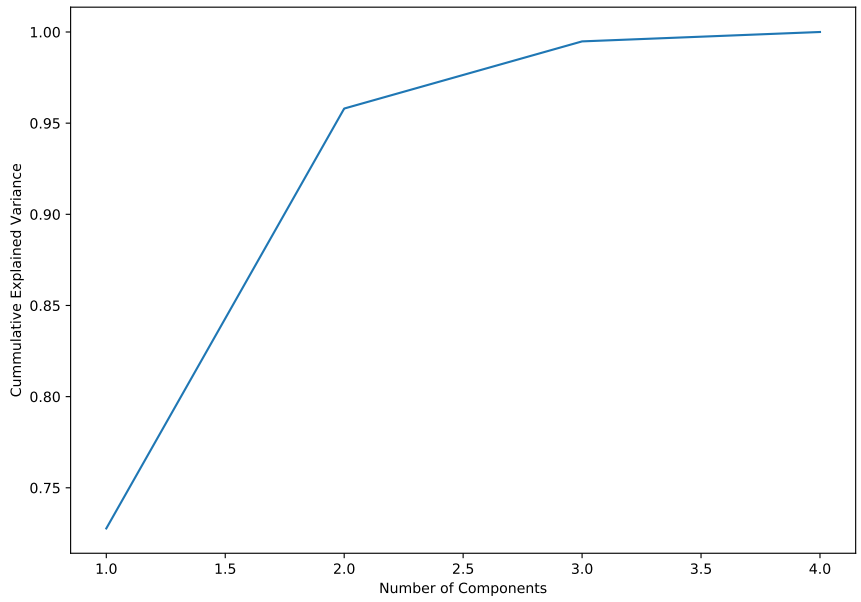
```
print(pca.explained_variance_ratio_)
```

```
## [0.72770452 0.23030523 0.03683832 0.00515193]
```

Variance Explained / Scree Plot

We can look for the 'elbow' on the scree plot to determine how many components to keep in our analysis. From the scree plot we decide to keep two components.

```
components = np.arange(1,5)
plt.plot(components, np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Cummulative Explained Variance')
```



PCA

```
pca = PCA(n_components=2)
pca.fit(x_scaled)
```

```
## PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
##   svd_solver='auto', tol=0.0, whiten=False)
```

PCA: Loadings

- The **first principal component** of a set of features $X_1, X_2, \ldots, X_p$ is the normalized linear combination of the features:

$$Z_1 = \phi_{11} X_1 + \phi_{21} X_2 + \ldots + \phi_{p1} X_p$$

that has the largest variance. By normalized, we mean that $\sum_{j=1}^{p} \phi_{j1}^2 = 1$.

- We refer to the elements $\phi_{11}, \ldots, \phi_{p1}$ as the **loadings** of the first principal component; together, the loadings make up the **principal component loading vector**, $\phi_1 = (\phi_{11}, \phi_{21}, \ldots, \phi_{p1})^T$.
- We constrain the loadings so that their sum of squares is equal to one, since otherwise setting these elements to be arbitrarily large in absolute value could result in an arbitrarily large variance.

PCA: Loadings

The first row are the loadings of the first PC and the second row are the loadings of the second PC.

```
pca_loadings = pca.components_
pca_loadings
```

```
## array([[ 0.52237162, -0.26335492,  0.58125401,  0.56561105],
##        [ 0.37231836,  0.92555649,  0.02109478,  0.06541577]])
```

```
sum(np.square(pca_loadings[0,0:4]))
```

```
## 1.0000000000000004
```

```
sum(np.square(pca_loadings[1,0:4]))
```

```
## 0.9999999999999998
```

PCA: Loadings

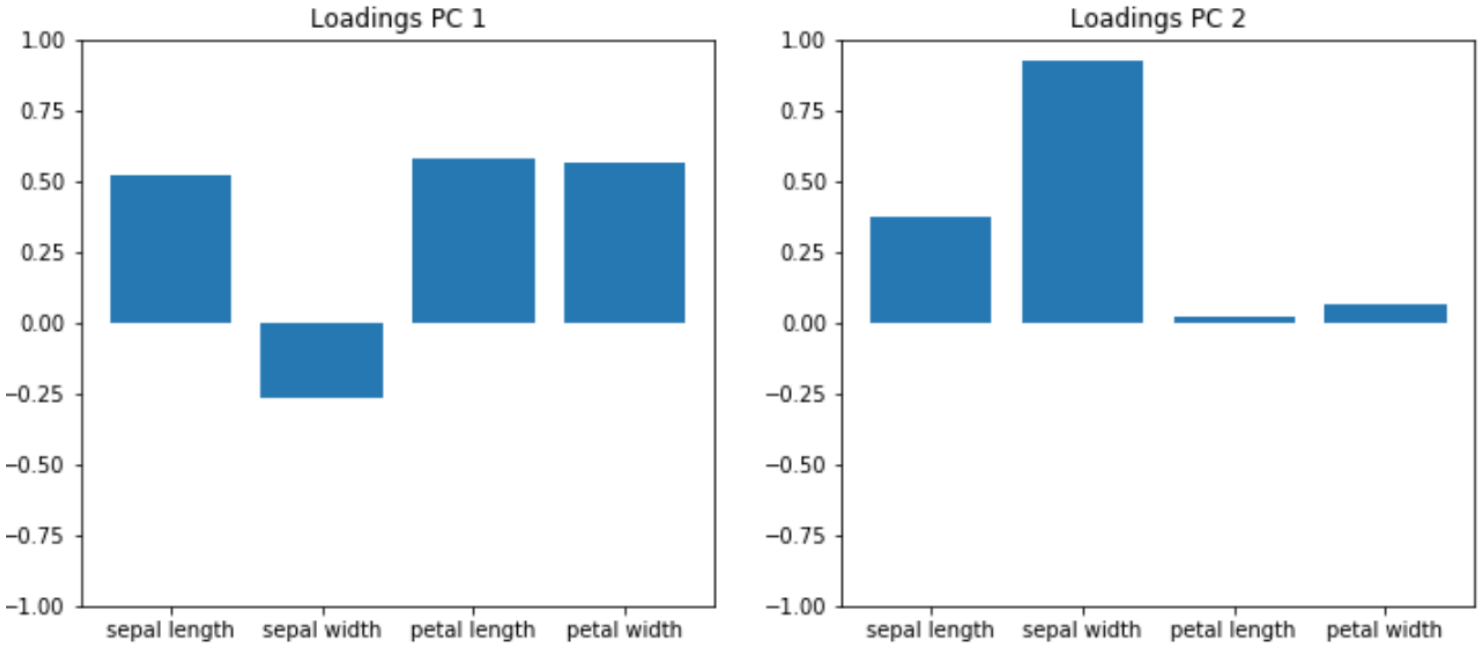
We will next plot the loadings for each variable for the first and second PC.

```
plt.figure()

for i in range(2):

    plt.subplot(1,2,i+1)
    heights = np.squeeze(pca_loadings[i,:])
    bars = ['sepal length','sepal width', 'petal length', 'petal width']
    x_pos = np.arange(len(bars))
    plt.bar(x_pos, heights)
    plt.xticks(x_pos, bars)
    plt.title("Loadings PC "+str(i+1))
    plt.ylim(-1,1)
    print(i)
```

PCA: Loadings



PCA: Scores

- The loading vector ϕ_1 with elements $\phi_{11}, \phi_{21}, \ldots, \phi_{p1}$ defines a **direction in feature space** along which the data vary the most.
- If we project the n data points $x_1, \ldots, x_n$ onto this direction, the projected values are the **principal component scores** $z_{11}, \ldots, z_{n1}$ themselves, i.e.

$$z_{i1} = \phi_{1i} x_{i1} + \phi_{2i} x_{i2} + \ldots + \phi_{pi} x_{ip}$$

PCA: Scores

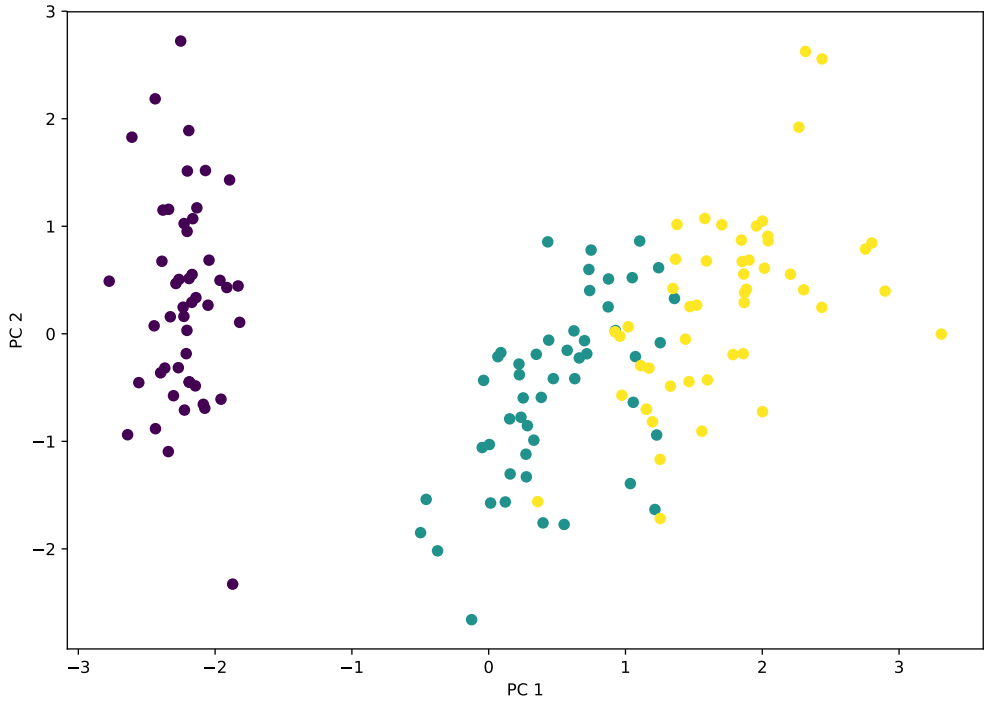
Note that we have collapsed the dimmensionality of our space from 4 dimmensions to 2 dimmensions.

```
pca_scores = pca.fit_transform(x_scaled)
pd.DataFrame(pca_scores).head()
```

```
##           0         1
## 0 -2.264542  0.505704
## 1 -2.086426 -0.655405
## 2 -2.367950 -0.318477
## 3 -2.304197 -0.575368
## 4 -2.388777  0.674767
```

PCA: Scores

```
from sklearn.preprocessing import LabelEncoder
LE = LabelEncoder()
plt.scatter(pca_scores[:,0], pca_scores[:,1], c = LE.fit_transform(y))
plt.xlabel('PC 1')
plt.ylabel('PC 2')
```



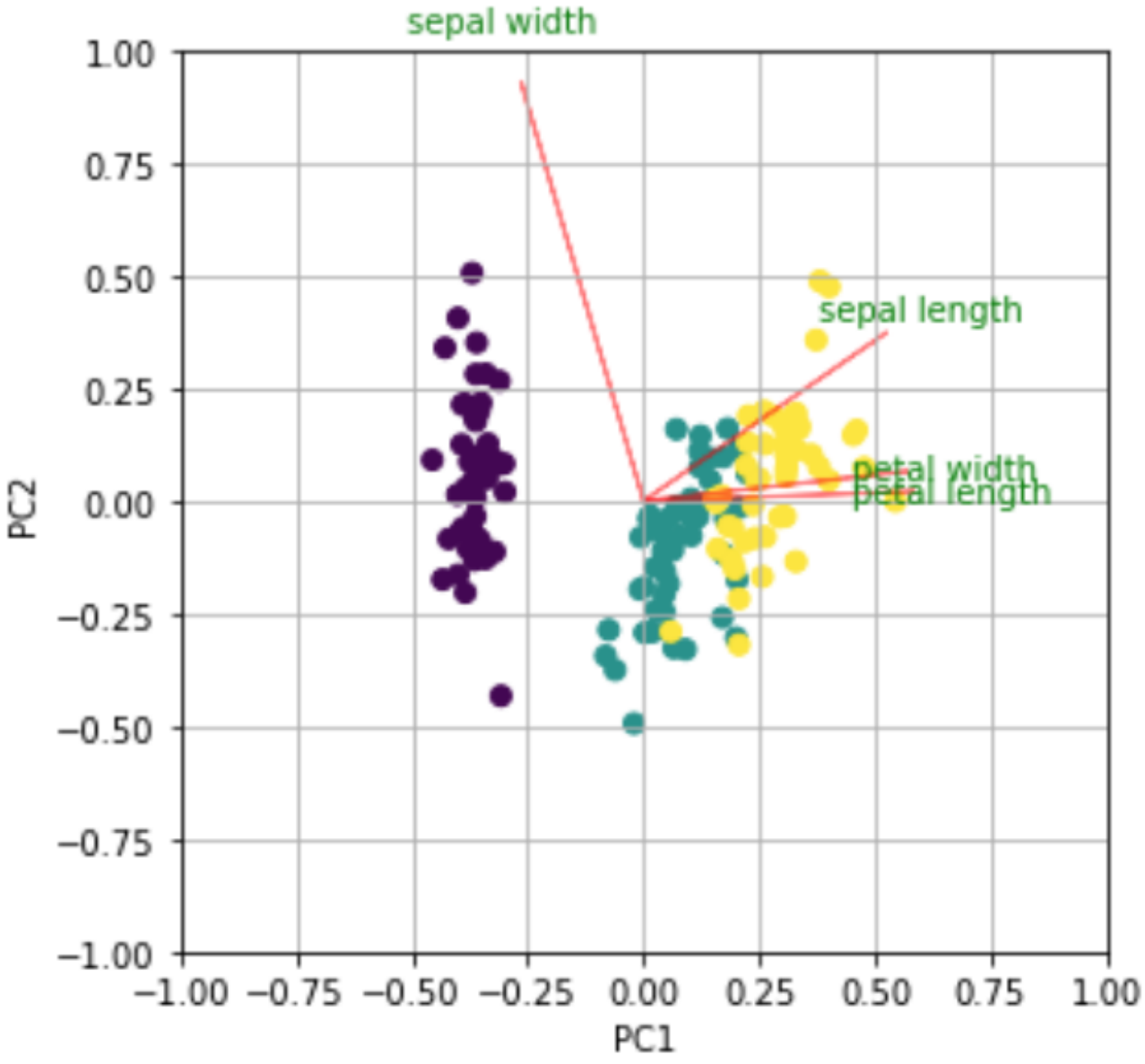
Biplot

Code from https://stackoverflow.com/questions/39216897/plot-pca-loadings-and-loading-in-biplot-in-sklearn-like-rs-autoplot

```
def myplot(score,coeff,labels=None):
    xs = score[:,0]
    ys = score[:,1]
    n = coeff.shape[0]
    scalex = 1.0/(xs.max() - xs.min())
    scaley = 1.0/(ys.max() - ys.min())
    plt.scatter(xs * scalex,ys * scaley, c = LE.fit_transform(y))
    for i in range(n):
        plt.arrow(0, 0, coeff[i,0], coeff[i,1],color = 'r',alpha = 0.5)
        if labels is None:
            plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, features[i],
                     color = 'g', ha = 'center', va = 'center')
        else:
            plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, labels[i],
                     color = 'g', ha = 'center', va = 'center')
    plt.xlim(-1,1)
    plt.ylim(-1,1)
    plt.xlabel("PC{}".format(1))
    plt.ylabel("PC{}".format(2))
    plt.grid()

myplot(pca_scores[:,0:2],np.transpose(pca.components_[0:2, :]))
```

Biplot


sepal width
sepal length
petal width
petal length
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
PC2
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00
PC1

Biplot

- From the loadings, we see the sepal width is less correlate with the other variables. Petal width and petal length appear to be the most correlated.
- The first PC is some combination of sepal length, petal length, and petal width. The second PC is dominated more by sepal width.
- The scores are able to discriminate between the different types of Iris flowers.

Notes

- In Lab and for your Final Project we will do PCA in cases where we have many more variables – the benifit of dimmension reduction becomes even more apparent there!
- You can cluster the score on the PCs to make meaningul groups (can work better than clustering directly on the data when you have many observations)
- You can also use the scores on the PCs in regression models, which you will see in Data Science II for supervised learning.

In Class Exercise

- Explore the data on pizza measurements (provided) from https://data.world/sdhilip/pizza-datasets/
- Perform PCA on the data. How many PCs do you need? Make and interpret a biplot.