BIOL4062/5062 Analysis of Biological Data: Assignment Guide

Reid Steele

2024-11-04

Table of contents

Introduction

Welcome to the assignment guide for BIOL4062/5062: Analysis of Biological Data.

This book is designed to walk you through the assignments for this class. It is a resource to help you figure out how to code for your assignments, and bring attention to key questions to ask yourself as you interpret your results, both statistically and biologically. Keep in mind that there are always different ways to get to the right answer with coding. This book is not a monolith, and you do not need to follow it if you don't want to (more in Assignment Guidelines), but make sure what you are doing is clear and sufficiently analogous to this guide, else you lose marks for being unclear, or running the wrong analyses.

Without further ado, let's get into it!

Part I

Assignment 1

1 Assignment 1a: Principal Components Analysis

Assignment 1a focuses on Principal Components Analysis (PCA). Think of PCA as a method of finding associations between data series.

For this tutorial, we're going to use the dataset in `fishcatch.csv`.

1.1 Looking at the Data

With any data analysis, step 1 is always to look at your data:

```
# Load in data
data = read.csv('fishcatch.csv')

# View data structure
head(data)
```

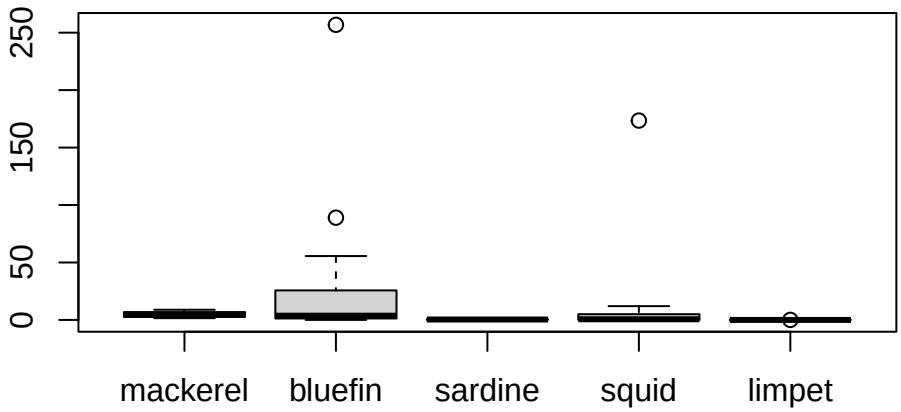
```
  Hauls mackerel bluefin sardine squid limpet
1     1    1.851   55.60   0.058  6.00 0.0004
2     2    1.925    1.20   0.252  0.08 0.0027
3     3    2.506    1.56   0.133  0.06 0.0015
4     4    1.537   30.00   0.064  9.35 0.0013
5     5    1.795    0.04   0.086  4.70 0.0022
6     6    3.371   45.00   0.078  7.66 0.0006
```

```
dim(data)
```

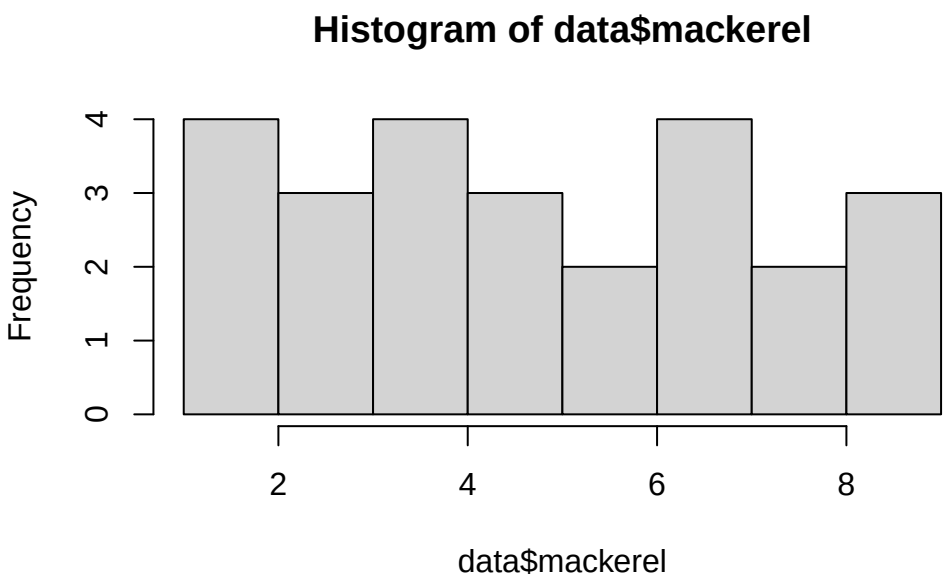
```
[1] 25  6
```

Our data is a 25 row, 6 column data frame, describing catch of 5 different fisheries species (columns 2-6) caught across 25 hauls (column 1). We want to know if certain species are associated with each other. Lets look a little deeper at the data:

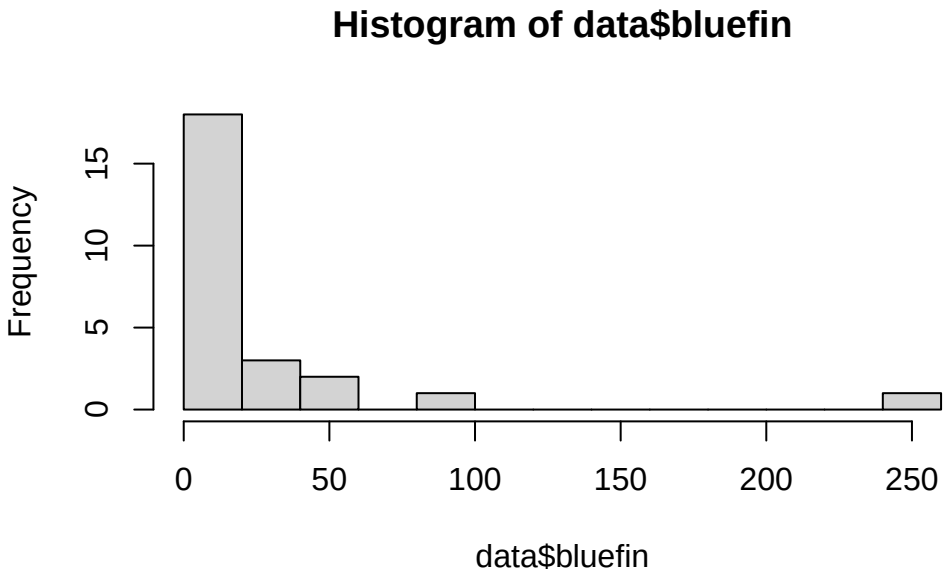
```r
# Generate boxplots
boxplot(data[,-1]) # Exclude haul
```



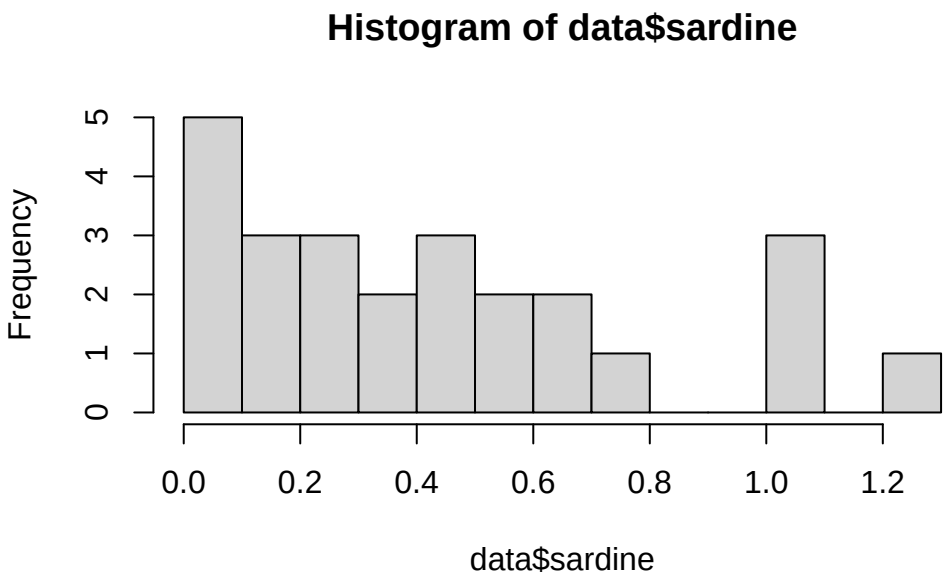

```r
# look at data distribution
# par(mfrow = c(3,2)) # 1 column 5 row grid plot
hist(data$mackerel, breaks = 10)
```



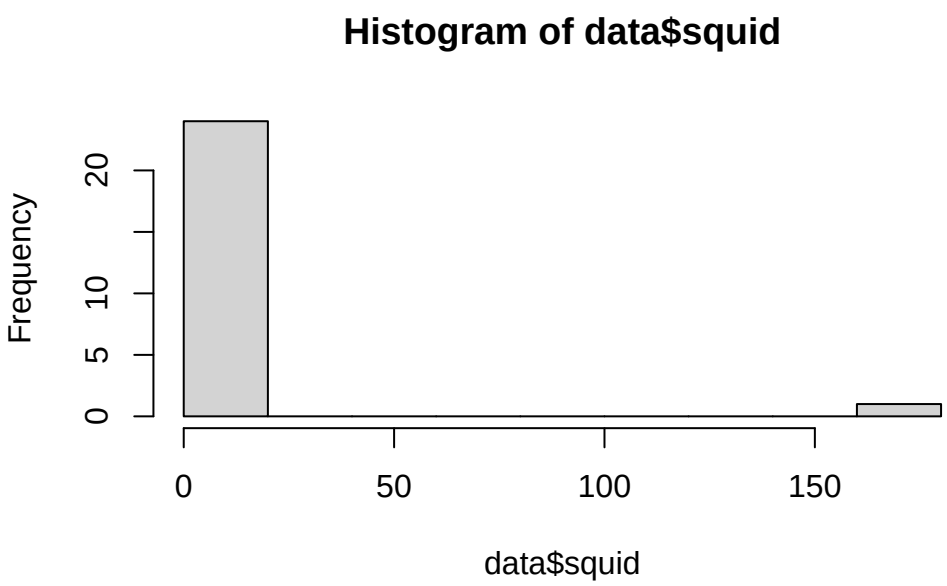

```r
hist(data$bluefin, breaks = 10)
```



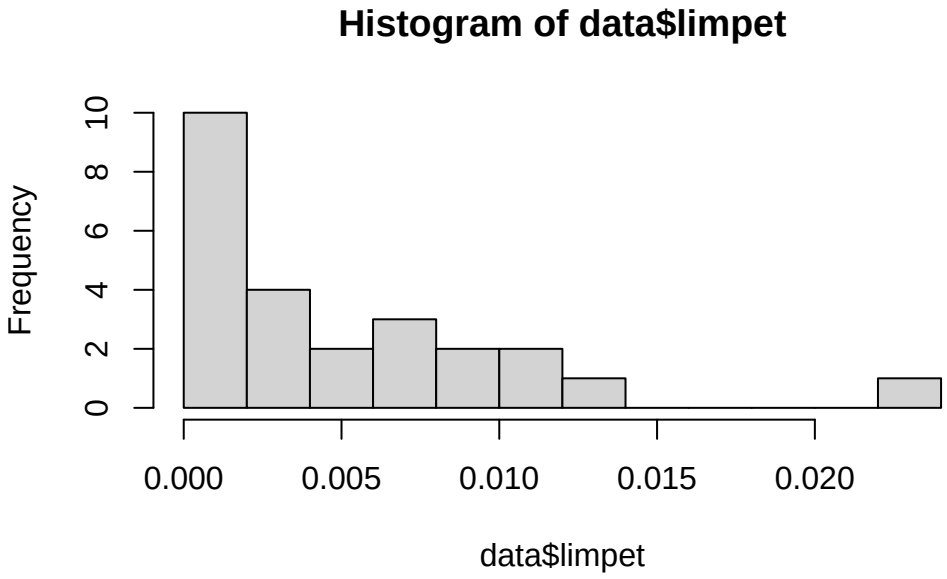
```r
hist(data$sardine, breaks = 10)
```




```r
hist(data$squid, breaks = 10)
```




```r
hist(data$limpet, breaks = 10)
```



A few things are immediately obvious from looking at our data:

1. There are some large outliers
2. The data scales vary greatly across species
3. The species all have relatively different distributions, none of which look normal.

Are these issues? How do we fix them?

1.2 Transformations

Look back at the PCA lecture. What are potential problems with PCA?

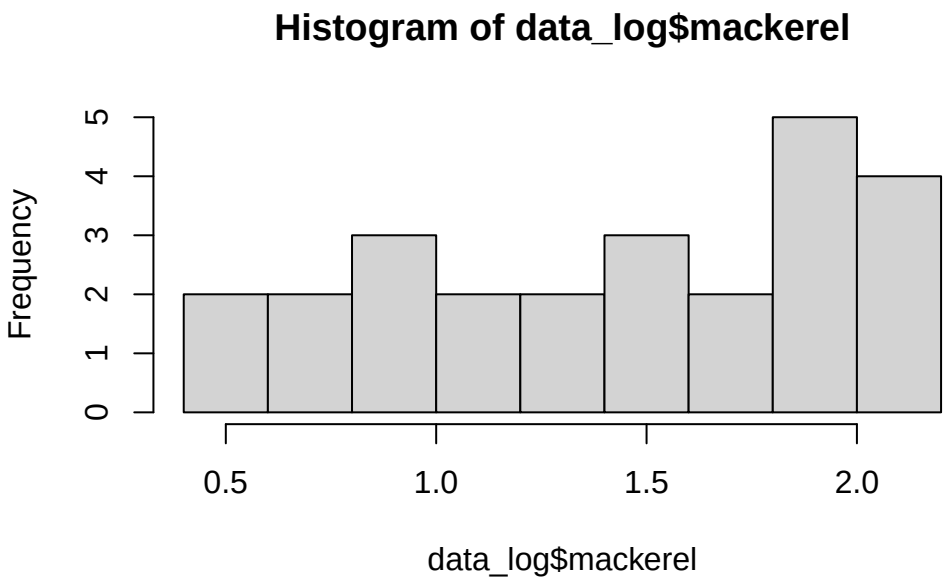
1. Covariance Matrix PCA requires data to be in the same units
2. Normality is desirable, but not essential
3. Precision is desireable, but not essential
4. Many zeroes in the data

We can fix issue 1 by logging our data:

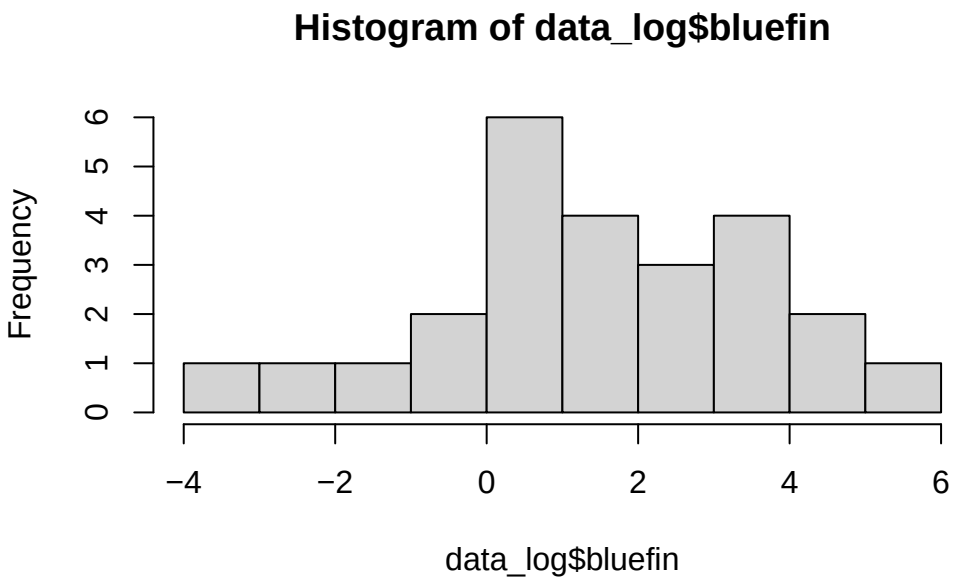
```
# Create a new data object so we can log the data
data_log = data

# Log data
data_log[,-1] = log(data_log[,-1]) # Remember to exclude haul
```

Now that we've transformed the data, let's check for normality again:

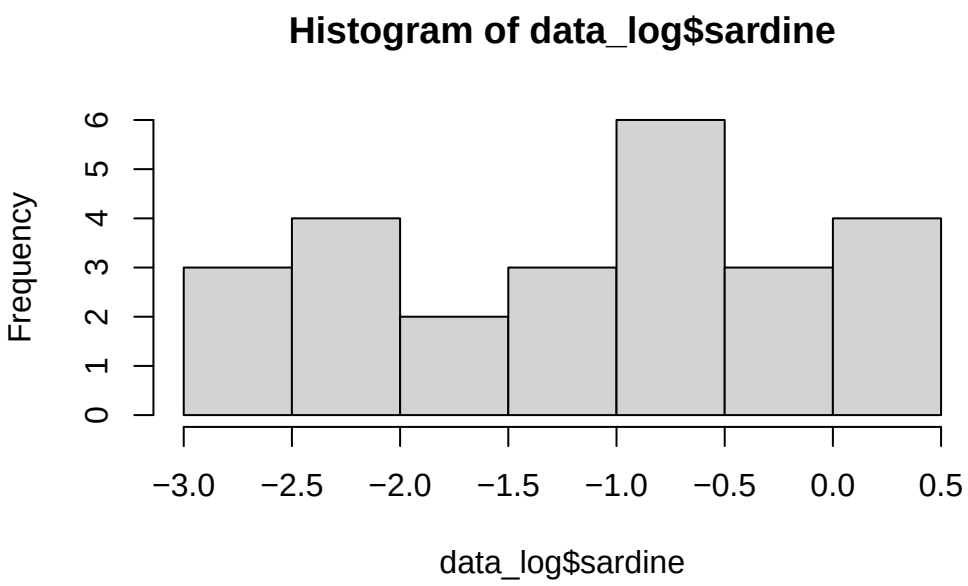
```
# Generate histograms
# par(mfrow = c(3,2)) # 1 column 5 row grid plot
hist(data_log$mackerel, breaks = 10)
```



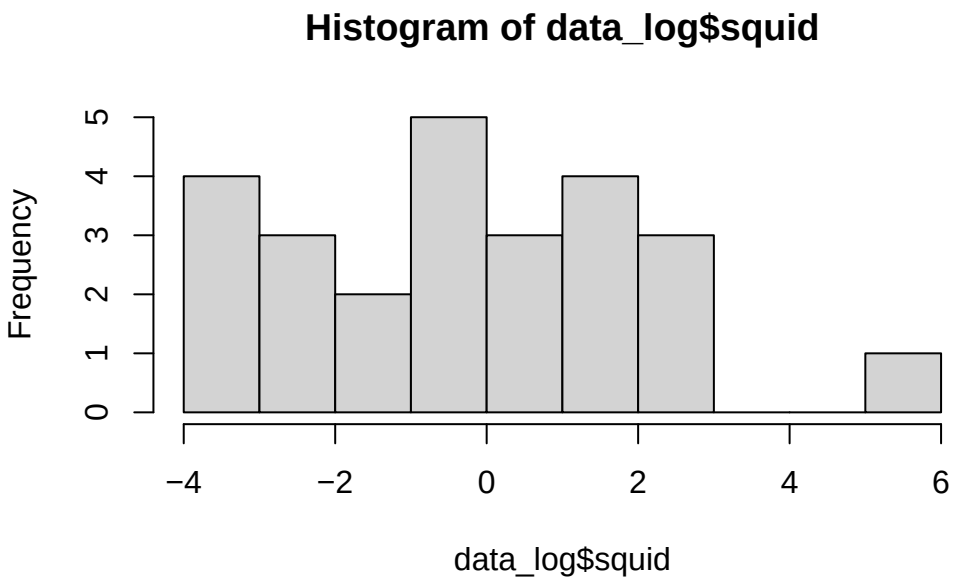

```r
hist(data_log$bluefin, breaks = 10)
```



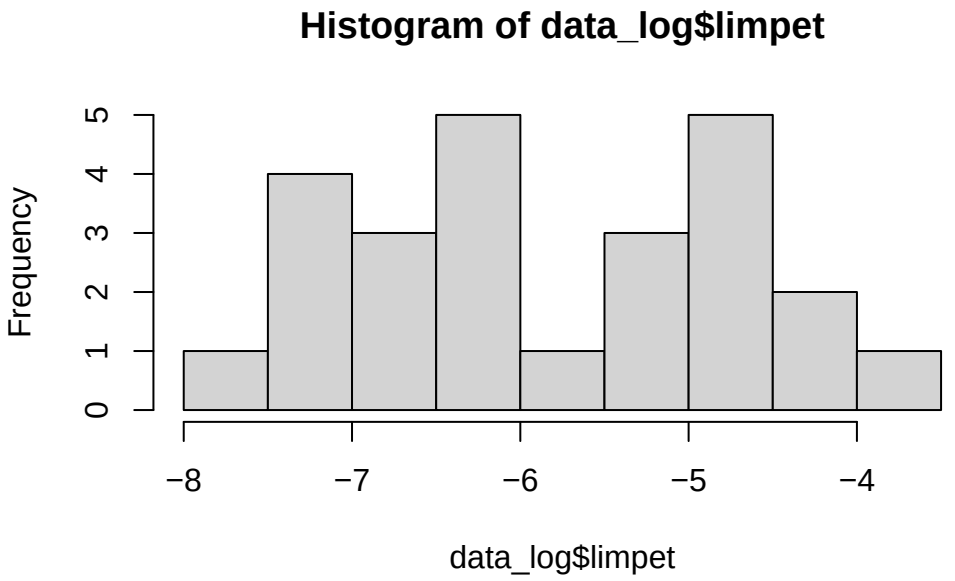
```r
hist(data_log$sardine, breaks = 10)
```




```r
hist(data_log$squid, breaks = 10)
```




```r
hist(data_log$limpet, breaks = 10)
```



These look much better. We can also confirm this statistically:

```
# Generate histograms
shapiro.test(data_log$mackerel)
```

```
    Shapiro-Wilk normality test

data:  data_log$mackerel
W = 0.9425, p-value = 0.1691
```

```
shapiro.test(data_log$bluefin)
```

```
    Shapiro-Wilk normality test

data:  data_log$bluefin
W = 0.98186, p-value = 0.9193
```

```
shapiro.test(data_log$sardine)
```

```
    Shapiro-Wilk normality test

data:  data_log$sardine
W = 0.94113, p-value = 0.1572
```

```
shapiro.test(data_log$squid)
```

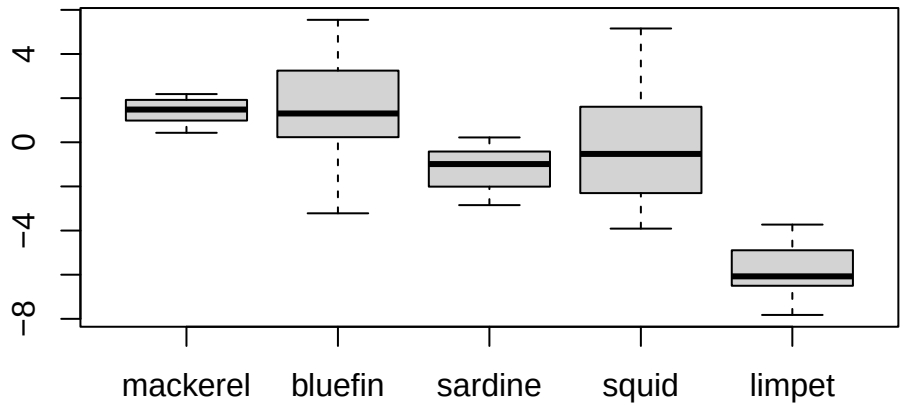
```
    Shapiro-Wilk normality test

data:  data_log$squid
W = 0.96226, p-value = 0.4613
```

```
shapiro.test(data_log$limpet)
```

```
    Shapiro-Wilk normality test

data:  data_log$limpet
W = 0.96437, p-value = 0.5082
```

All 5 species fail to reject the null hypothesis that the data are normally distributed. Logging the data also helps deal with the outliers:

```r
# Generate boxplots
boxplot(data_log[,-1])
```



Note that we can only log the data if there are no zeroes:

```r
# Generate test data
data_test = data; data_test[1,6] = 0 # Change the first limpet value to 0

# Try to log the data
data_test[1,] # Print first row
```

```
  Hauls mackerel bluefin sardine squid limpet
1     1    1.851    55.6   0.058     6      0
```

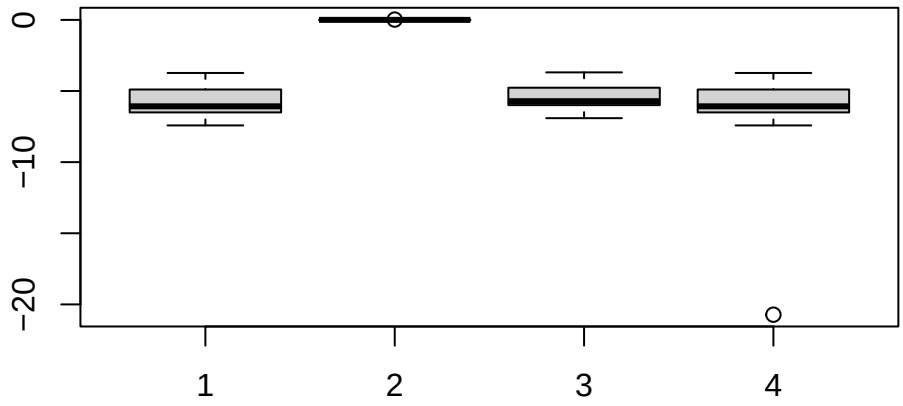
```
log(data_test[,-1])[1,] # Print logs of the first row
```

```
  mackerel  bluefin   sardine    squid limpet
1 0.615726 4.018183 -2.847312 1.791759   -Inf
```

log(0) returnes negative infinity. That's going to be a problem later in our analysis. We can fix that by adding a small increment before taking the log. Keep in mind though that each species has a different magnitude in this dataset, and adding an inappropriate increment could cause us trouble later:

```
# Test boxplots of different increments
boxplot(log(data_test$limpet), # Warning because of -Inf
        log(data_test$limpet + 1),
        log(data_test$limpet + 0.001),
        log(data_test$limpet + 0.000000001))
```

```
Warning in bplt(at[i], wid = width[i], stats = z$stats[, i], out =
z$out[z$group == : Outlier (-Inf) in boxplot 1 is not drawn
```



If the increment is too big, we eliminate the variance in our data. If the increment is to small, we create an outlier.

1.3 Running PCA

Now that we've checked and transformed our data, we're ready to run PCA. There are two kinds of PCA: We can run PCA on the Covariance Matrix, or the Correlation Matrix.

1.3.1 Covariance Matrix

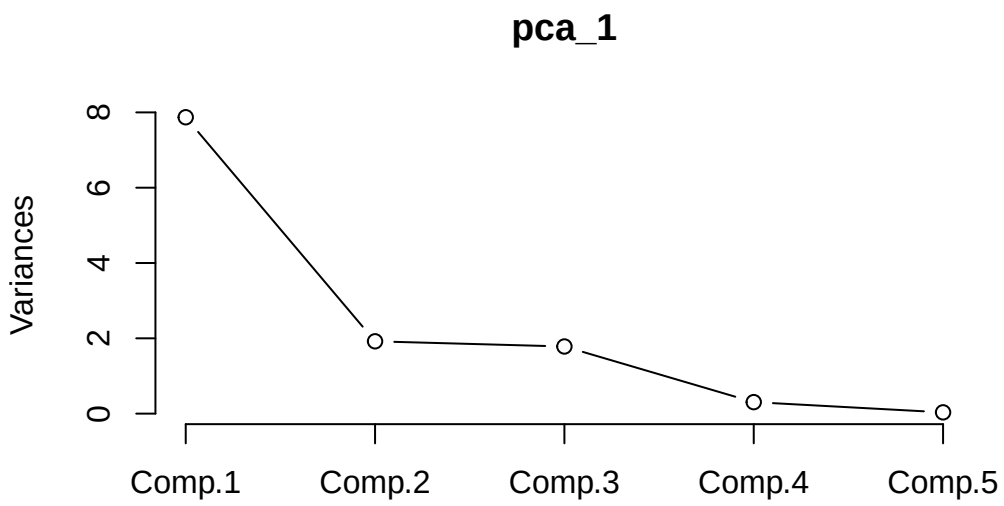
We can run PCA on the covariance matrix as follows:

```
# Run PCA - Covariance
pca_1 = princomp(data_log[,-1]) # We don't want haul in our PCA!
summary(pca_1)
```

```
Importance of components:
                          Comp.1    Comp.2    Comp.3     Comp.4      Comp.5
Standard deviation     2.8055354 1.3857803 1.3351790 0.55247629 0.182937780
Proportion of Variance 0.6607195 0.1612035 0.1496458 0.02562199 0.002809263
Cumulative Proportion  0.6607195 0.8219229 0.9715687 0.99719074 1.000000000
```

Running a summary on our PCA gives us the standard deviation of each principal component, the proportion of variance explained by each principal component, and the cumulative variance explained as we add each component. Here, we see the first principal component explains 66% of the variance. The second explains 16%, which adds up to 82% with the first component, and so on up to component 5. We can visualize the cumulative variance explained with a scree plot:

```
# Generate scree plot
plot(pca_1, type = 'l') # Scree is built into the plot for PCA
```



We see most of the variance is explained by component 1, then a similar lesser amount is explained by 2 and 3, followed by another drop to 4 and 5.

```
# Print loadings
print(loadings(pca_1),cutoff=0.00) #all loadings!
```

```
Loadings:
         Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
mackerel  0.018  0.294  0.047  0.527  0.796
bluefin  -0.654  0.136  0.739 -0.089 -0.020
sardine   0.060  0.626 -0.015  0.520 -0.577
squid    -0.745  0.049 -0.664  0.029  0.018
limpet    0.116  0.707 -0.102 -0.665  0.183

               Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
SS loadings       1.0    1.0    1.0    1.0    1.0
Proportion Var    0.2    0.2    0.2    0.2    0.2
Cumulative Var    0.2    0.4    0.6    0.8    1.0
```

The PCA loadings are the correlations between the variables and each component. Here, we see bluefin and squid are strongly negatively correlated with component 1, while mackerel,

sardine, and limpet are weakly positively correlated with component 1. We can continue this type of interpretation through the other components as well.

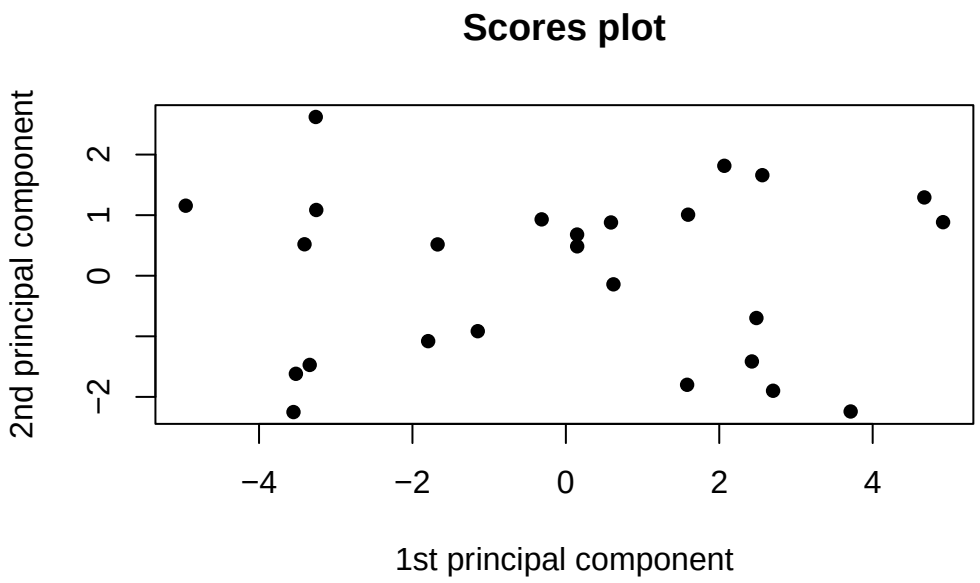
Our PCA object also contains the PCA scores for each individual data point:

```
# Print PCA scores
head(pca_1$scores)
```

```
       Comp.1     Comp.2     Comp.3     Comp.4      Comp.5
[1,] -3.551007 -2.2507718  0.6725187 -0.1223707 -0.06754585
[2,]  2.484116 -0.6980669  0.4886052 -0.3932653 -0.53609582
[3,]  2.425206 -1.4149241  0.9556963 -0.2274184 -0.07560834
[4,] -3.339469 -1.4723306 -0.2089590 -0.8854067 -0.03598265
[5,]  1.580988 -1.8002844 -4.6958830 -0.4313924  0.13590020
[6,] -3.519487 -1.6187995  0.3362833  0.1040827  0.32134755
```

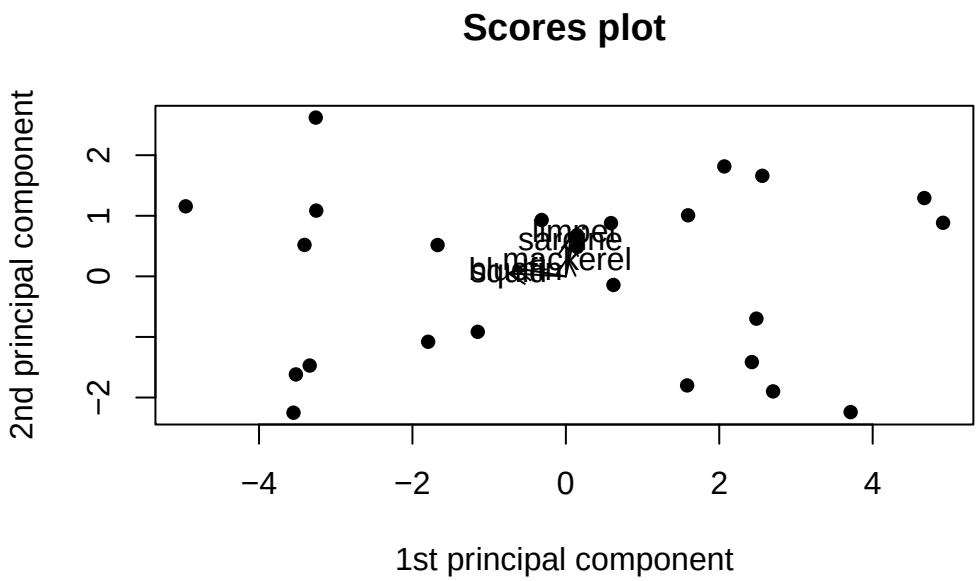
Scores are the value of each data point on each principal component. Lets try plotting them:

```
# Plot scores - components 1 and 2
plot(pca_1$scores[,1], # Scores on component 1
     pca_1$scores[,2], # Scores on component 3
     pch=16, # Point 16 (colored circle)
     xlab="1st principal component",ylab="2nd principal component",main="Scores plot") # A
```



This generates a scatterplot showing us the value of each data point in principal components 1 (x) and 2 (y). Now lets add on the loadings:

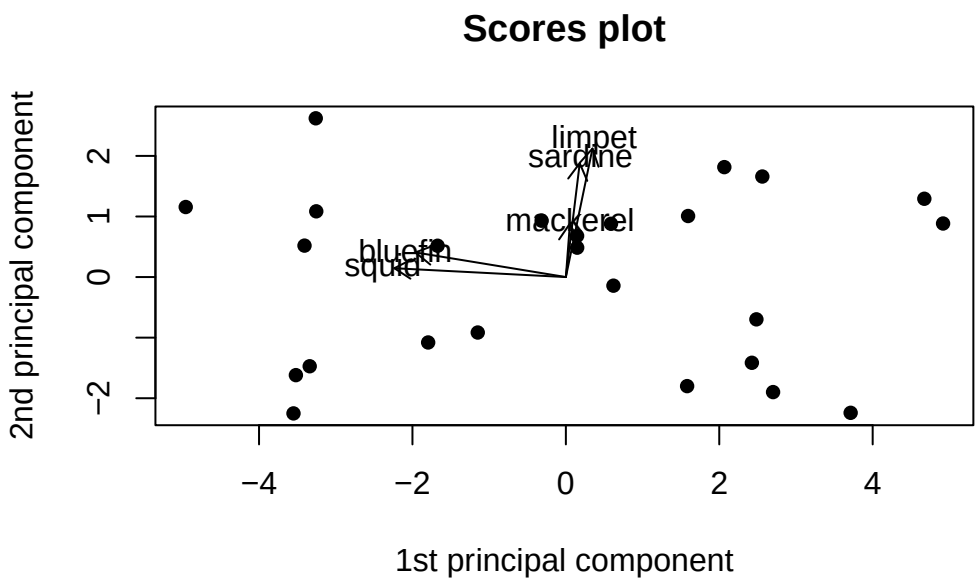
```
# Plot scores - components 1 and 2
plot(pca_1$scores[,1], # Scores on component 1
     pca_1$scores[,2], # Scores on component 3
     pch=16, # Point 16 (colored circle)
     xlab="1st principal component",ylab="2nd principal component",main="Scores plot") # A

# Add loadings to plot
arrows(0,0, # Draw arrows from zero
       pca_1$loadings[,1], # Draw to PC1 loading in X
       pca_1$loadings[,2], # Draw to PC2 loading in Y
       col="black", length = 0.1) # Arrow color and arrowhead length
text(pca_1$loadings[,1],pca_1$loadings[,2],names(data_log[,-1]),cex=1.0 ,col="black") # Ad
```



The arrows are a little small, so let's add a scaling factor:

```
# Plot scores - components 1 and 2
plot(pca_1$scores[,1], # Scores on component 1
     pca_1$scores[,2], # Scores on component 3
```

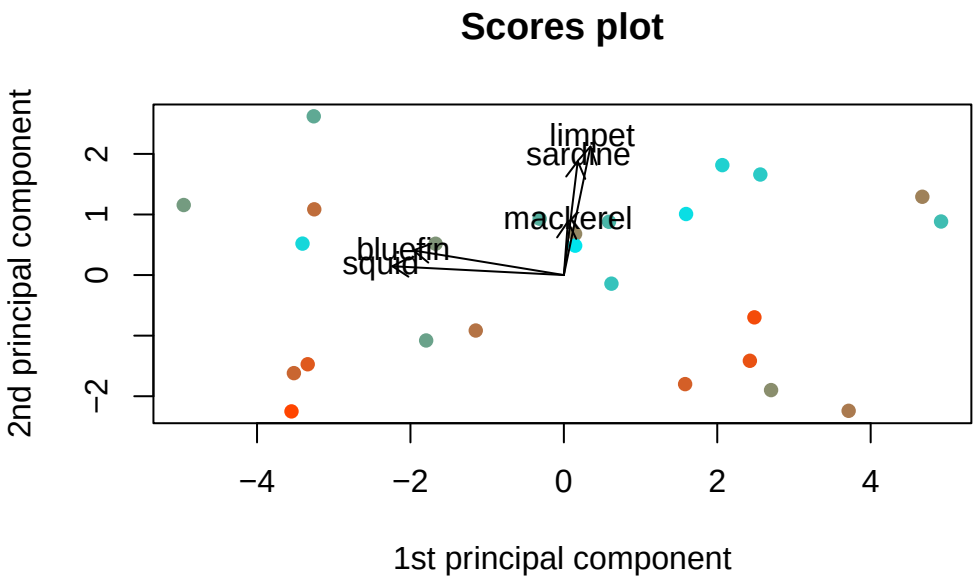
```
     pch=16, # Point 16 (colored circle)
     xlab="1st principal component",ylab="2nd principal component",main="Scores plot") # A

# Add loadings to plot
sf = 3 # Scaling factor
sft = 3.2 # Scaling factor for text
arrows(0,0, # Draw arrows from zero
       pca_1$loadings[,1]*sf, # Draw to PC1 * scaling factor loading in X
       pca_1$loadings[,2]*sf, # Draw to PC2 * scaling factor loading in Y
       col="black", length = 0.1) # Arrow color and arrowhead length
text(pca_1$loadings[,1]*sft,pca_1$loadings[,2]*sft, names(data_log[,-1]), cex=1.0, col="bl
```



What about the haul number? Does that have an effect? Let's try adding that on as well:

```
# Create a color palette
colfunc = colorRampPalette(c('orangered1', 'turquoise2'))

# Plot scores - components 1 and 2
plot(pca_1$scores[,1], # Scores on component 1
     pca_1$scores[,2], # Scores on component 3
     pch=16, # Point 16 (colored circle)
```

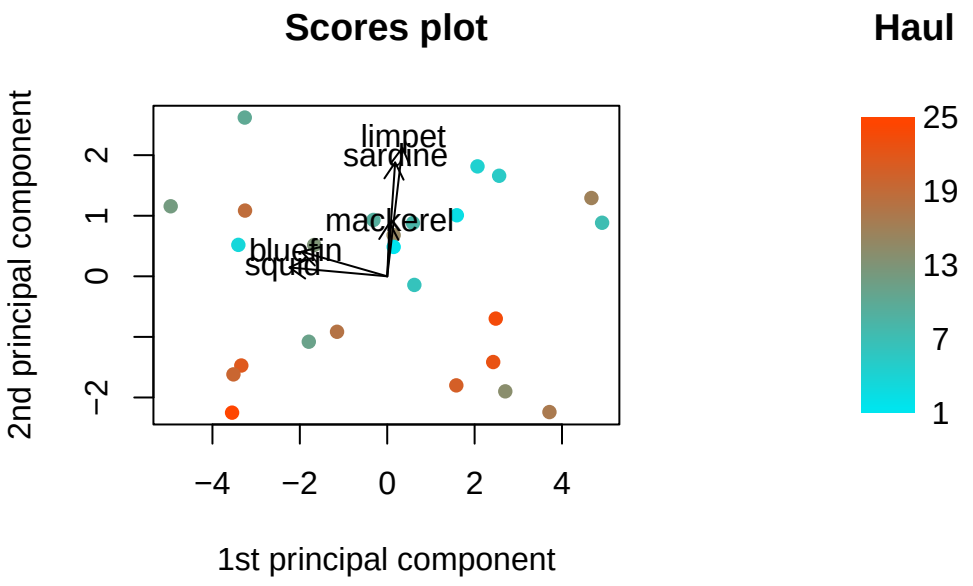
```
     col = colfunc(nrow(pca_1$scores)), # Color points by haul using our color palette
     xlab="1st principal component",ylab="2nd principal component",main="Scores plot") # A

# Add loadings to plot
sf = 3 # Scaling factor
sft = 3.2 # Scaling factor for text
arrows(0,0, # Draw arrows from zero
       pca_1$loadings[,1]*sf, # Draw to PC1 * scaling factor loading in X
       pca_1$loadings[,2]*sf, # Draw to PC2 * scaling factor loading in Y
       col="black", length = 0.1) # Arrow color and arrowhead length
text(pca_1$loadings[,1]*sft,pca_1$loadings[,2]*sft, names(data_log[,-1]), cex=1.0, col="bl
```



Since we used color for haul, we need to add a legend:

```
# Set plot layout
layout(matrix(1:2,ncol=2), # 1 row, 2 columns
       width = c(2,1), # Width
       height = c(1,1)) # Height

# Create a color palette
colfunc = colorRampPalette(c('orangered1', 'turquoise2'))
```

```r
# Plot scores - components 1 and 2
plot(pca_1$scores[,1], # Scores on component 1
     pca_1$scores[,2], # Scores on component 3
     pch=16, # Point 16 (colored circle)
     col = colfunc(nrow(pca_1$scores)), # Color points by haul using our color palette
     xlab="1st principal component",ylab="2nd principal component",main="Scores plot") # A

# Add loadings to plot
sf = 3 # Scaling factor
sft = 3.2 # Scaling factor for text
arrows(0,0, # Draw arrows from zero
       pca_1$loadings[,1]*sf, # Draw to PC1 * scaling factor loading in X
       pca_1$loadings[,2]*sf, # Draw to PC2 * scaling factor loading in Y
       col="black", length = 0.1) # Arrow color and arrowhead length
text(pca_1$loadings[,1]*sft,pca_1$loadings[,2]*sft, names(data_log[,-1]), cex=1.0, col="bl

# Generate legend
legend_image <- as.raster(matrix(colfunc(nrow(pca_1$scores)), ncol=1))
plot(c(0,2),c(0,1),type = 'n', axes = F,xlab = '', ylab = '', main = 'Haul')
text(x=1.5, y =seq(0,1,l=5), labels = seq(1,25,l=5))
rasterImage(legend_image, 0, 0, 1,1)
```



Now we have a completed scores plot with loadings arrows. How would you interpret this plot?

1.3.2 Correlation Matrix

Now let's try the correlation matrix. The correlation matrix performs the same analysis, but on standardized data. The princomp() function does this for us if we set cor = T:

```
# Run PCA - Correlation
pca_2 = princomp(data_log[,-1], cor = T)
summary(pca_2)
```

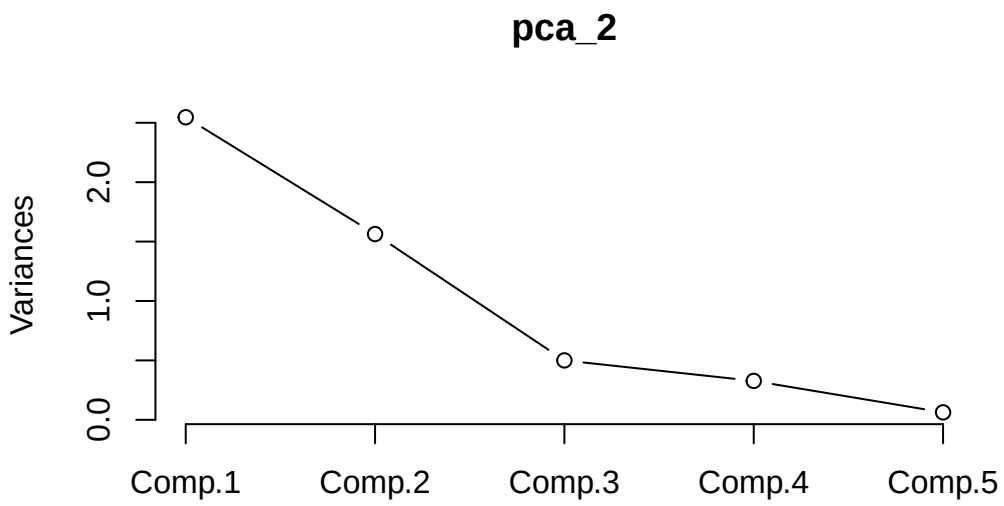
```
Importance of components:
                          Comp.1    Comp.2     Comp.3     Comp.4     Comp.5
Standard deviation     1.595782 1.2503536 0.70708572 0.57220519 0.25041296
Proportion of Variance 0.509304 0.3126768 0.09999404 0.06548376 0.01254133
Cumulative Proportion  0.509304 0.8219809 0.92197491 0.98745867 1.00000000
```

```
# In case you don't believe me, heres the covariance matrix if we pre-standardize the data
pca_test = princomp(scale(data_log[-1]))
summary(pca_test)
```

```
Importance of components:
                          Comp.1    Comp.2     Comp.3     Comp.4     Comp.5
Standard deviation     1.563541 1.2250914 0.69279969 0.56064429 0.24535359
Proportion of Variance 0.509304 0.3126768 0.09999404 0.06548376 0.01254133
Cumulative Proportion  0.509304 0.8219809 0.92197491 0.98745867 1.00000000
```

Now we can go through the same pattern of analyses as we did for covariance:

```
# Generate scree plot
plot(pca_2, type = 'l') # Scree is built into the plot for PCA
```

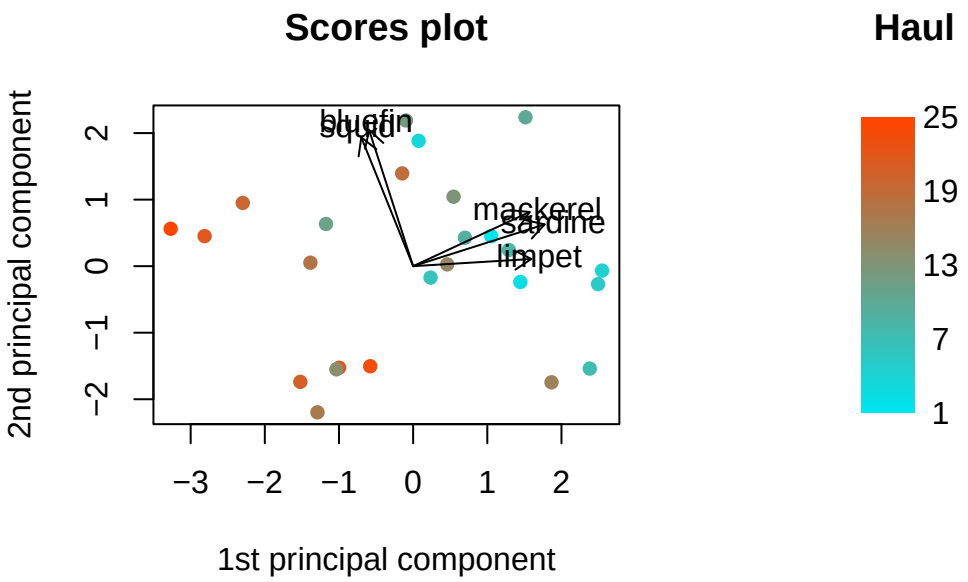



```
# Print loadings
print(loadings(pca_2),cutoff=0.00) #all loadings!
```

```
Loadings:
         Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
mackerel  0.524  0.272  0.527  0.297  0.535
bluefin  -0.198  0.682  0.264 -0.651 -0.050
sardine   0.591  0.209  0.025  0.109 -0.771
squid    -0.233  0.645 -0.472  0.550  0.059
limpet    0.532  0.036 -0.655 -0.416  0.338

                Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
SS loadings        1.0    1.0    1.0    1.0    1.0
Proportion Var     0.2    0.2    0.2    0.2    0.2
Cumulative Var     0.2    0.4    0.6    0.8    1.0
```

```
# Set plot layout
layout(matrix(1:2,ncol=2), # 1 row, 2 columns
       width = c(2,1), # Width
       height = c(1,1)) # Height
```

```r
# Create a color palette
colfunc = colorRampPalette(c('orangered1', 'turquoise2'))

# Plot scores - components 1 and 2
plot(pca_2$scores[,1], # Scores on component 1
     pca_2$scores[,2], # Scores on component 3
     pch=16, # Point 16 (colored circle)
     col = colfunc(nrow(pca_2$scores)), # Color points by haul using our color palette
     xlab="1st principal component",ylab="2nd principal component",main="Scores plot") # A

# Add loadings to plot
sf = 3 # Scaling factor
sft = 3.2 # Scaling factor for text
arrows(0,0, # Draw arrows from zero
       pca_2$loadings[,1]*sf, # Draw to PC1 * scaling factor loading in X
       pca_2$loadings[,2]*sf, # Draw to PC2 * scaling factor loading in Y
       col="black", length = 0.1) # Arrow color and arrowhead length
text(pca_2$loadings[,1]*sft,pca_2$loadings[,2]*sft, names(data_log[,-1]), cex=1.0, col="bl

# Generate legend
legend_image <- as.raster(matrix(colfunc(nrow(pca_2$scores)), ncol=1))
plot(c(0,2),c(0,1),type = 'n', axes = F,xlab = '', ylab = '', main = 'Haul')
text(x=1.5, y =seq(0,1,l=5), labels = seq(1,25,l=5))
rasterImage(legend_image, 0, 0, 1,1)
```



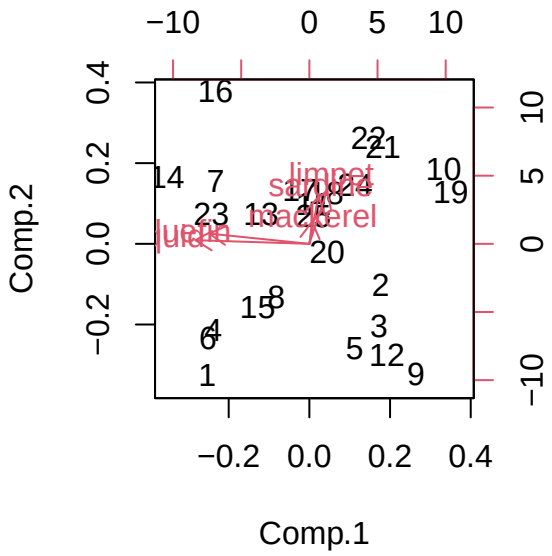
How would you interpret this plot? Does it differ from the covariance plot?

1.3.3 Alternative Methods

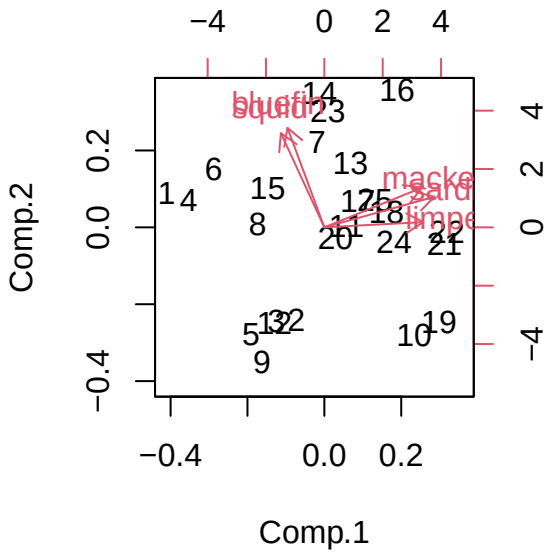
There are a few other ways you can generate, and/or plot your PCAs if you prefer.

Biplot:

```
# Exploring biplot
biplot(pca_1) # Covariance
```

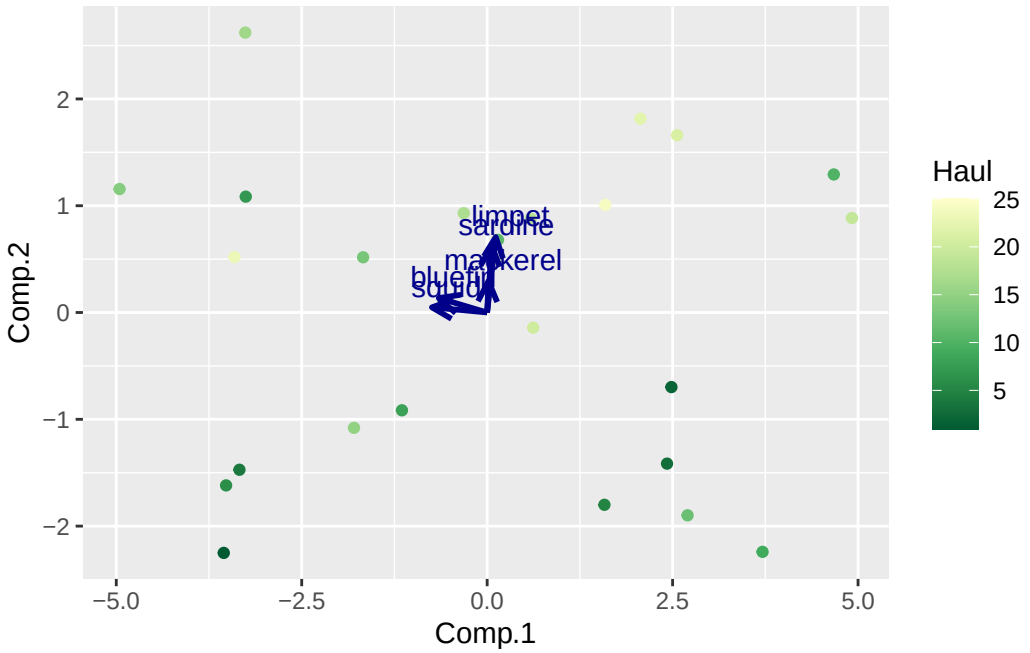



```
biplot(pca_2) # Correlation
```



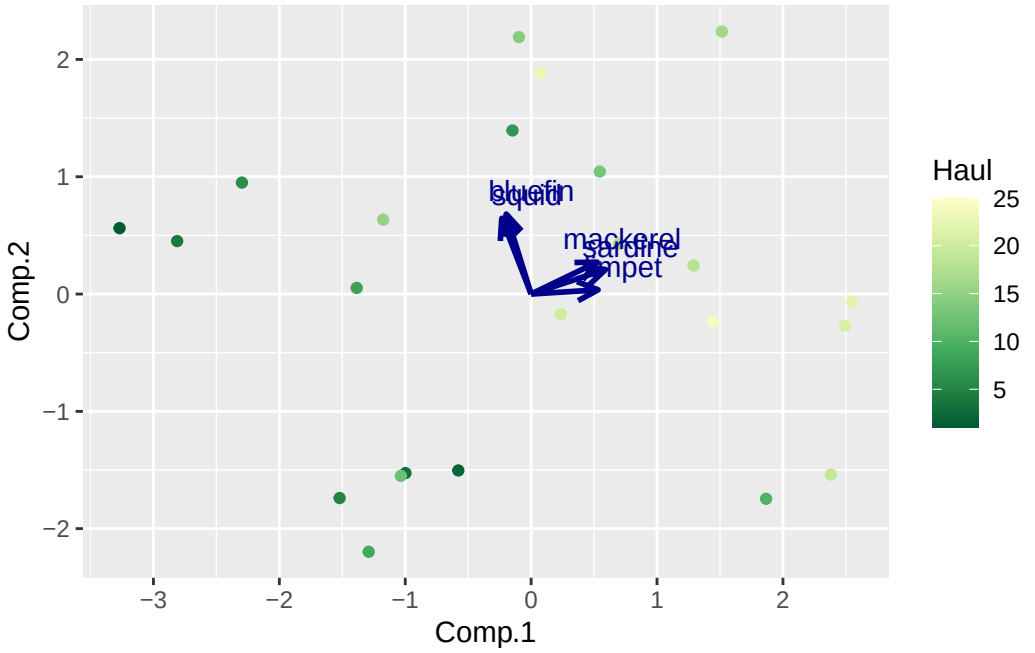
ggplot:

```
library(ggplot2)

# ggplot version - Covariance

# turn PCA scores into data frame
pca_1_plot = data.frame(Haul = data_log$Haul, pca_1$scores)

# Turn PCA loadings into data frame (This gets a little complicated)
pca_1_loadings = as.data.frame(matrix(as.numeric(pca_1$loadings),
                                      dim(pca_1$loadings)[1], dim(pca_1$loadings)[2]))
colnames(pca_1_loadings) = colnames(pca_1_plot)[-1]

# Plot
ggplot(pca_1_plot, aes(x = Comp.1, y = Comp.2, color = Haul)) +

  # Scores
  geom_point() + scale_colour_distiller(palette = 15) +

  # Loadings
  geom_segment(data = pca_1_loadings, aes(x = 0, y = 0,xend = Comp.1 , yend = Comp.2),
    arrow = arrow(length = unit(0.3, "cm"), type = "open", angle = 25),
    linewidth = 1, color = "darkblue") +

  # Labels
  geom_text(data = pca_1_loadings, color = 'darkblue', nudge_x = 0.2, nudge_y = 0.2, # Lab
                aes(x = Comp.1, y = Comp.2, label = colnames(data_log)[-1]))
```



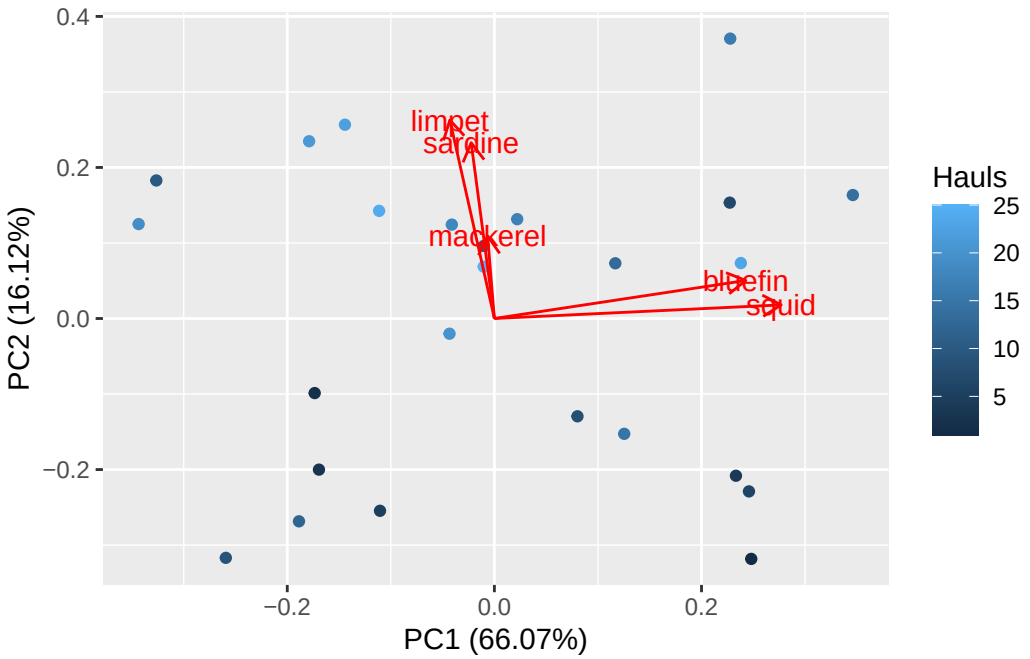

```
# ggplot version - Correlation

# turn PCA scores into data frame
pca_2_plot = data.frame(Haul = data_log$Haul, pca_2$scores)

# Turn PCA loadings into data frame
pca_2_loadings = as.data.frame(matrix(as.numeric(pca_2$loadings),
                                      dim(pca_2$loadings)[1], dim(pca_2$loadings)[2]))
colnames(pca_2_loadings) = colnames(pca_2_plot)[-1]

# Plot
ggplot(pca_2_plot, aes(x = Comp.1, y = Comp.2, color = Haul)) +

  # Scores
  geom_point() + scale_colour_distiller(palette = 15) +

  # Loadings
  geom_segment(data = pca_2_loadings, aes(x = 0, y = 0,xend = Comp.1 , yend = Comp.2),
               arrow = arrow(length = unit(0.3, "cm"), type = "open", angle = 25),
               linewidth = 1, color = "darkblue") +

  # Labels
```

```
  geom_text(data = pca_2_loadings, color = 'darkblue', nudge_x = 0.2, nudge_y = 0.2, # Lab
            aes(x = Comp.1, y = Comp.2, label = colnames(data_log)[-1]))
```

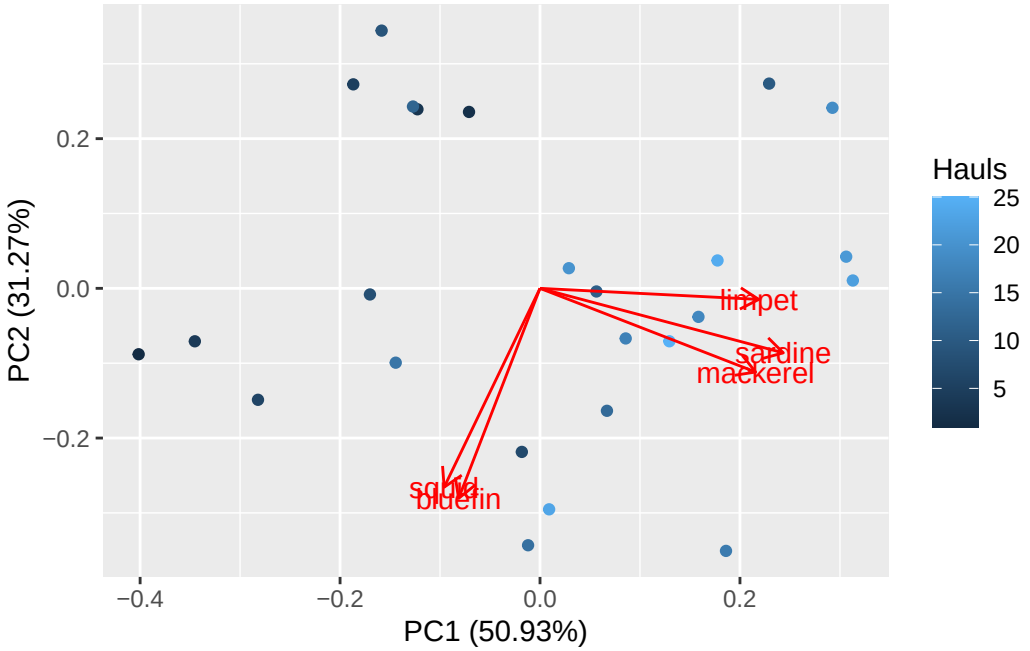


You can also run PCA using the prcomp() function instead of princomp(), setting scale = T if you want the correlation matrix. You can then use autoplot() with the ggfortify package to plot the results.

```
# ggplot v2
library(ggfortify)

# Run PCA - Covariance
pca_1a = prcomp(data_log[,-1])

# Run autoplot
autoplot(pca_1a, data = data_log, color = 'Hauls', loadings = T, loadings.label = T)
```



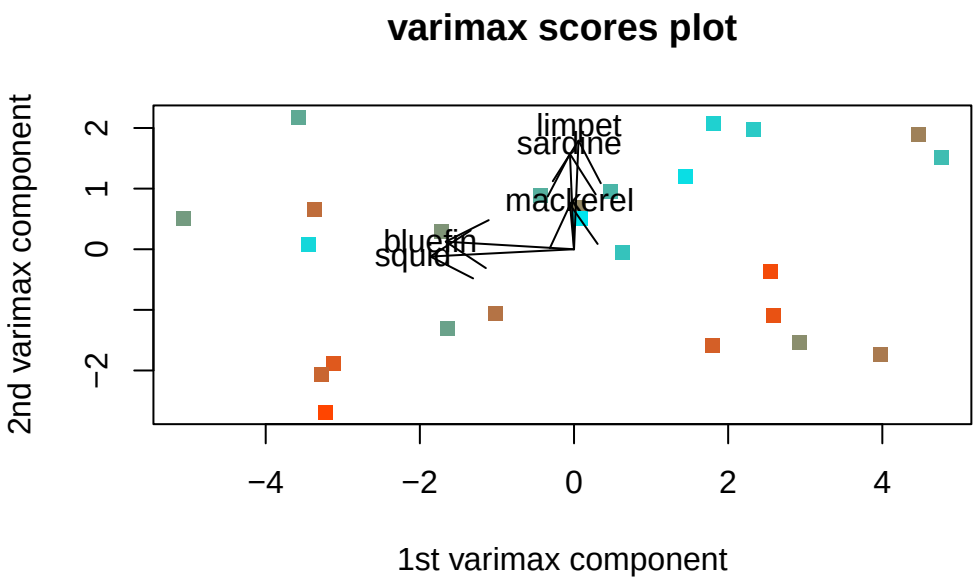

```
# Run PCA - Correlation
pca_2a = prcomp(data_log[,-1], scale = T)

# Run autoplot
autoplot(pca_2a, data = data_log, color = 'Hauls', loadings = T, loadings.label = T)
```



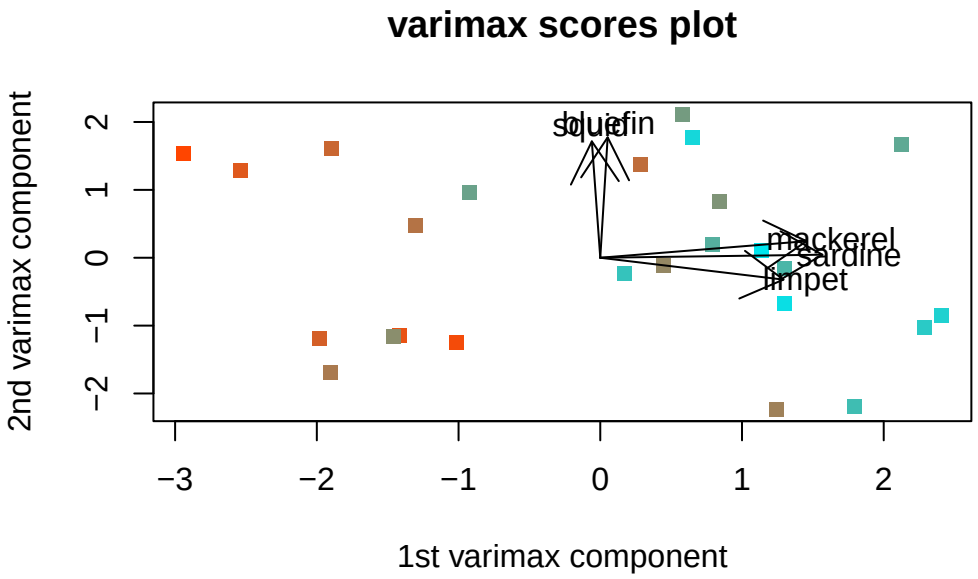
1.4 Varimax Rotation (Optional)

Varimax rotation attempts to improve the interpretability of PCA results by lining up loadings with the axes. This can be useful, particularly with large numbers of variables.

```
# Scaling factors
sf = 2.5
sft = 2.8

# Varimax rotation - Covariance
v1 = varimax(pca_1$loadings[,1:2])
v1_scores = pca_1$scores[,1:2]%*%v1$rotmat

# Plot scores - components 1 and 2
plot(v1_scores[,1],v1_scores[,2],pch=15, col = colfunc(nrow(v1_scores)),
     xlab="1st varimax component",ylab="2nd varimax component",main="varimax scores plot")

# Add loadings
arrows(0,0,v1$loadings[,1]*sf,v1$loadings[,2]*sf,col="black")
text(v1$loadings[,1]*sft,v1$loadings[,2]*sft,names(data_log[,-1]),asp=1,cex=1.0 ,col="blac
```




```
# Varimax rotation - Correlation
v2 = varimax(pca_2$loadings[,1:2])
v2_scores = pca_2$scores[,1:2]%*%v2$rotmat

# Plot scores - components 1 and 2
plot(v2_scores[,1],v2_scores[,2],pch=15, col = colfunc(nrow(v2_scores)),
     xlab="1st varimax component",ylab="2nd varimax component",main="varimax scores plot")

# Add loadings
arrows(0,0,v2$loadings[,1]*sf,v2$loadings[,2]*sf,col="black")
text(v2$loadings[,1]*sft,v2$loadings[,2]*sft,names(data_log[,-1]),asp=1,cex=1.0 ,col="blac
```



Note that it's pretty hard to tell the hauls apart using this color scale. Make sure your plots are always clear and readable.

1.5 Tips for your assignment:

Some things you may want to think about for your assignment:

1. Do your covariance and correlation plots differ? Do you think one is better suited to answering your research question? Why? Is your answer conceptual, or does it have to do with the results? Both?

2. How would you quantitatively examine the effect of haul on the PCA scores above? Is it associated with any of the principal components?

3. How would you interpret your statistical results biologically? You don't have to be right, but don't be vague, and don't contradict your results.

2 Assignment 1b: Linear Discriminant Analysis

Assignment 1b focuses on Linear Discriminant Analysis (LDA), also known as Canonical Variate Analysis. LDA is used to disclose relationships between groups, create models to differentiate between groups based on data, and discern the contribution of different variables to a model's ability do discriminate between groups.

For this tutorial, we'll be using `snake.csv`.

2.1 Looking at the data

```
# Load in data
snake = read.csv('snake.csv')

# Look at data
head(snake)
```

```
  Species   M1   M2   M3   M4   M5   M6
1       A 41.6  6.7  8.2 12.2 24.7 27.0
2       A 40.2  8.5  9.2 15.5 27.1 30.3
3       A 40.4 12.6 14.2 19.6 46.9 26.8
4       A 26.4  9.0  8.6 14.0 37.6 32.2
5       A 34.4  7.0 12.1 11.1 31.0 35.8
6       A 38.8  8.2 10.2 12.4 42.2 33.6
```

```
dim(snake)
```

```
[1] 35  7
```

Our data is a 35 row, 7 column data frame. The first column identifies the species of snake (A or B). The other columns are morphological measurements of each individual snake. We want to know if we can use the morphological measurements of the snakes to determine their species. Let's keep examining the data:

```r
# Make a boxplot
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.4.4     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.0
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

```r
# Convert the data to long format so we can use ggplot
snake_long = pivot_longer(snake, # Enter data
                          colnames(snake)[-1], # Pivot all columns except species
                          names_to = 'Measurement', values_to = 'Value') # Feed labels to

# Lets take a look at the new data frame
head(snake_long)
```

```
# A tibble: 6 x 3
  Species  Measurement Value
  <chr>    <chr>       <dbl>
1 "   A  " M1           41.6
2 "   A  " M2            6.7
3 "   A  " M3            8.2
4 "   A  " M4           12.2
5 "   A  " M5           24.7
6 "   A  " M6           27
```

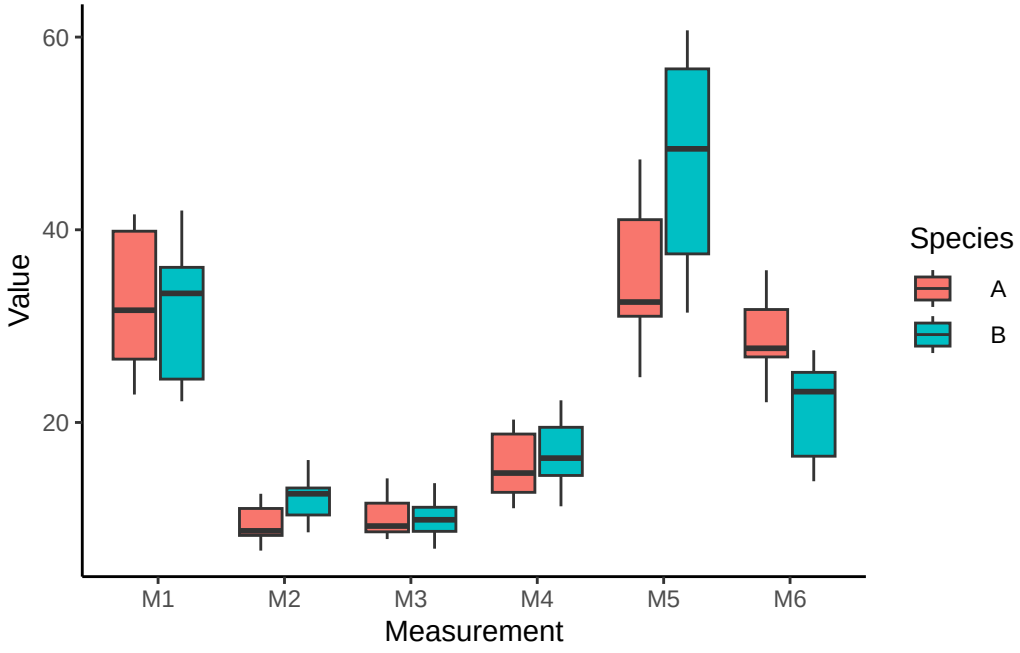
```r
# We've converted from wide format to long format,
# now all the data values are contained in a single column
# which is described by a metadata column

# You can also do this with melt from reshape2
library(reshape2)
```

```
Attaching package: 'reshape2'

The following object is masked from 'package:tidyr':

    smiths
```

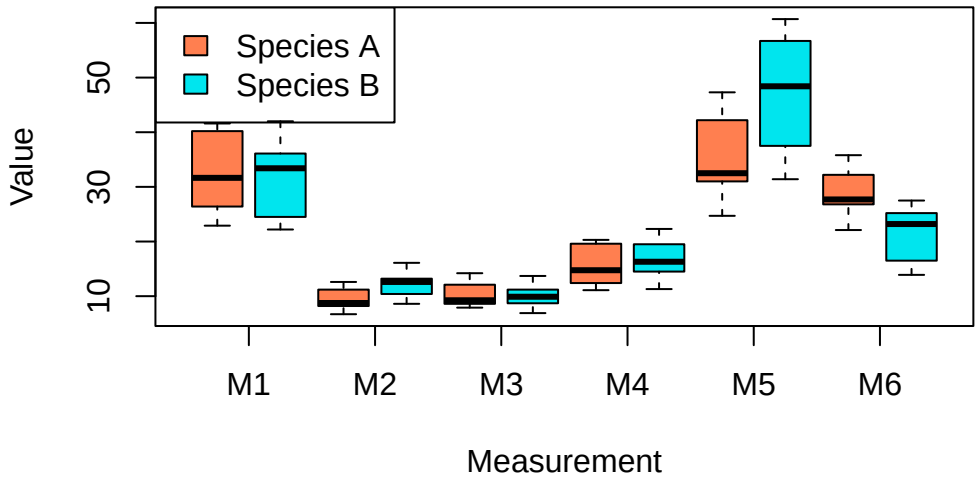
```
head(melt(snake))
```

```
Using Species as id variables

  Species variable value
1       A       M1  41.6
2       A       M1  40.2
3       A       M1  40.4
4       A       M1  26.4
5       A       M1  34.4
6       A       M1  38.8
```

```
# Let's make a boxplot
ggplot(snake_long, aes(x = Measurement, y = Value, fill = Species)) +
  geom_boxplot() + theme_classic()
```



```
# We can do this in R base plot too
boxplot(Value ~ Species*Measurement, # Plot value by species and measurement
        data = snake_long, col = c('coral', 'turquoise2'), # Color by species
        xaxt = 'n', xlab = 'Measurement') # Remove and label x axis
legend('topleft', legend = c('Species A', 'Species B'), fill = c('coral', 'turquoise2')) #
axis(1, at = seq(1.5,11.5,2), labels = colnames(snake)[-1]) # Add x axis back in with appr
```



Some of our measurements are very similar across species, and others are quite different. Do they differ statistically as a whole?

2.2 MANOVA

The purpose of LDA is to try to discriminate our snakes into species based on their measurements. However, that only makes sense to do if our two species of snake actually differ across the measurements. Our first step then is to discern whether our snake species differ as a multivariate whole. We'll do this using a MANOVA.

```
# Run MANOVA
sm = manova(cbind(M1,M2,M3,M4,M5,M6) ~ Species, data = snake)
summary(sm, test = 'Hotelling')
```

```
          Df Hotelling-Lawley approx F num Df den Df   Pr(>F)
Species    1           1.2263   5.7229      6     28 0.000552 ***
Residuals 33
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(sm, test = 'Wilks')
```

```
          Df   Wilks approx F num Df den Df   Pr(>F)
Species    1 0.44917   5.7229      6     28 0.000552 ***
Residuals 33
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

By both the Hotelling's and Wilks' tests, our MANOVA is significant, indicating the snake species vary as a multivariate whole.

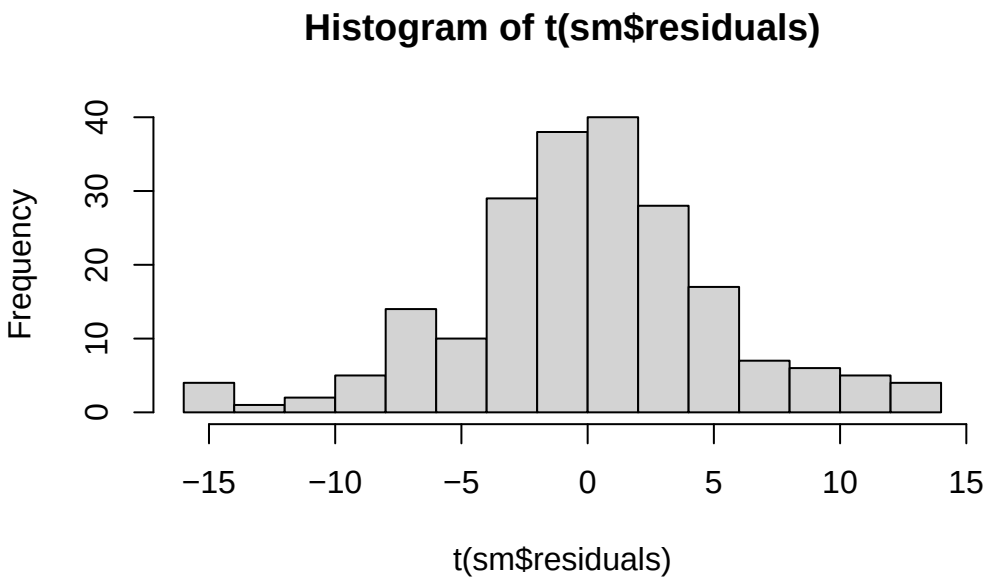
What about our assumptions though? Our MANOVA assumptions are normality, linearity, and homogeneity of covariances. You've been told to assume the latter, so let's skip that one.

```
# Testing normality
library(mvnormtest)
mshapiro.test(t(sm$residuals))
```

```
    Shapiro-Wilk normality test

data:  Z
W = 0.91571, p-value = 0.01075
```

Uh oh, the residuals are significantly non-normal. Let's take a look at them visually:

```
# Residual histogram
hist(t(sm$residuals), breaks = 20)
```



Visually, our residuals actually look quite close to normal. There may be some slight skew, or outliers that are forcing our residuals to statistical non-normality. We might be able to fix this by removing multivariate outliers, or by transforming some of our data (feel free to play around with these ideas!), but based on the shape of our residuals, it is unlikely that our model is fatally biased, and we may end up doing more harm than good. Based on this, we can conclude that our two species have significantly different morphometries given the measurements provided.

2.3 Linear Discriminant Analysis

Now that we've confirmed our species differ as a multivariate whole, we can try to use LDA to build a model to predict which species each snake belongs to based on its measurements.

```
# LDA
library(MASS)
```

```
Attaching package: 'MASS'
```

```
The following object is masked from 'package:dplyr':
```

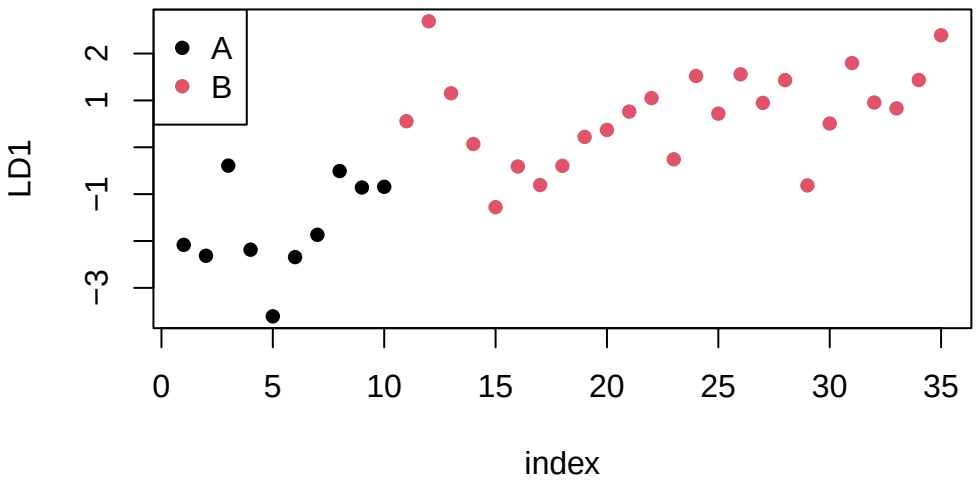
select

```r
ldaf1 <- lda(Species ~ M1+M2+M3+M4+M5+M6, snake)
ldaf1
```

```
Call:
lda(Species ~ M1 + M2 + M3 + M4 + M5 + M6, data = snake)

Prior probabilities of groups:
      A         B
0.2857143 0.7142857

Group means:
           M1     M2    M3    M4     M5     M6
   A   32.700  9.410 10.16 15.54 35.290 28.950
   B   31.496 12.128 10.18 16.78 47.356 21.752

Coefficients of linear discriminants:
           LD1
M1  0.01428023
M2  0.29104494
M3 -0.07327616
M4 -0.05544769
M5  0.03629586
M6 -0.17208517
```

Running our LDA object tells us the prior probabilities used for each species (the proportion of each species in the data), the group means for each measure on each species, and the linear discriminant (LD1) for each measure. We can then plot the LD1 value for each individual:

```r
# Plot discriminant function analysis

# Create a data frame to plot
ldaf_plot = cbind(snake, # Data
                  predict(ldaf1)$x, # LD1 value for each individual given its measurements
                  index = seq(1,nrow(snake), 1)) # Row/Individual number

# Plot
plot(LD1 ~ index, data = ldaf_plot, col = as.factor(snake$Species), pch = 16)
legend('topleft', legend = c('A', 'B'), col = c(1, 2), pch = 16) # Add legend
```



Here we can see higher LD1 values are associated with species B, while lower LD1 values are associated with species A. This is just based on model fit however; how do we know we aren't overfitting? One way to avoid overfitting is by jackknifing (AKA leave-one-out cross validation in this context). This method runs the model once without each point in the dataset, then calculates the posterior probability that the left out point belongs to each species. Let's try it out:

```
# LDA 2, CV = T
ldaf2 = lda(Species ~ M1+M2+M3+M4+M5+M6, snake, CV = T)

# Gather posteriors
as.data.frame(cbind(ldaf2$posterior, # Pull posteriors from ldaf2
                    ResultantSpp=as.character(ldaf2$class))) # Pull predicted species (i.e
```

```
                    A                     B   ResultantSpp
1     0.897801237948675     0.102198762051325          A
2     0.957033498274347    0.0429665017256531          A
3   0.00486396795570833      0.995136032044292          B
4     0.939579607872302    0.0604203921276979          A
5     0.999020574119129 0.000979425880871094          A
6     0.958283942083953     0.041716057916047          A
7     0.859914694048175     0.140085305951825          A
```

```
8    0.0790276689479006      0.9209723310521          B
9     0.250711809994116    0.749288190005884          B
10    0.277233534989757    0.722766465010243          B
11   0.0654339037846645    0.934566096215335          B
12 8.13045175684633e-05    0.999918695482432          B
13  0.00857331675606214    0.991426683243938          B
14    0.119793120831736    0.880206879168264          B
15    0.868897347918874    0.131102652081126          A
16    0.291404395123413    0.708595604876587          B
17    0.580893601645515    0.419106398354485          A
18    0.407526292222816    0.592473707777184          B
19    0.097139347240766    0.902860652759234          B
20   0.0629455676122022    0.937054432387798          B
21   0.0262176442553781    0.973782355744622          B
22   0.0110464412594654    0.988953558740534          B
23     0.37967670676917     0.62032329323083          B
24  0.00300786068222619    0.996992139317774          B
25   0.0331152011340242    0.966884798865976          B
26  0.00270158005931187    0.997298419940688          B
27   0.0161164609849135    0.983883539015087          B
28  0.00346528534198866    0.996534714658011          B
29    0.761253716426843    0.238746283573157          A
30   0.0597294571669348    0.940270542833065          B
31  0.00139900114299064    0.998600998857009          B
32   0.0146304515487079    0.985369548451292          B
33   0.0215114427320867    0.978488557267913          B
34  0.00359029891803412    0.996409701081966          B
35 8.92739861715428e-05    0.999910726013828          B
```

How does this differ from the predictions from our first model?

```r
# Pull ldaf1 model predictions
ldaf_pred = predict(ldaf1)$class

# Gather Predictions
ldaf_diff = data.frame(ldaf1 = as.character(ldaf_pred), ldaf2 = as.character(ldaf2$class))

# Add match column
ldaf_diff$match = (ldaf_diff$ldaf1 == ldaf_diff$ldaf2)

# Which ones are different?
ldaf_diff[which(ldaf_diff$match == F),]
```

```
    ldaf1  ldaf2 match
17    B      A   FALSE
29    B      A   FALSE
```

Individuals 17 and 29 both differed in species prediction between the model fit and the jackknife posterior probability. Now let's check the accuracy of our model fit:

```
# Calculate error
ldaf_wrong = length(which(ldaf_pred != snake$Species)) # Number of incorrect predictions
ldaf_err = ldaf_wrong/nrow(snake) # Divide by number of individuals for error

# Print error
ldaf_wrong
```

```
[1] 5
```

```
ldaf_err
```

```
[1] 0.1428571
```

Our model classified 5 out of 35 (~14.3%) of the snakes as the incorrect species, meaning 30/35 were correct (~85.7%). Not bad, but can we do better?

2.4 Model Selection

Our previous model used all 6 measurements, but do we really need all of them, or are some of them unhelpful (or even detrimental)? To test this, we can run model selection using the `stepclass()` function:

```
# stepclass package
library(klaR)

# Model selection (forward)
ms_f = stepclass(Species ~ M1+M2+M3+M4+M5+M6,data=snake,
          method="lda", fold=35, direction="forward")
```

```
`stepwise classification', using 35-fold cross-validated correctness rate of method lda'.
```

```
35 observations of 6 variables in 2 classes; direction: forward

stop criterion: improvement less than 5%.

correctness rate: 0.85714;  in: "M6";  variables (1): M6

 hr.elapsed min.elapsed sec.elapsed
       0.00        0.00        0.77
```

```r
# Print model selection result
ms_f
```

```
method      : lda
final model : Species ~ M6
<environment: 0x00000226bfe55160>

correctness rate = 0.8571
```

After model selection, we end up with a model using only M6 to predict species, with a correctness rate of 85.7%. This model has the same correctness as the full model, using only one measurement. In other words, this model is more **efficient** - it gets to the same accuracy using less information.

This model was generated using forward model selection, meaning the selection process works exclusively by adding variables to the model. We can also do the opposite:

```r
# stepclass package
library(klaR)

# Model selection (forward)
ms_b = stepclass(Species ~ M1+M2+M3+M4+M5+M6,data=snake,
          method="lda", fold=35, direction="backward")
```

```
 `stepwise classification', using 35-fold cross-validated correctness rate of method lda'.

35 observations of 6 variables in 2 classes; direction: backward

stop criterion: improvement less than 5%.
```

```
correctness rate: 0.8;  starting variables (6): M1, M2, M3, M4, M5, M6
correctness rate: 0.85714;  out: "M5";  variables (5): M1, M2, M3, M4, M6

 hr.elapsed min.elapsed sec.elapsed
       0.00        0.00        0.87
```

```
# Print model selection result
ms_b
```

```
method      : lda
final model : Species ~ M1 + M2 + M3 + M4 + M6
<environment: 0x00000226c03d00b0>

correctness rate = 0.8571
```

Backwards model selection works by removing variables from the full model. This means backwards selection should always return a model with a equal or more variables than forwards selection.

Lastly, we can run both:

```
# stepclass package
library(klaR)

# Model selection (forward)
ms_d = stepclass(Species ~ M1+M2+M3+M4+M5+M6,data=snake,
          method="lda", fold=35, direction="both")
```

```
 `stepwise classification', using 35-fold cross-validated correctness rate of method lda'.

35 observations of 6 variables in 2 classes; direction: both

stop criterion: improvement less than 5%.

correctness rate: 0.85714;  in: "M6";  variables (1): M6

 hr.elapsed min.elapsed sec.elapsed
       0.00        0.00        0.66
```

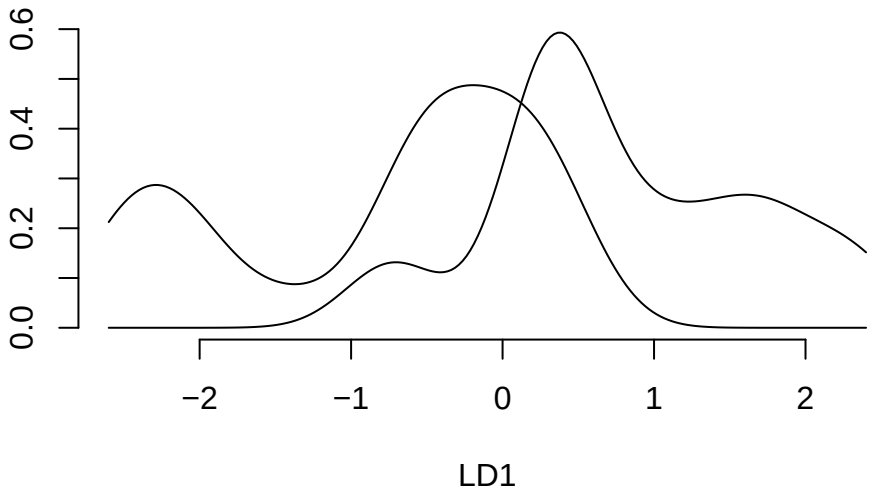
```r
# Print model selection result
ms_d
```

```
method      : lda
final model : Species ~ M6
<environment: 0x00000226c0c0d000>

correctness rate = 0.8571
```

2.5 Plotting Probabilities

Lets finish off by making some plots to visualize our LDA model results.

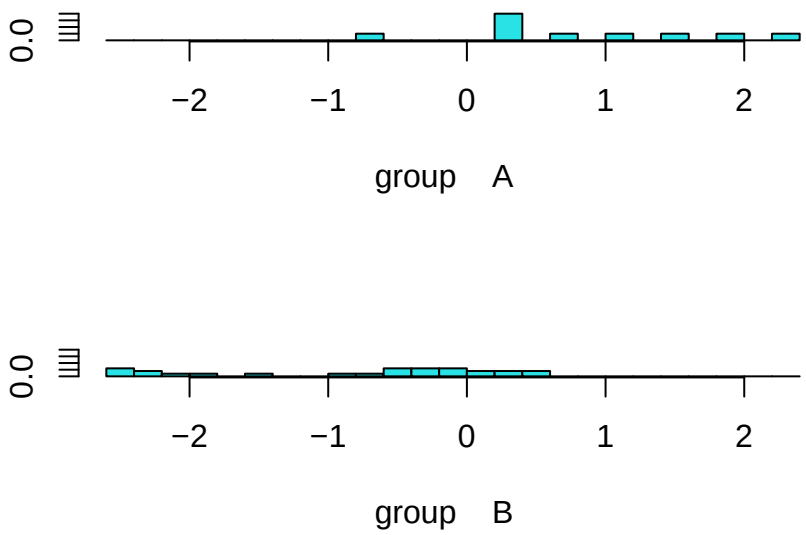
```r
# Pick a model to plot
ldaf3 = lda(Species ~ M6, data = snake)

# Plot density curve
plot(ldaf3, dimen = 1, type = 'dens')
```



This plots the posterior probabilites of an individual belonging to either species given its LD1 value. Remember from earlier that species A is associated with lower LD1 values.

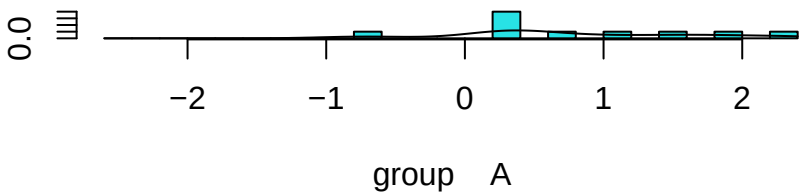
We can also make this plot as a histogram:

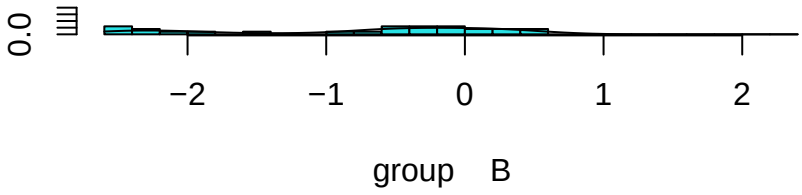
```
# Plot density curve
par(mar = c (4,4,4,4))
plot(ldaf3, dimen = 1, type = 'hist')
```



Or combine both plots:

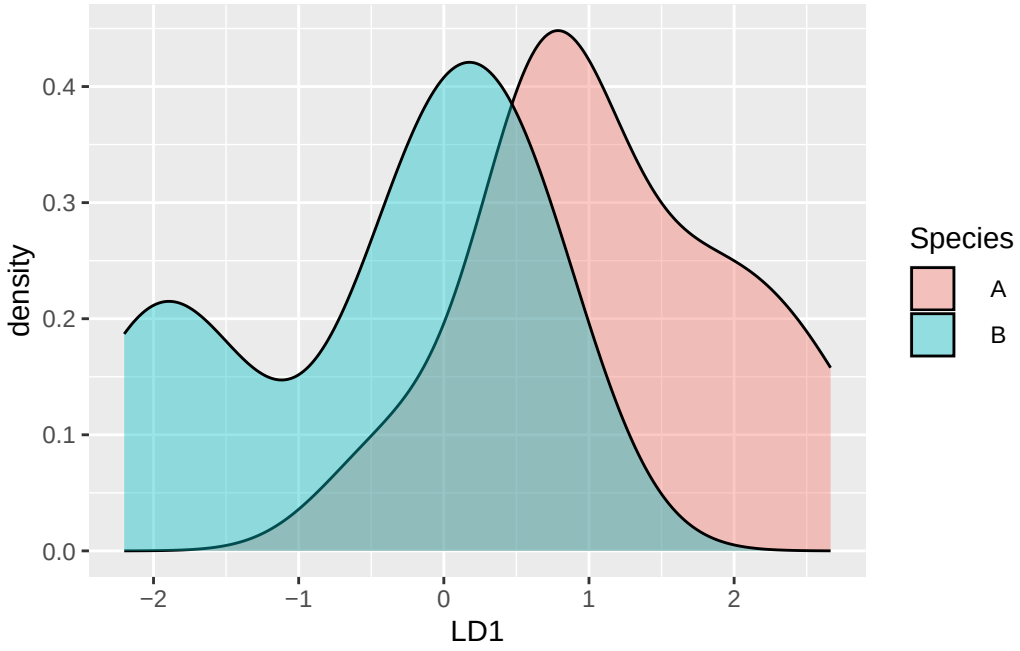
```
# Plot density curve
par(mar = c (4,4,4,4))
plot(ldaf3, dimen = 1, type = 'both')
```





As always, we can also do this with ggplot too:

```
# Predict species
ldaf3_pred = predict(ldaf3)

# Plot
pred_species = as.data.frame(ldaf3_pred$x) # Gather LD1 values
pred_species$Species = snake$Species # Gather true species from data

# Plot
ggplot(pred_species, aes(x = LD1, fill = Species))+
  geom_density(alpha = 0.4)# alpha tells you how transparent the plots will be
```



2.6 Tips for your assignment

Some things you may want to think about for your assignment:

1. How would you pick which model you think is best? What factors would you consider? Are there any factors you would consider other than those discussed in this tutorial?

2. LDA also assumes the data are independent. Do we know this assumption is respected? Why or why not? What would constitute it not being respected?

3. How would you interpret your statistical results biologically (can be in terms of the snakes, how you would study them, or both)? You don't have to be right, but don't be vague, and don't contradict your results.

3 Assignment 1c: Cluster Analysis and Multidimensional Scaling

This assignment is centered on cluster analysis and multidimensional scaling (MDS), which are both methods of measuring associations within a group (e.g. associations between individuals within a population).

For this tutorial, we'll be using `monkey.csv`.

3.1 Looking at the data

You know the drill by now:

```
# Load in data
data = read.csv('monkey.csv', row.names = 1) # First column is row names
data # Print data
```

```
      ind1 ind2 ind3 ind4 ind5 ind6 ind7 ind8 ind9 ind10 ind11 ind12 ind13
ind1    21    2    2   10    2    2    8    0    0     8    14    12     4
ind2     2   21   16    2   16    8    2    2    4     4     4     0     2
ind3     2   16   21    0   10   16    2    0    2     4     4     0     2
ind4    10    2    0   21    2    2   16    2    2     8    12     8     4
ind5     2   16   10    2   21   10    2    4    0     2     4     4     2
ind6     2    8   16    2   10   21    4    2    0     0     0     4     4
ind7     8    2    2   16    2    4   21    4    2    16     8     8     4
ind8     0    2    0    2    4    2    4   21    0     2     0     0     0
ind9     0    4    2    2    0    0    2    0   21     0     4     0     0
ind10    8    4    4    8    2    0   16    2    0    21    14    14     2
ind11   14    4    4   12    4    0    8    0    4    14    21    12     4
ind12   12    0    0    8    4    4    8    0    0    14    12    21     2
ind13    4    2    2    4    2    4    4    0    0     2     4     2    21
```

Our data is a matrix containing the number of social interactions observed between individuals in a group of monkeys at the zoo. The matrix is symmetrical - the top/right half is identical to the bottom/left half.

3.2 Calculating Dissimilarity

For this assignment we'll be using 3 R functions: `hclust`, `metaMDS` (from the `vegan` package), `isoMDS` (from the `MASS`package), and `cmdscale()`. Let's see what type of input data those functions need:

```
# Check help functions
library(vegan)
library(MASS)
?hclust()
?metaMDS()
?isoMDS()
?cmdscale()
```

You'll notice all of these functions require a **dissimilarity matrix** produced by `dist`. Let's start by running `dist()`.

```
# Convert data to a dist object
dist = as.dist(data)
dist # Print dist
```

```
      ind1 ind2 ind3 ind4 ind5 ind6 ind7 ind8 ind9 ind10 ind11 ind12
ind2     2
ind3     2   16
ind4    10    2    0
ind5     2   16   10    2
ind6     2    8   16    2   10
ind7     8    2    2   16    2    4
ind8     0    2    0    2    4    2    4
ind9     0    4    2    2    0    0    2    0
ind10    8    4    4    8    2    0   16    2    0
ind11   14    4    4   12    4    0    8    0    4    14
ind12   12    0    0    8    4    4    8    0    0    14    12
ind13    4    2    2    4    2    4    4    0    0     2     4     2
```

Now our data is in a `dist` object. All of the redundant entries in the data have been removed.

Right now, our data reflects **similarity (i.e. high numbers reflect greater association between individuals)**. We need to convert it to **dissimilarity**. Dissimilarity is simply the opposite of similarity. We can convert similarity to dissimilarity by subtracting each data value from the maximum of the data.

```r
# Convert to dissimilarity
dist = max(dist) - dist
dist # Print dist
```

```
      ind1 ind2 ind3 ind4 ind5 ind6 ind7 ind8 ind9 ind10 ind11 ind12
ind2    14
ind3    14    0
ind4     6   14   16
ind5    14    0    6   14
ind6    14    8    0   14    6
ind7     8   14   14    0   14   12
ind8    16   14   16   14   12   14   12
ind9    16   12   14   14   16   16   14   16
ind10    8   12   12    8   14   16    0   14   16
ind11    2   12   12    4   12   16    8   16   12     2
ind12    4   16   16    8   12   12    8   16   16     2     4
ind13   12   14   14   12   14   12   12   16   16    14    12    14
```

Now we're ready to run our analyses!

3.3 Hierarchical Cluster Analysis

Remember from lecture there are 4 types of hierarchical cluster analysis:

1. Single linkage
2. Average linkage
3. Complete linkage
4. Ward linkage

Let's run through them one by one:

3.3.1 Single linkage

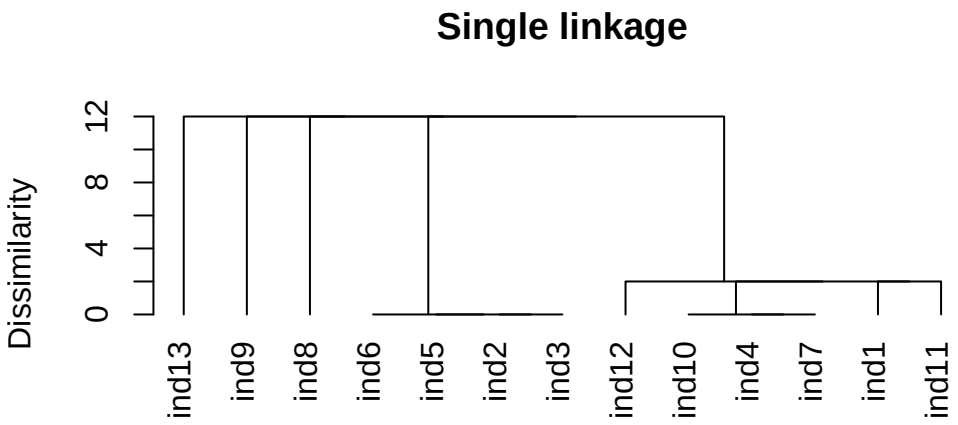
We can run all 4 types of cluster analysis using the `hclust()` R function:

```r
# run single linkage cluster analysis
clust_1 = hclust(dist, method = 'single')
clust_1 # print object
```

```
Call:
hclust(d = dist, method = "single")

Cluster method   : single
Number of objects: 13
```

Printing the `hclust` object doesn't really tell us much. For more detail, we're going to have to plot it:

```
# Plot single linkage tree
plot(clust_1, hang = -1, main = 'Single linkage',
     ylab = 'Dissimilarity', # Label y axis
     xlab = '', sub = '') # Remove x-axis label
```



This outputs a tree showing the associations between our individual monkeys. dissimilarity is on the y-axis. The greater the distance between individuals on the y-axis, the greater their dissimilarity. Our tree has grouped the monkeys according to how frequently they interact with each other. For example. individuals 2, 3, 5, and 6 interact often, as evidenced by their low dissimilarity.

But how well does this tree fit the data? To answer that question, we need to calculate the cophenetic correlation coefficient (CCC):

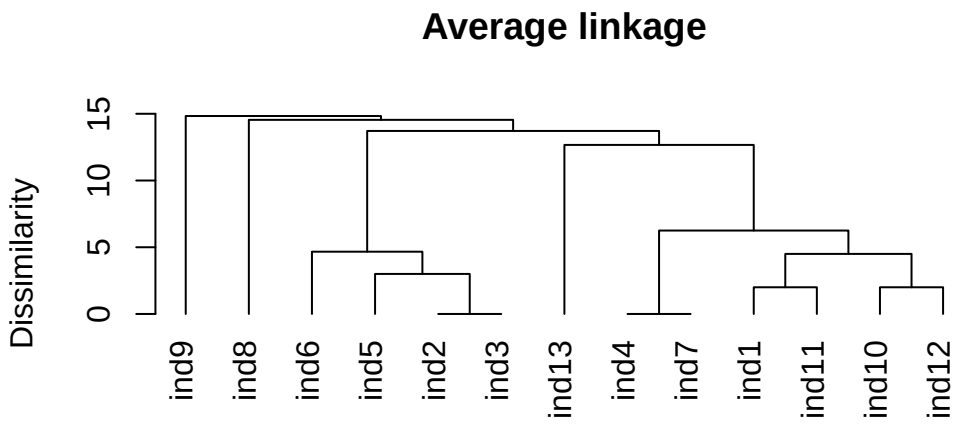
```
# Calculate CCC
coph_1 = cophenetic(clust_1) # Get cophenetic
ccc_1 = cor(coph_1, dist) # Calculate correlation of the cophenetic with the data
ccc_1 # Print CCC
```

```
[1] 0.9036043
```

That's a pretty high correlation coefficient, indicating our dendrogram represented the structure in the original data very well. Let's try some other methods:

3.3.2 Average Linkage

```
# run cluster analysis
clust_2 = hclust(dist, method = 'average')

# Plot
plot(clust_2, hang = -1, main = 'Average linkage', ylab = 'Dissimilarity', xlab = '', sub
```

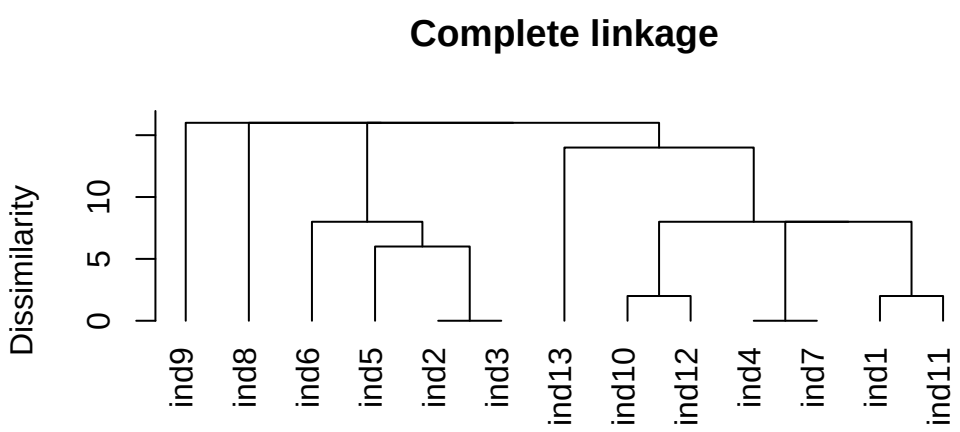


```
# Calculate CCC
coph_2 = cophenetic(clust_2)
ccc_2 = cor(coph_2, dist)
ccc_2
```

```
[1] 0.9288949
```

3.3.3 Complete Linkage

```
# run cluster analysis
clust_3 = hclust(dist, method = 'complete')

# Plot
plot(clust_3, hang = -1, main = 'Complete linkage', ylab = 'Dissimilarity', xlab = '', sub
```

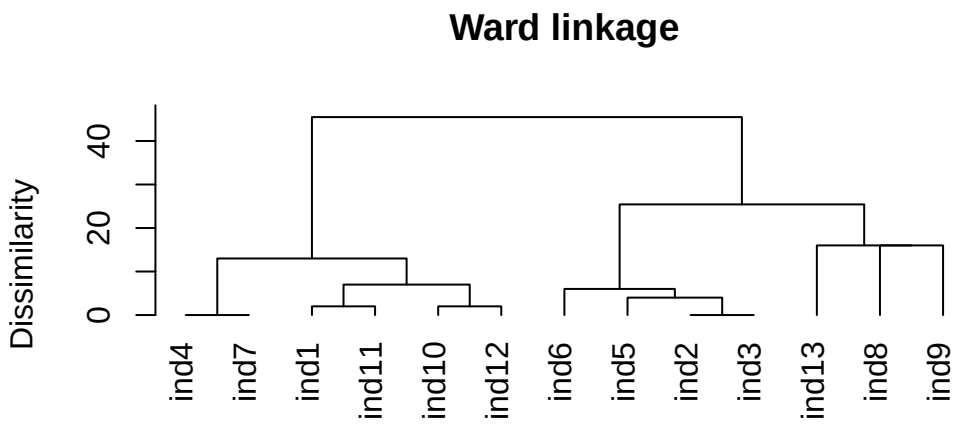



```
# Calculate CCC
coph_3 = cophenetic(clust_3)
ccc_3 = cor(coph_3, dist)
ccc_3
```

```
[1] 0.9141956
```

3.3.4 Ward Linkage

```
# run cluster analysis
clust_4 = hclust(dist, method = 'ward.D')

# Plot
plot(clust_4, hang = -1, main = 'Ward linkage', ylab = 'Dissimilarity', xlab = '', sub = '
```




```
# Calculate CCC
coph_4 = cophenetic(clust_4)
ccc_4 = cor(coph_4, dist)
ccc_4
```

```
[1] 0.7633159
```

Each method gives a slightly different tree and CCC value. Where are they similar? Where do they differ? Which one(s) would you trust? Why?

3.4 Multidimensional Scaling

Another method we can use to test for associations between our monkeys is multidimensional scaling (MDS). There are two types of MDS: non-metric, and metric MDS. Let's start with non-metric MDS.

3.4.1 Non-Metric MDS

```
# Run non-metric MDS - metaMDS
mds1 = metaMDS(dist, wascores = F)
```

```
Run 0 stress 0.07592385
Run 1 stress 0.07875788
Run 2 stress 0.08055013
Run 3 stress 0.07366297
... New best solution
... Procrustes: rmse 0.1937845  max resid 0.5647286
Run 4 stress 0.08055013
Run 5 stress 0.072393
... New best solution
... Procrustes: rmse 0.1707802  max resid 0.5155166
Run 6 stress 0.07239299
... New best solution
... Procrustes: rmse 2.01727e-05  max resid 4.354993e-05
... Similar to previous best
Run 7 stress 0.07875788
Run 8 stress 0.072393
... Procrustes: rmse 2.216451e-05  max resid 4.829459e-05
... Similar to previous best
Run 9 stress 0.07575137
Run 10 stress 0.08055013
Run 11 stress 0.08575957
Run 12 stress 0.2169828
Run 13 stress 0.1428286
Run 14 stress 0.1810485
Run 15 stress 0.07351183
Run 16 stress 0.07875788
Run 17 stress 0.07239302
... Procrustes: rmse 8.399926e-05  max resid 0.0001842553
... Similar to previous best
Run 18 stress 0.08055013
```

```
Run 19 stress 0.07358653
Run 20 stress 0.08569059
*** Best solution repeated 3 times
```

```
# Print mds results
mds1
```

```
Call:
metaMDS(comm = dist, wascores = F)

global Multidimensional Scaling using monoMDS

Data:     dist
Distance: user supplied

Dimensions: 2
Stress:     0.07239299
Stress type 1, weak ties
Best solution was repeated 3 times in 20 tries
The best solution was from try 6 (random start)
Scaling: centring, PC rotation
Species: scores missing
```

By default, metaMDS has two dimensions. This MDS has a stress value of 0.072. Remember from lecture that stress < 0.10 is a "good representation", so this MDS result is pretty good. If we want, we can test different numbers of dimensions (k) and create a scree plot to find the best one:

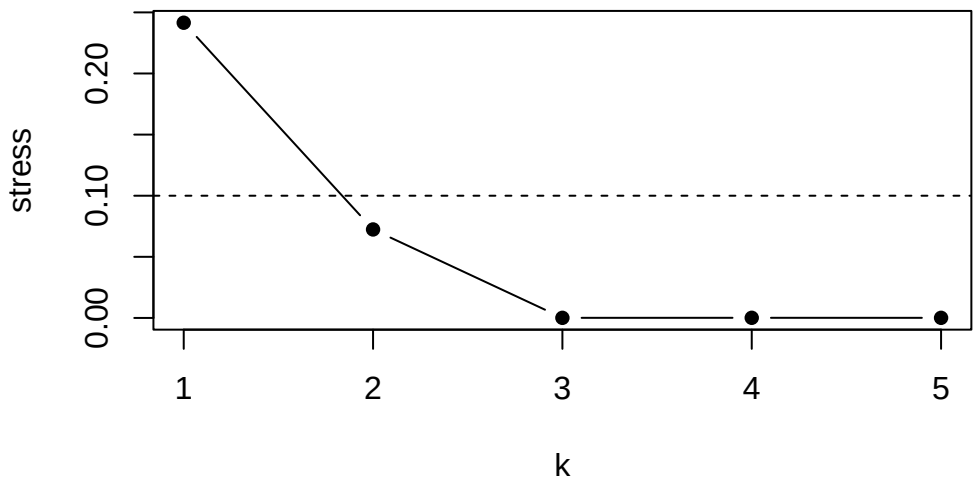
```
# Create a container object
scree = data.frame(k = 1:5, stress = NA)

# Loop through k 1 to 5
for(k in 1:5){

  # Run MDS
  mds = metaMDS(dist, wascores = F, k = k) # Set k to our loop index

  # Pull out stress
  scree[k,'stress'] = mds$stress # Fill kth row of the column 'stress' in scree
```

```
} # End loop
```

```
# Print results
scree
```

```
  k       stress
1 1 2.415380e-01
2 2 7.239299e-02
3 3 8.193178e-05
4 4 1.140069e-04
5 5 9.642494e-05
```

```
# Make scree plot
plot(stress ~ k, data = scree, # Plot stress against k
     type = 'b', # Lines and points
     pch = 16) # Point 16 (filled circle)
abline(h = 0.1, lty = 'dashed') # Plot a dashed line at 0.1
```

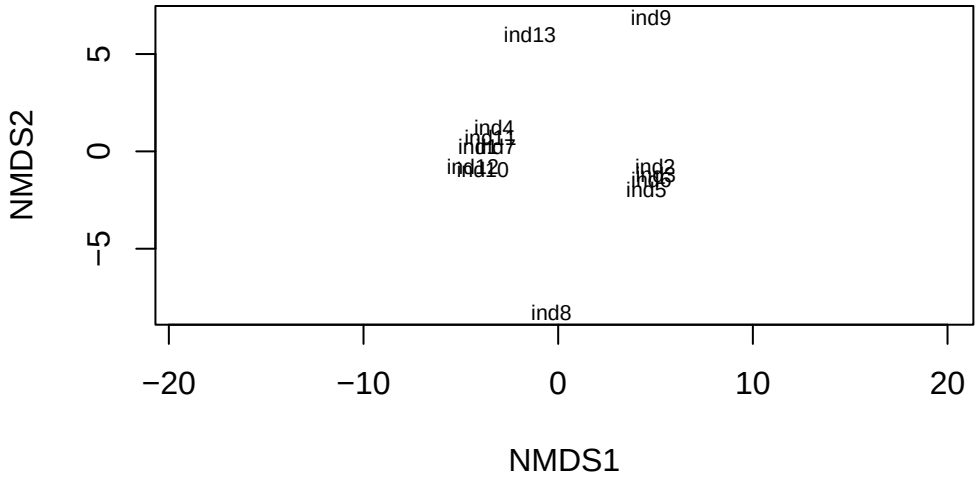


We have an elbow at k=3, but we also get warnings that our dataset may be too small using k=3. The stress at k=2 is low enough that we can stick to using that.

Let's plot our results:

```
# Plot result
plot(mds1, type = 't')
```

```
species scores not available
```



Here we've plotted the values of our two MDS dimensions against each other for each individual. Similar to the cluster analysis, we see certain individuals are grouped together. Is it the same groups of individuals? What does that tell you about your results?

Let's try a different non-metric MDS function:

```
# Run non-metric MDS - isoMDS
mds2 = isoMDS(dist)
```

```
Error in isoMDS(dist): zero or negative distance between objects 2 and 3
```

Uh oh. This function doesn't like zeroes in the data. Let's fix that by translating our data to proportions, and adding a small increment.

```r
# Translate to proportions
dist2 = dist/max(dist)

# Add an increment
dist2 = dist2 + 0.0001

# Print new dist
dist2
```

```
        ind1   ind2   ind3   ind4   ind5   ind6   ind7   ind8   ind9  ind10
ind2  0.8751
ind3  0.8751 0.0001
ind4  0.3751 0.8751 1.0001
ind5  0.8751 0.0001 0.3751 0.8751
ind6  0.8751 0.5001 0.0001 0.8751 0.3751
ind7  0.5001 0.8751 0.8751 0.0001 0.8751 0.7501
ind8  1.0001 0.8751 1.0001 0.8751 0.7501 0.8751 0.7501
ind9  1.0001 0.7501 0.8751 0.8751 1.0001 1.0001 0.8751 1.0001
ind10 0.5001 0.7501 0.7501 0.5001 0.8751 1.0001 0.0001 0.8751 1.0001
ind11 0.1251 0.7501 0.7501 0.2501 0.7501 1.0001 0.5001 1.0001 0.7501 0.1251
ind12 0.2501 1.0001 1.0001 0.5001 0.7501 0.7501 0.5001 1.0001 1.0001 0.1251
ind13 0.7501 0.8751 0.8751 0.7501 0.8751 0.7501 0.7501 1.0001 1.0001 0.8751
       ind11  ind12
ind2
ind3
ind4
ind5
ind6
ind7
ind8
ind9
ind10
ind11
ind12 0.2501
ind13 0.7501 0.8751
```

Let's make sure this doesn't mess with our results:

```r
# Run non-metric MDS - metaMDS
mds1 = metaMDS(dist2, wascores = F)
```

```
Run 0 stress 0.07575137
Run 1 stress 0.07239305
... New best solution
... Procrustes: rmse 0.2253637  max resid 0.6639914
Run 2 stress 0.1832299
Run 3 stress 0.07366297
Run 4 stress 0.07875788
Run 5 stress 0.07239301
... New best solution
... Procrustes: rmse 0.0001681742  max resid 0.0003613014
... Similar to previous best
Run 6 stress 0.0757299
Run 7 stress 0.1851728
Run 8 stress 0.07227846
... New best solution
... Procrustes: rmse 0.01249024  max resid 0.03087978
Run 9 stress 0.08575957
Run 10 stress 0.08055013
Run 11 stress 0.07875788
Run 12 stress 0.07875788
Run 13 stress 0.1183846
Run 14 stress 0.07358653
Run 15 stress 0.07358653
Run 16 stress 0.07875788
Run 17 stress 0.07875788
Run 18 stress 0.07875788
Run 19 stress 0.1193093
Run 20 stress 0.0757299
*** Best solution was not repeated -- monoMDS stopping criteria:
    14: stress ratio > sratmax
     6: scale factor of the gradient < sfgrmin
```

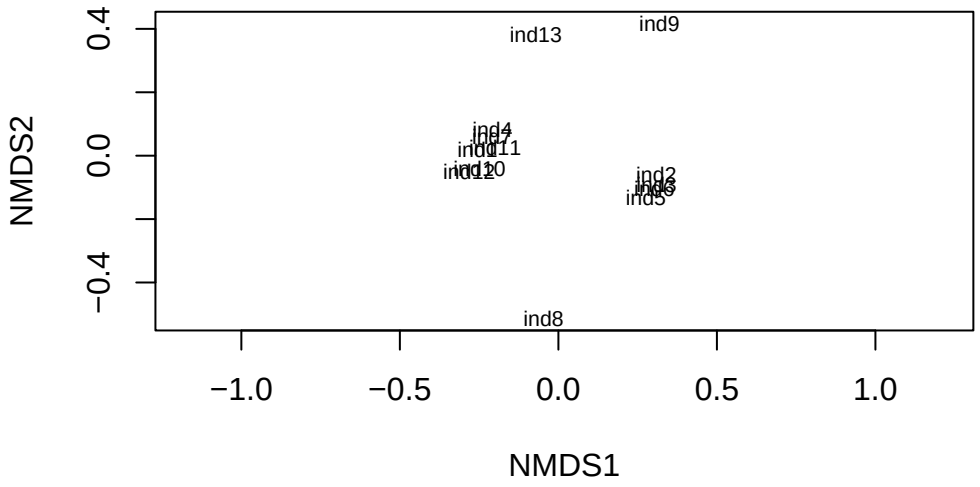
```
# Print mds results
mds1
```

```
Call:
metaMDS(comm = dist2, wascores = F)

global Multidimensional Scaling using monoMDS

Data:     dist2
```

```
Distance: user supplied

Dimensions: 2
Stress:     0.07227846
Stress type 1, weak ties
Best solution was not repeated after 20 tries
The best solution was from try 8 (random start)
Scaling: centring, PC rotation
Species: scores missing
```

```r
# Plot result
plot(mds1, type = 't')
```

```
species scores not available
```



The values have shifted around a bit but the structure and interpretation of the plot is the same. Let's continue on:

```r
# Run non-metric MDS - isoMDS
mds2 = isoMDS(dist2)
```

```
initial  value 24.760322
iter   5 value 14.153502
iter  10 value 12.254154
iter  15 value 11.639473
iter  20 value 11.360460
final  value 11.341572
converged
```

```
# Print output
mds2
```

```
$points
            [,1]         [,2]
ind1   0.5060798  0.103253718
ind2  -0.6157097 -0.285522725
ind3  -0.6133549 -0.310223762
ind4   0.5008471  0.144443835
ind5  -0.6264450 -0.279477605
ind6  -0.6228134 -0.325577873
ind7   0.4940123  0.147103149
ind8  -0.6783876  0.680257989
ind9  -0.2575043  1.075686777
ind10  0.5482152  0.033286470
ind11  0.5478840  0.082682890
ind12  0.5895070  0.001005794
ind13  0.2276697 -1.066918657

$stress
[1] 11.34157
```

The modelling algorithms seems to be a little different, and we end up with a different stress result - in this case, one that is above the 10% threshold (note that stress is in % in this function, unlike metaMDS where it is in proportion). Let's try another scree plot:

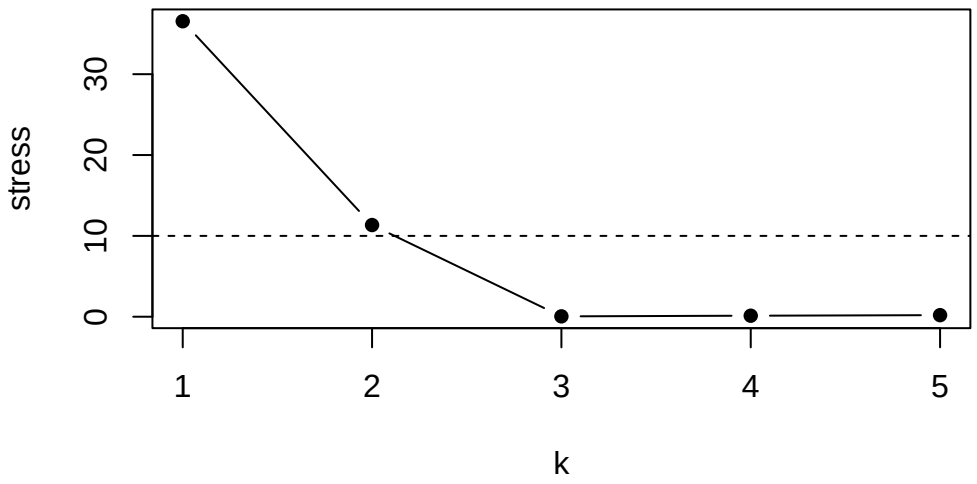
```
# Create a container object
scree = data.frame(k = 1:5, stress = NA)

# Loop through k 1 to 5
for(k in 1:5){

  # Run MDS
```

```
  mds = isoMDS(dist2, k = k) # Set k to our loop index

  # Pull out stress
  scree[k,'stress'] = mds$stress # Fill kth row of the column 'stress' in scree

} # End loop
```

```
# Print results
scree
```

```
  k      stress
1 1 36.54857293
2 2 11.34157190
3 3  0.04614441
4 4  0.12630298
5 5  0.19439990
```

```
# Make scree plot
plot(stress ~ k, data = scree, # Plot stress against k
     type = 'b', # Lines and points
     pch = 16) # Point 16 (filled circle)
abline(h = 10, lty = 'dashed') # Plot a dashed line at 0.1
```



In this case, it seems we're better off using 3 dimensions:

```
# Run non-metric MDS - isoMDS
mds2 = isoMDS(dist2, k = 3)
```

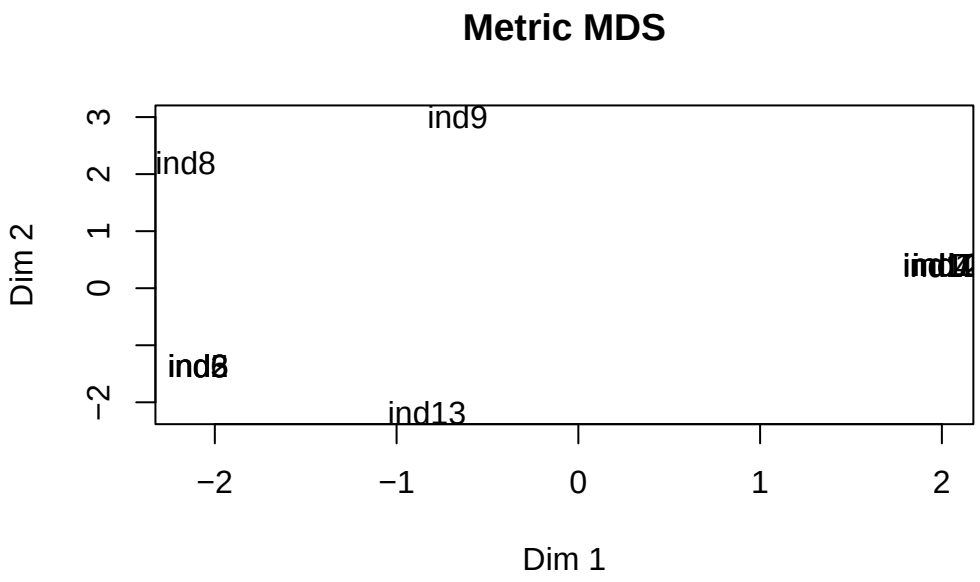
```
initial  value 18.960422
iter   5 value 11.725940
iter  10 value 6.417141
iter  15 value 4.149185
iter  20 value 1.466748
iter  25 value 0.764657
iter  30 value 0.449114
iter  35 value 0.302911
iter  40 value 0.156116
iter  45 value 0.087536
iter  50 value 0.046144
final  value 0.046144
stopped after 50 iterations
```

```
# Print output
mds2
```

```
$points
            [,1]       [,2]       [,3]
ind1   2.0028765  0.4079841 -0.4202376
ind2  -2.0901193 -1.3657278  1.2720375
ind3  -2.0910832 -1.3666943  1.2729356
ind4   2.0066696  0.4042391 -0.4161698
ind5  -2.0896284 -1.3631915  1.2769256
ind6  -2.0921681 -1.3637290  1.2724192
ind7   2.0032691  0.4111298 -0.4185160
ind8  -2.1600174  2.2033842 -1.8938700
ind9  -0.6641520  2.9972846  2.5707229
ind10  2.0009095  0.4046552 -0.4185896
ind11  2.0041166  0.4044773 -0.4197885
ind12  2.0019583  0.4023552 -0.4223419
ind13 -0.8326313 -2.1761669 -3.2555274

$stress
[1] 0.04614441
```

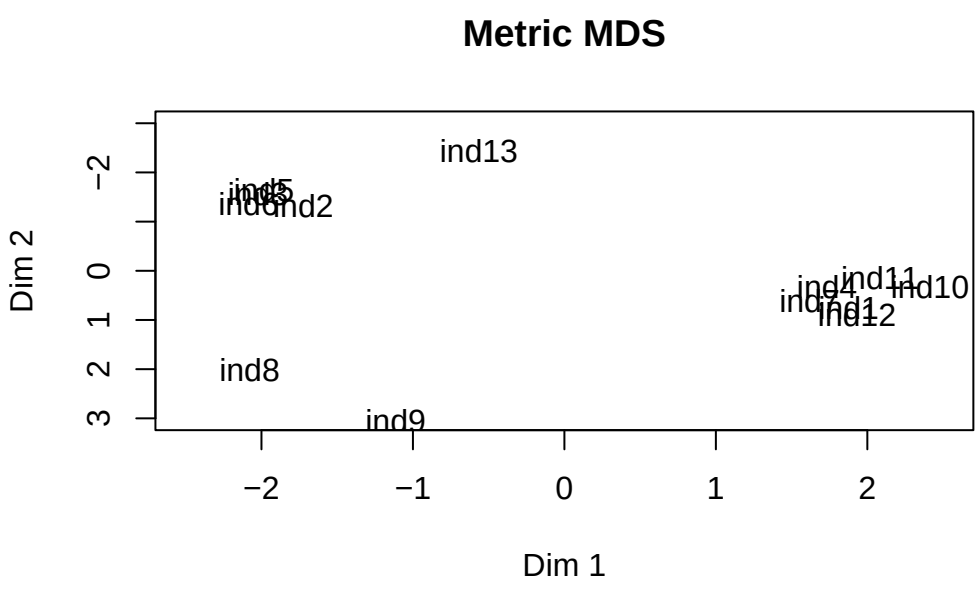
Let's plot our results:

```r
# Plot isoMDS
plot(mds2$points[,1], mds2$points[,2], # MDS dimension 1 and 2 values
     type = 'n', # Don't plot any points
     xlab = 'Dim 1', ylab = 'Dim 2', main = 'Metric MDS') # Labelling

# Plot individual names
text(mds2$points[,1], mds2$points[,2], rownames(data))
```



All of our grouped individuals are plotted on top of each other. Let's try adding some random jiggle so we can see them

```
# Plot isoMDS
plot(mds2$points[,1], mds2$points[,2], # MDS dimension 1 and 2 values
     type = 'n', # Don't plot any points
     xlab = 'Dim 1', ylab = 'Dim 2', main = 'Metric MDS', # Labelling
     xlim = c(-2.5, 2.5), ylim = c(3, -3)) # Set axis limits

# Set random seed for consistency
set.seed(1212)

# Plot individual names
text(mds2$points[,1] + rnorm(13, 0, 0.2), # Add random values pulled from a
     mds2$points[,2] + rnorm(13, 0, 0.2), # normal distribution with mean 0, sd 0.2
     rownames(data)) # Add names
```



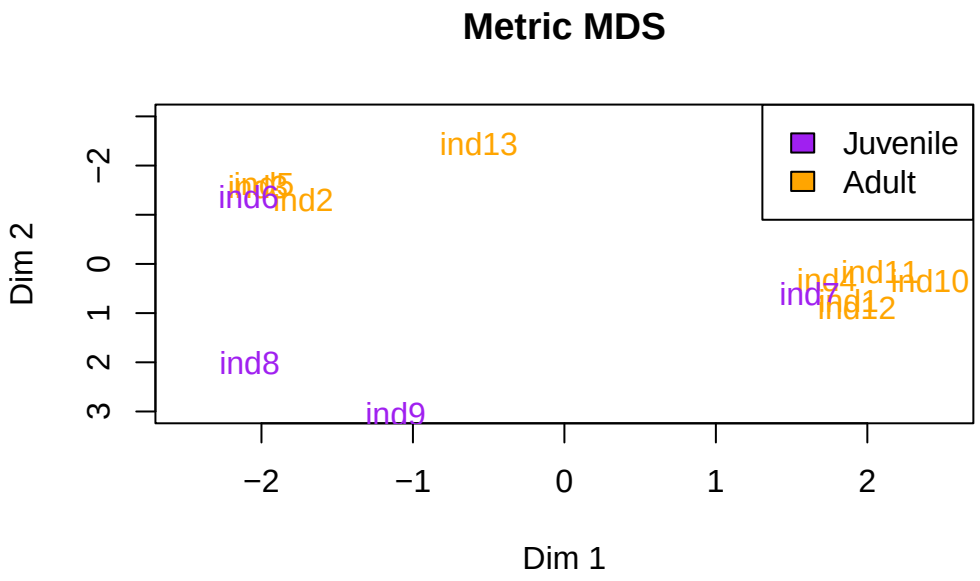
That's a bit better. We can also add some color to this plot if we want - say, individuals 6 to 9 are juveniles:

```
# Plot isoMDS
plot(mds2$points[,1], mds2$points[,2], # MDS dimension 1 and 2 values
     type = 'n', # Don't plot any points
     xlab = 'Dim 1', ylab = 'Dim 2', main = 'Metric MDS', # Labelling
     xlim = c(-2.5, 2.5), ylim = c(3, -3)) # Set axis limits

# Set random seed for consistency
set.seed(1212)

# Juvenile identifier
ad = c(rep(1,5), rep(0,4), rep(1,4)) # ad is 1 for first 5 and last 4
ad
```

```
[1] 1 1 1 1 1 0 0 0 0 1 1 1 1
```

```
# Plot individual names
text(mds2$points[,1] + rnorm(13, 0, 0.2), # Add random values pulled from a
     mds2$points[,2] + rnorm(13, 0, 0.2), # normal distribution with mean 0, sd 0.2
```

```
     rownames(data), # Add names
     col = ifelse(ad == 0, 'purple', 'orange')) # color
# Add a legend
legend('topright', legend = c('Juvenile', 'Adult'), fill = c('purple', 'orange'))
```



Does this plot match the previous one, and/or the cluster analyses?

3.4.2 Metric MDS

We can run metric MDS using the `cmdscale()` function:

```
# run metric MDS
mds3 = cmdscale(dist, eig = T)
mds3
```

```
$points
             [,1]       [,2]
ind1   5.84977914  2.7981654
ind2  -7.46899061 -0.4452479
ind3  -7.87360160  2.4742095
```

```
ind4   6.04970828 -1.1964346
ind5  -6.77887791  2.8518073
ind6  -7.21161499  3.9150940
ind7   4.79289905 -0.9896933
ind8  -2.46317365 -5.3304368
ind9  -1.86216019 -9.7770302
ind10  5.45394235  1.1317599
ind11  5.27120921 -0.1097836
ind12  6.20052885  3.6063451
ind13  0.04035206  1.0712450

$eig
 [1]  4.150450e+02  1.794715e+02  1.684147e+02  1.309366e+02  6.931537e+01
 [6]  5.811388e+01  3.306204e+01  1.780496e+01 -4.263256e-14 -8.643935e+00
[11] -2.208342e+01 -4.468321e+01 -7.952271e+01

$x
NULL

$ac
[1] 0

$GOF
[1] 0.4844901 0.5545014
```

Metric MDS doesn't have stress. Instead, we have to look at goodness of fit (GOF) to assess how well the analysis worked. GOF is similar to an R^2 value, where numbers closer to 1 indicate a better fit (though be wary of overfitting!). There are two different GOF values for each metric MDS.

As with the other MDS functions, k defaults to 2. We can make another scree plot:

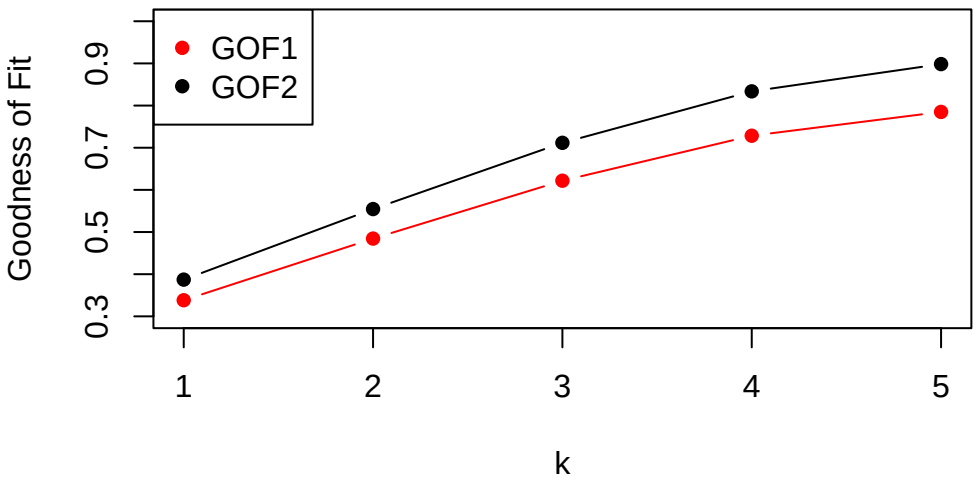
```r
# Create a container object
scree = data.frame(k = 1:5, GOF1 = NA, GOF2 = NA)

# Loop through k 1 to 5
for(k in 1:5){

  # Run MDS
  mds = cmdscale(dist, eig = T, k = k) # Set k to our loop index

  # Pull out stress
```

```
  scree[k,c(2,3)] = mds$GOF # Fill kth row of the GOF columns in scree

} # End loop
```

```
# Print results
scree
```

```
  k      GOF1      GOF2
1 1 0.3382331 0.3871096
2 2 0.4844901 0.5545014
3 3 0.6217365 0.7115806
4 4 0.7284408 0.8337043
5 5 0.7849281 0.8983543
```

```
# Make scree plot
plot(GOF2 ~ k, data = scree, # Plot stress against k
     type = 'b', # Lines and points
     pch = 16, # Point 16 (filled circle)
     ylab = 'Goodness of Fit', ylim = c(0.3, 1))
points(GOF1 ~ k, data = scree, type = 'b', pch = 16, col = 'red') # Add second GOF value
abline(h = 0.1, lty = 'dashed') # Plot a dashed line at 0.1
legend('topleft', pch = 16, legend = c('GOF1', 'GOF2'), col = c('red', 'black')) # Add leg
```



Goodness of fit scales linearly, so what k to use is more of a judgement call.

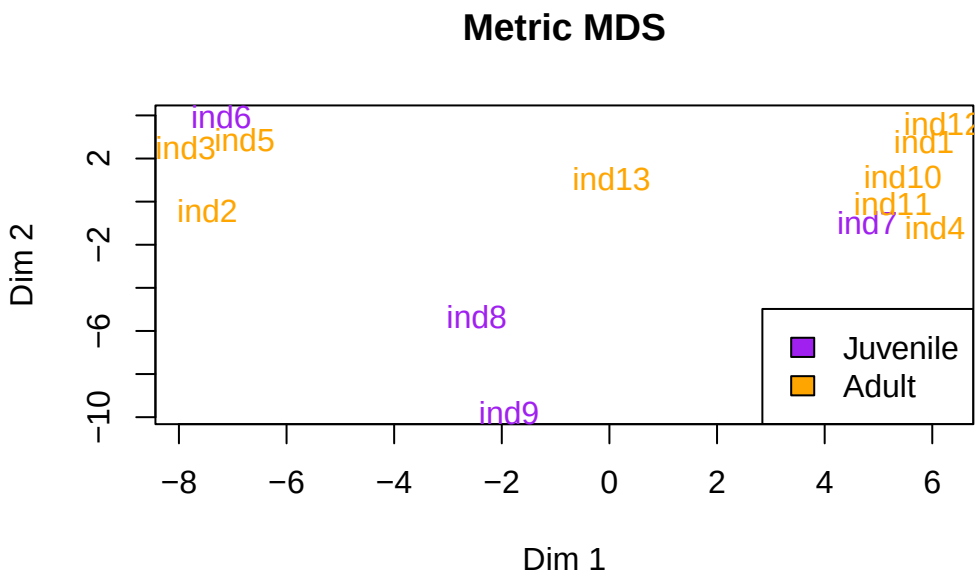
```
# run metric MDS
mds3 = cmdscale(dist, k=4, eig = T)
mds3
```

```
$points
             [,1]       [,2]       [,3]       [,4]
ind1   5.84977914  2.7981654  1.2685787 -0.6738375
ind2  -7.46899061 -0.4452479  2.6496404  2.1091477
ind3  -7.87360160  2.4742095  3.4326421  1.2339175
ind4   6.04970828 -1.1964346 -0.9306802 -1.2319775
ind5  -6.77887791  2.8518073 -1.2465315  2.3989342
ind6  -7.21161499  3.9150940 -1.9699687 -2.4369605
ind7   4.79289905 -0.9896933 -2.7067201 -0.2043847
ind8  -2.46317365 -5.3304368 -9.4034890  2.0126521
ind9  -1.86216019 -9.7770302  5.2905629 -1.1532342
ind10  5.45394235  1.1317599  0.3565324  4.1657272
ind11  5.27120921 -0.1097836  4.1678975  1.4238728
ind12  6.20052885  3.6063451 -0.2995254  1.5298300
ind13  0.04035206  1.0712450 -0.6089393 -9.1736869
```

```
$eig
 [1]  4.150450e+02  1.794715e+02  1.684147e+02  1.309366e+02  6.931537e+01
 [6]  5.811388e+01  3.306204e+01  1.780496e+01 -4.263256e-14 -8.643935e+00
[11] -2.208342e+01 -4.468321e+01 -7.952271e+01

$x
NULL

$ac
[1] 0

$GOF
[1] 0.7284408 0.8337043
```

Let's plot the first two dimensions:

```r
# Plot metric MDS
plot(mds3$points[,1], mds3$points[,2], # MDS dimension 1 and 2 values
     type = 'n', # Don't plot any points
     xlab = 'Dim 1', ylab = 'Dim 2', main = 'Metric MDS') # Labelling

# Plot individual names
text(mds3$points[,1], # Add random values pulled from a
     mds3$points[,2], # normal distribution with mean 0, sd 0.2
     rownames(data), # Add names
     col = ifelse(ad == 0, 'purple', 'orange')) # color
# Add a legend
legend('bottomright', legend = c('Juvenile', 'Adult'), fill = c('purple', 'orange'))
```

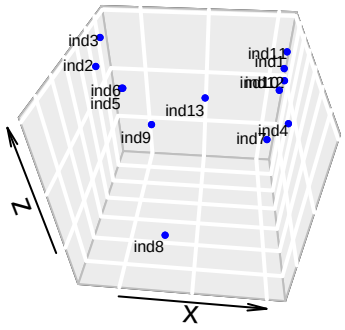


3.4.3 3D Plotting (Optional)

It may not be necessary, but if your MDS has more than 2 dimensions, you can try plotting it in three dimensions and see if it helps:

```
library(plot3D)

# Prepare data to plot
x = mds3$points[,1]
y = mds3$points[,2]
z = mds3$points[,3]

# Create 3D plot
scatter3D(x,y,z, colvar = NULL, col = 'blue',
          pch = 16, cex = 0.5, bty = 'g', theta = 5)

# Add text
text3D(x,
       # Add some jiggle to the labels
       y+rnorm(13, mean = 0, sd = 0.5), z + rnorm(13, mean = 0, sd = 0.5),
                   labels = names(mds3$points[,1]), add = T, colkey = F,
```

```
                    cex = 0.5, adj = 1, d = 2)
```



3.5 Mantel Test (Graduate Students Only)

We can infer to some extent whether juveniles and adults preferentially associate with each other from our colored MDS plots, but we can also test it statistically using a Mantel test. To run the Mantel test, we need to convert our adult index into a `dist` object:

```
# Create dist matrix for adults
ad_dist = dist(ad)
ad_dist
```

```
   1 2 3 4 5 6 7 8 9 10 11 12
2  0
3  0 0
4  0 0 0
5  0 0 0 0
6  1 1 1 1 1
7  1 1 1 1 1 0
8  1 1 1 1 1 0 0
```

```
9  1 1 1 1 1 0 0 0
10 0 0 0 0 0 1 1 1 1
11 0 0 0 0 0 1 1 1 1  0
12 0 0 0 0 0 1 1 1 1  0  0
13 0 0 0 0 0 1 1 1 1  0  0  0
```

Note this is dissimilarity: adult-juvenile pairs are assigned 1, and same-class pairs are assigned 0.

The Mantel test looks for correlation between this matrix and our original dissociation matrix, and statistically tests if the associations are different from what we would expect due to chance.

```
# Run mantel test
library(ade4)
mantel.rtest(ad_dist, dist, nrepet = 999)
```

```
Warning in is.euclid(m1): Zero distance(s)
```

```
Warning in is.euclid(m2): Zero distance(s)
```

```
Monte-Carlo test
Call: mantelnoneuclid(m1 = m1, m2 = m2, nrepet = nrepet)

Observation: 0.1686576

Based on 999 replicates
Simulated p-value: 0.073
Alternative hypothesis: greater

     Std.Obs  Expectation     Variance
 1.369210491 -0.001062026  0.015364686
```

It's very close, but we don't have statistically significant evidence that juveniles and adults associate preferentially with each other in this case.

3.6 Tips for your Assignment:

Some things you may want to think about for your assignment:

1. How would you pick which cluster analyses and MDS analyses are best for your data? Are they conceptual, or do they have to do with the results? Do they agree?

2. How would you interpret your statistical results biologically? You don't have to be right, but don't be vague, and don't contradict your results.

4 Assignment 1d: Multiple Linear Regression

5 Assignment 1e: Bayesian Data Analysis

Assignment Guidelines

This section will run through some general advice on how to put together your assignments, and how to make my life easier marking them:

General Advice

1. Read the grading rubric!

- It is quite objective. There is little latitude for part marks if you are missing parts that you need.

2. You don't need to tell me the statistical theory or background (unless it's relevant to your answers)

- All you have to do is answer the questions in the assignment. Anything you write outside of that is just eating up your page limit.

3. Put your biological interpretations together at the end

- Your interpretations are more likely to make sense and easier to mark if you put them at the end, and include all of your results together in them rather than inserting them throughout piecemeal.
- **Also, make sure your biological interpretations are consistent with your data/results! They don't have to be correct, but they have to match your data.**

4. Make sure your figures are readable

- I can't tell if your interpretation of your figures is correct if I can't interpret your figures.
- Alltext on figures should be readable.
- If you use color, make sure that the colors you use are clearly distinguishable.

5. You're not alone!

- If you have questions, come to the drop-in sessions, read the discussion boards, or email me to ask questions or set up a meeting if those don't work for you.

- Do the first two even if you don't have questions: You may find the answer to questions you didn't know you had.
- You're also welcome to ask questions after an assignment if you want to know why you were graded the way you were, or if you have questions about the comments provided or what you may have done wrong.

6. Ask if you need an extension

- We're pretty reasonable.

Submission Formatting

Here are some guidelines on how to submit your assignment to make my life easier. **You won't lose marks for not following them, but I would greatly appreciate it if you did.**

1. Hand in your assignment in 3 parts:

a) Your assignment text

- All assignment 1s have a 2 page limit. Put all your text first, with figures and tables together separately at the end. It is easier for me to tell how many pages you wrote this way. You're not going to lose marks if you're slightly over (this is not a writing class), if you're going to lose marks for writing too many pages I'm going to be able to tell anyways.
- Should be a doc or PDF file so I can open it in BrightSpace. Doesn't matter if it's produced through word, markdown, etc, as long as it's in one of those two formats.

b) Your script, submitted as a .txt file

- Submitting as a .txt file allows me to open the script in BrightSpace rather than having to download it if it's a .R file. It is much more difficult for me to run your script should I need to if you paste it in your assignment document.

c) The data you read into your script

- This is just to make it easy for me to run your script if I think there is a mistake.

2. Don't put your name on your assignments, in your scripts, or in any of your file names

- The BrightSpace system is anonymous so I mark your assignments blind. That doesn't work if you write your name.
- Delete your file paths in your script for submission if they have your name in them.

- **You do need to put your B0 number, data code number, and whether you're a graduate student or an undergraduate student.**

3. **Follow the assignment guide**

   - You don't have to follow this guide to get full marks on the assignments (as always, there are many correct answers when it comes to coding). That said, it's easier for me to follow what you're doing if you're doing the same thing as everyone else.
   - It's OK to do your own thing, but if you make a mistake, its going to be much harder for me to help you out, and it's going to take significantly more effort on my end to mark.