Multivariate Analysis Workshop

R. Alex Thompson & Madeline Franz

4/11/2022

Welcome to the Multivariate Statistics in Ecology and Earth Sciences Workshop 2022. The central aims of this workshop are: (i) introduce attendees to the basic principles behind various multivariate methods, (ii) outline a general strategy for deciding if a multivariate approach is best for your data, and (iii) work through examples of multivariate analysis in R, providing sample code and data for future exploration and learning.

**Overview of Multivariate Analysis and Multivariate Data**  
Technological advances have made it easier for researchers to measure many more variables at once than has been historically possible. Multivariate data lets us view a study system from a variety of perspectives (re: dimensions) and has helped us to learn a great deal about natural systems in the process. Yet, without multivariate analysis, it would be difficult to make sense of all of how these different puzzle pieces all fit together. Examples of this include coordinated trait responses, variable interactions, contrasts, and latent effects. Put simply, multivariate analysis can be thought of as a tool that allows us to examine all of the puzzle pieces at once to find those which fit “best” among all possible combinations.

**Why Multivariate Analysis**  
So, why multivariate analysis? The archetypal scientist might just as well argue for evaluating correlations among variables one-by-one. This technique might work for some variables (i.e. photoperiod and plant growth) but not so well for other variables that are clearly coupled (i.e. growth and photosynthesis). In the latter example, latent, hidden effects that we can’t (or simply don’t) measure or account for are responsible for driving relationships. This is a classic example of Gestalt: the whole is greater than the sum of its parts. Multivariate analysis is like a peak behind the curtain, letting us look at how different combinations or groups of variables affect another variable. More importantly, multivariate analysis can let us look at how multiple variables affect multiple other variables.

Another example is how observations might group themselves. As humans, we like to categorize or cluster things we see in nature. Species, clouds, rocks, etc. While these categories can be useful for making sense of the world, it is often the case that nature doesn’t care about our tidy categories. More often than not, we are interested in the cases where nature breaks our rules. How then, can we group phenomena more heuristically? The answer is found in multivariate analysis and we specifically address these examples in this tutorial. To understand how we can do this, the next section sets the stage for a geometric understanding of multivariate analysis.

**When To Use Multivariate Analysis (and When Not To)**  
While multivariate analysis can be an incredibly powerful tool, it can also be misused. How do we exploit the power of multivariate statistics, while avoiding some of these misuses? The first step is to evaluate whether or not your data is a good fit for the technique at hand. Once you think you have data that might benefit from a multivariate analysis, use the table below to check which of the methods we walk through in this course are a good candidate. Most modern statistical methods use multivariate perspectives and this is in no way a comprehensive guide but rather a “How to” for getting started.

**How to use Multivariate Analysis**  
In this tutorial, we will introduce you to four methods used in multivariate statistics: Canonical Correlation, Clustering Algorithms, Classification, and Ordination (Principal Components Analysis). In order to illustrate these techniques in the simplest way, we have chosen to use the infamous *iris* dataset provided in RStudio. This dataset, while simple, is very well-behaved. While we seldom have well-behaved data in practice, this makes teaching multivariate methods much easier. Not only will the basic principles of multivariate analysis become more clear, but our ability to make conclusions will be simplified. This is in line with aim (i), and is the motivating aim of this tutorial.

Let us start by first exploring the *iris* dataset, so that we may develop a more intuitive sense of the project at hand.

library(tidyverse) # a majority of the code in this tutorial will be based in the tidyverse

## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.0 ──

## ✓ ggplot2 3.3.2 ✓ purrr 0.3.4  
## ✓ tibble 3.0.4 ✓ dplyr 1.0.2  
## ✓ tidyr 1.1.2 ✓ stringr 1.4.0  
## ✓ readr 1.4.0 ✓ forcats 0.5.0

## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()

library(flextable) # this is a good package for making nice tables but feel free to use whatever you like

##   
## Attaching package: 'flextable'

## The following object is masked from 'package:purrr':  
##   
## compose

iris<-iris # load the data that comes with R  
iris # view the data

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
## 1 5.1 3.5 1.4 0.2 setosa  
## 2 4.9 3.0 1.4 0.2 setosa  
## 3 4.7 3.2 1.3 0.2 setosa  
## 4 4.6 3.1 1.5 0.2 setosa  
## 5 5.0 3.6 1.4 0.2 setosa  
## 6 5.4 3.9 1.7 0.4 setosa  
## 7 4.6 3.4 1.4 0.3 setosa  
## 8 5.0 3.4 1.5 0.2 setosa  
## 9 4.4 2.9 1.4 0.2 setosa  
## 10 4.9 3.1 1.5 0.1 setosa  
## 11 5.4 3.7 1.5 0.2 setosa  
## 12 4.8 3.4 1.6 0.2 setosa  
## 13 4.8 3.0 1.4 0.1 setosa  
## 14 4.3 3.0 1.1 0.1 setosa  
## 15 5.8 4.0 1.2 0.2 setosa  
## 16 5.7 4.4 1.5 0.4 setosa  
## 17 5.4 3.9 1.3 0.4 setosa  
## 18 5.1 3.5 1.4 0.3 setosa  
## 19 5.7 3.8 1.7 0.3 setosa  
## 20 5.1 3.8 1.5 0.3 setosa  
## 21 5.4 3.4 1.7 0.2 setosa  
## 22 5.1 3.7 1.5 0.4 setosa  
## 23 4.6 3.6 1.0 0.2 setosa  
## 24 5.1 3.3 1.7 0.5 setosa  
## 25 4.8 3.4 1.9 0.2 setosa  
## 26 5.0 3.0 1.6 0.2 setosa  
## 27 5.0 3.4 1.6 0.4 setosa  
## 28 5.2 3.5 1.5 0.2 setosa  
## 29 5.2 3.4 1.4 0.2 setosa  
## 30 4.7 3.2 1.6 0.2 setosa  
## 31 4.8 3.1 1.6 0.2 setosa  
## 32 5.4 3.4 1.5 0.4 setosa  
## 33 5.2 4.1 1.5 0.1 setosa  
## 34 5.5 4.2 1.4 0.2 setosa  
## 35 4.9 3.1 1.5 0.2 setosa  
## 36 5.0 3.2 1.2 0.2 setosa  
## 37 5.5 3.5 1.3 0.2 setosa  
## 38 4.9 3.6 1.4 0.1 setosa  
## 39 4.4 3.0 1.3 0.2 setosa  
## 40 5.1 3.4 1.5 0.2 setosa  
## 41 5.0 3.5 1.3 0.3 setosa  
## 42 4.5 2.3 1.3 0.3 setosa  
## 43 4.4 3.2 1.3 0.2 setosa  
## 44 5.0 3.5 1.6 0.6 setosa  
## 45 5.1 3.8 1.9 0.4 setosa  
## 46 4.8 3.0 1.4 0.3 setosa  
## 47 5.1 3.8 1.6 0.2 setosa  
## 48 4.6 3.2 1.4 0.2 setosa  
## 49 5.3 3.7 1.5 0.2 setosa  
## 50 5.0 3.3 1.4 0.2 setosa  
## 51 7.0 3.2 4.7 1.4 versicolor  
## 52 6.4 3.2 4.5 1.5 versicolor  
## 53 6.9 3.1 4.9 1.5 versicolor  
## 54 5.5 2.3 4.0 1.3 versicolor  
## 55 6.5 2.8 4.6 1.5 versicolor  
## 56 5.7 2.8 4.5 1.3 versicolor  
## 57 6.3 3.3 4.7 1.6 versicolor  
## 58 4.9 2.4 3.3 1.0 versicolor  
## 59 6.6 2.9 4.6 1.3 versicolor  
## 60 5.2 2.7 3.9 1.4 versicolor  
## 61 5.0 2.0 3.5 1.0 versicolor  
## 62 5.9 3.0 4.2 1.5 versicolor  
## 63 6.0 2.2 4.0 1.0 versicolor  
## 64 6.1 2.9 4.7 1.4 versicolor  
## 65 5.6 2.9 3.6 1.3 versicolor  
## 66 6.7 3.1 4.4 1.4 versicolor  
## 67 5.6 3.0 4.5 1.5 versicolor  
## 68 5.8 2.7 4.1 1.0 versicolor  
## 69 6.2 2.2 4.5 1.5 versicolor  
## 70 5.6 2.5 3.9 1.1 versicolor  
## 71 5.9 3.2 4.8 1.8 versicolor  
## 72 6.1 2.8 4.0 1.3 versicolor  
## 73 6.3 2.5 4.9 1.5 versicolor  
## 74 6.1 2.8 4.7 1.2 versicolor  
## 75 6.4 2.9 4.3 1.3 versicolor  
## 76 6.6 3.0 4.4 1.4 versicolor  
## 77 6.8 2.8 4.8 1.4 versicolor  
## 78 6.7 3.0 5.0 1.7 versicolor  
## 79 6.0 2.9 4.5 1.5 versicolor  
## 80 5.7 2.6 3.5 1.0 versicolor  
## 81 5.5 2.4 3.8 1.1 versicolor  
## 82 5.5 2.4 3.7 1.0 versicolor  
## 83 5.8 2.7 3.9 1.2 versicolor  
## 84 6.0 2.7 5.1 1.6 versicolor  
## 85 5.4 3.0 4.5 1.5 versicolor  
## 86 6.0 3.4 4.5 1.6 versicolor  
## 87 6.7 3.1 4.7 1.5 versicolor  
## 88 6.3 2.3 4.4 1.3 versicolor  
## 89 5.6 3.0 4.1 1.3 versicolor  
## 90 5.5 2.5 4.0 1.3 versicolor  
## 91 5.5 2.6 4.4 1.2 versicolor  
## 92 6.1 3.0 4.6 1.4 versicolor  
## 93 5.8 2.6 4.0 1.2 versicolor  
## 94 5.0 2.3 3.3 1.0 versicolor  
## 95 5.6 2.7 4.2 1.3 versicolor  
## 96 5.7 3.0 4.2 1.2 versicolor  
## 97 5.7 2.9 4.2 1.3 versicolor  
## 98 6.2 2.9 4.3 1.3 versicolor  
## 99 5.1 2.5 3.0 1.1 versicolor  
## 100 5.7 2.8 4.1 1.3 versicolor  
## 101 6.3 3.3 6.0 2.5 virginica  
## 102 5.8 2.7 5.1 1.9 virginica  
## 103 7.1 3.0 5.9 2.1 virginica  
## 104 6.3 2.9 5.6 1.8 virginica  
## 105 6.5 3.0 5.8 2.2 virginica  
## 106 7.6 3.0 6.6 2.1 virginica  
## 107 4.9 2.5 4.5 1.7 virginica  
## 108 7.3 2.9 6.3 1.8 virginica  
## 109 6.7 2.5 5.8 1.8 virginica  
## 110 7.2 3.6 6.1 2.5 virginica  
## 111 6.5 3.2 5.1 2.0 virginica  
## 112 6.4 2.7 5.3 1.9 virginica  
## 113 6.8 3.0 5.5 2.1 virginica  
## 114 5.7 2.5 5.0 2.0 virginica  
## 115 5.8 2.8 5.1 2.4 virginica  
## 116 6.4 3.2 5.3 2.3 virginica  
## 117 6.5 3.0 5.5 1.8 virginica  
## 118 7.7 3.8 6.7 2.2 virginica  
## 119 7.7 2.6 6.9 2.3 virginica  
## 120 6.0 2.2 5.0 1.5 virginica  
## 121 6.9 3.2 5.7 2.3 virginica  
## 122 5.6 2.8 4.9 2.0 virginica  
## 123 7.7 2.8 6.7 2.0 virginica  
## 124 6.3 2.7 4.9 1.8 virginica  
## 125 6.7 3.3 5.7 2.1 virginica  
## 126 7.2 3.2 6.0 1.8 virginica  
## 127 6.2 2.8 4.8 1.8 virginica  
## 128 6.1 3.0 4.9 1.8 virginica  
## 129 6.4 2.8 5.6 2.1 virginica  
## 130 7.2 3.0 5.8 1.6 virginica  
## 131 7.4 2.8 6.1 1.9 virginica  
## 132 7.9 3.8 6.4 2.0 virginica  
## 133 6.4 2.8 5.6 2.2 virginica  
## 134 6.3 2.8 5.1 1.5 virginica  
## 135 6.1 2.6 5.6 1.4 virginica  
## 136 7.7 3.0 6.1 2.3 virginica  
## 137 6.3 3.4 5.6 2.4 virginica  
## 138 6.4 3.1 5.5 1.8 virginica  
## 139 6.0 3.0 4.8 1.8 virginica  
## 140 6.9 3.1 5.4 2.1 virginica  
## 141 6.7 3.1 5.6 2.4 virginica  
## 142 6.9 3.1 5.1 2.3 virginica  
## 143 5.8 2.7 5.1 1.9 virginica  
## 144 6.8 3.2 5.9 2.3 virginica  
## 145 6.7 3.3 5.7 2.5 virginica  
## 146 6.7 3.0 5.2 2.3 virginica  
## 147 6.3 2.5 5.0 1.9 virginica  
## 148 6.5 3.0 5.2 2.0 virginica  
## 149 6.2 3.4 5.4 2.3 virginica  
## 150 5.9 3.0 5.1 1.8 virginica

Now, if we inspect our data we see that it contains 150 observations over just 5 variables, one of them being categorical. This is great, because when , computation is greatly simplified.

Let us first consider the scenario where we collect this data with the goal of interrogating relationships between the variables, grouping of variables, and the species. There’s a lot to unpack here. First, we could simply run linear (or nonlinear) regression models on each pair of variables separately, giving six unique figures and statistical outputs. That’s not to mention if we decide to group by species, which further complicates things. In reality, most of our data contains variables that are correlated. A motivating example is climate data, where temperature, precipitation, and VPD can all interact. Clearly, this is not preferable since correlations between variables can lead to unnecessarily large p, make prediction more difficult, and may lead to overfitting.

Let us check for correlations in our data.

cor(iris[1:4])

## Sepal.Length Sepal.Width Petal.Length Petal.Width  
## Sepal.Length 1.0000000 -0.1175698 0.8717538 0.8179411  
## Sepal.Width -0.1175698 1.0000000 -0.4284401 -0.3661259  
## Petal.Length 0.8717538 -0.4284401 1.0000000 0.9628654  
## Petal.Width 0.8179411 -0.3661259 0.9628654 1.0000000

It appears that most of our variables are correlated, which limits what we may include in a regression model. However, what if we want to test how a set of multiple variables correlates with another set of multiple variables? In mathematical terms, we split our data into two (not necessarily equally sized!) matrices: and . Although the number of variables in each of the matrices can be unequal, the number of observations must be equal for the analysis to work.

We are now ready to introduce our first technique, *Canonical Correlation*. Canonical correlation is an analysis that allows us to identify and test the strength of relationships between sets of variables. It does this by creating linear combinations of each variable set, and calculating the correlations between them. This can be used to, for example, compare a number of climate variables to multiple metrics of plant growth.

Before we can do anything, however, it’s best to scale and center our data. What does this mean? In short, if some of our variables have more variance than others, or are measured at a different scale (i.e. values are in the 100s versus 10s), this can create issues with multivariate techniques. A fundamental component of any multivariate analysis is the concept of distance. If one variable has greater variance than the other variables, it is likely that your results will be strongly influenced by this single variable. Obviously, this is bad. To correct for this we can use R’s built in scale and center functions to give our data unit variance - essentially homogenizing the variance of each column so that they are equally weighted in the calculations.

iris\_scale<-scale(iris[1:4])

#Install packages  
library(CCA) #for canonical correlations

## Loading required package: fda

## Loading required package: splines

## Loading required package: Matrix

##   
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':  
##   
## expand, pack, unpack

## Loading required package: fds

## Loading required package: rainbow

## Loading required package: MASS

##   
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':  
##   
## select

## Loading required package: pcaPP

## Loading required package: RCurl

##   
## Attaching package: 'RCurl'

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

## Loading required package: deSolve

##   
## Attaching package: 'fda'

## The following object is masked from 'package:graphics':  
##   
## matplot

## Loading required package: fields

## Loading required package: spam

## Loading required package: dotCall64

## Loading required package: grid

## Spam version 2.7-0 (2021-06-25) is loaded.  
## Type 'help( Spam)' or 'demo( spam)' for a short introduction   
## and overview of this package.  
## Help for individual functions is also obtained by adding the  
## suffix '.spam' to the function name, e.g. 'help( chol.spam)'.

##   
## Attaching package: 'spam'

## The following object is masked from 'package:Matrix':  
##   
## det

## The following objects are masked from 'package:base':  
##   
## backsolve, forwardsolve

## Loading required package: viridis

## Loading required package: viridisLite

##   
## Try help(fields) to get started.

library(CCP) #to assess statistical significance of canonical correlations  
  
#Canonical Correlation  
##Select the variables to be used in each of the two categories  
sepal <- iris[,c(1:2)]  
petal <-iris[,c(3:4)]  
  
matcor(sepal, petal) #displays correlations within and between categories

## $Xcor  
## Sepal.Length Sepal.Width  
## Sepal.Length 1.0000000 -0.1175698  
## Sepal.Width -0.1175698 1.0000000  
##   
## $Ycor  
## Petal.Length Petal.Width  
## Petal.Length 1.0000000 0.9628654  
## Petal.Width 0.9628654 1.0000000  
##   
## $XYcor  
## Sepal.Length Sepal.Width Petal.Length Petal.Width  
## Sepal.Length 1.0000000 -0.1175698 0.8717538 0.8179411  
## Sepal.Width -0.1175698 1.0000000 -0.4284401 -0.3661259  
## Petal.Length 0.8717538 -0.4284401 1.0000000 0.9628654  
## Petal.Width 0.8179411 -0.3661259 0.9628654 1.0000000

iris\_cc <- cc(sepal, petal)#this function performs the canonical correlation analysis  
  
# display the canonical correlations  
iris\_cc$cor

## [1] 0.9409690 0.1239369

# raw canonical coefficients  
iris\_cc[3:4]

## $xcoef  
## [,1] [,2]  
## Sepal.Length -1.0689812 0.5797395  
## Sepal.Width 0.8549914 2.1462750  
##   
## $ycoef  
## [,1] [,2]  
## Petal.Length -0.8491249 -1.918708  
## Petal.Width 0.6938022 4.809531

# compute canonical loadings  
iris\_cc2 <- comput(sepal, petal, iris\_cc)  
  
# display canonical loadings  
iris\_cc2[3:6]

## $corr.X.xscores  
## [,1] [,2]  
## Sepal.Length -0.9290009 0.3700774  
## Sepal.Width 0.4767332 0.8790480  
##   
## $corr.Y.xscores  
## [,1] [,2]  
## Petal.Length -0.9313286 0.01769543  
## Petal.Width -0.8604722 0.05015620  
##   
## $corr.X.yscores  
## [,1] [,2]  
## Sepal.Length -0.8741611 0.04586624  
## Sepal.Width 0.4485912 0.10894647  
##   
## $corr.Y.yscores  
## [,1] [,2]  
## Petal.Length -0.9897548 0.1427778  
## Petal.Width -0.9144533 0.4046915

rho <- iris\_cc$cor #make a vector of the canonical correlation coefficients  
  
n <- dim(sepal)[1] #number of observations of each variable  
p <- length(sepal) #number of 'independent' variables  
q <- length(petal) #number of 'dependent' variables  
  
p.asym(rho, n, p, q, tstat = 'Wilks') #specify test statistic to be used. could also use Hotelling, Pillai or Roy

## Wilks' Lambda, using F-approximation (Rao's F):  
## stat approx df1 df2 p.value  
## 1 to 2: 0.1128174 144.337576 4 292 0.0000000  
## 2 to 2: 0.9846396 2.293196 1 147 0.1320893

Often, we want to group our data. While grouping can be done a priori during the data collection process (i.e. treatments, sites, etc.), sometimes we don’t know what the groups are. There are two general approaches associated with post hoc grouping: *Clustering* and *Classification*.

Let’s start with Classification. Here we will be discussing linear discriminant analysis, a technique that allows us to observe and test the differences between different groups or populations of observations. This analysis can elucidate the relative importance of different variables in separating populations. Once the discriminatory analysis has been performed, the resulting model can be used to classify future observations into one of the populations.

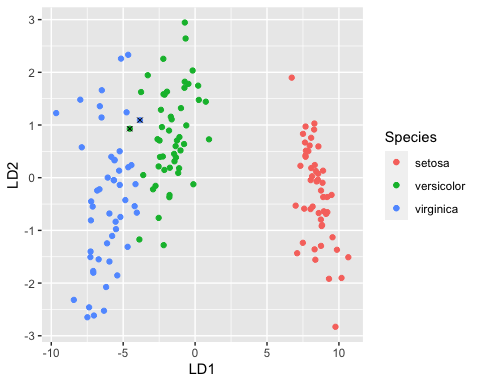
library(MASS)  
library(ggplot2)  
  
#make this example reproducible  
set.seed(666)  
  
#Here we split the data into training and testing sets. We use 70% of dataset as training set (randomly selected) and the remaining 30% as testing set  
sample <- sample(c(TRUE, FALSE), nrow(iris), replace=TRUE, prob=c(0.7,0.3)) #select 70% of the rows/observations  
train <- iris[sample, ] #the selected 70% become the training data  
test <- iris[!sample, ] #everything not selected (30%) becomes testing data  
  
#fit the linear discriminant analysis model on the training data  
iris\_lda <- lda(Species~., data=train)  
  
#view model output  
iris\_lda

## Call:  
## lda(Species ~ ., data = train)  
##   
## Prior probabilities of groups:  
## setosa versicolor virginica   
## 0.3267327 0.3069307 0.3663366   
##   
## Group means:  
## Sepal.Length Sepal.Width Petal.Length Petal.Width  
## setosa 5.000000 3.378788 1.430303 0.2363636  
## versicolor 6.048387 2.812903 4.361290 1.3645161  
## virginica 6.559459 2.981081 5.529730 2.0162162  
##   
## Coefficients of linear discriminants:  
## LD1 LD2  
## Sepal.Length 0.8685692 0.3266404  
## Sepal.Width 1.3269025 -2.3797084  
## Petal.Length -2.3779689 0.9052843  
## Petal.Width -3.1199182 -3.0365410  
##   
## Proportion of trace:  
## LD1 LD2   
## 0.9937 0.0063

#use LDA model to make predictions on test data  
predicted <- predict(iris\_lda, test)#here the predictions are made  
  
mean(predicted$class==test$Species)#here we test the predictions by comparing them to the known species to determine if the predictions are correct

## [1] 0.9591837

missclass <- predicted$class==test$Species #here we can see which predictions are missclassified (show up as 'FALSE')  
  
#define data to plot  
lda\_train\_plot <- cbind(train, predict(iris\_lda)$x) #adds 2 columns of discriminants to the training points  
lda\_test\_plot <- cbind(test, predicted$x) #adds 2 columns of discriminants to the testing points  
  
miss\_dat <- cbind(lda\_test\_plot, missclass) #adds the 'misclassification' true/false to the testing points  
lda\_miss\_plot <- miss\_dat[missclass==FALSE,] #selects only the rows that were missclassified  
  
#create plot  
ggplot(lda\_train\_plot, aes(LD1, LD2)) + #initiates the plot and sets axes based on the linear discriminats  
 geom\_point(aes(color = Species)) + #adds the training points to the graph, colored by their species  
 geom\_point(data = lda\_test\_plot, aes(color = Species)) + #adds testing points to the graph, colored by their actual species  
 geom\_point(data = lda\_miss\_plot, shape=4) #puts an 'x' over the misclassified points



Clearly, *Classification* requires some a priori knowledge about our data. Furthermore, it’s strength comes in classifying *new* observations. But what if we don’t have new observations and we just want to group our existing observations? Well, this is where *Clustering* algorithms can help. Clustering algorithms are highly effective mathematical tools that leverage concepts of distance between points in multiple dimensions () to find the best groups. Sometimes our data makes this easy. Sometimes it is not.

**K-means** There are many different algorithms that can be used in clustering. These include hierarchical methods like single linkage or average linkage. What we’re going to cover here is the k-means algorithm which is widely used across many fields. The basic idea is to imagine a moving window across all of your data points. This window is calculating the distance between points and clustering them based on minimum distances. Essentially, you specify k clusters (k = 1, 2, 3, etc.) and run the algorithm. The distance of each observation is compared the average distance of each of the k clusters. The points in each cluster are shuffled based on this algorithm until the within-cluster distance is minimized while the between-cluster distance is maximized. Intuitively this makes sense - we group close observations together and maximize the distance between the (fewest number of) groups.

*K-means algorithm*  
*1. Specify the number of K clusters*  
*2. Randomly shuffle K observations into K clusters to find the initial centroids*  
*3. For every observation, compute the sum of squared distances from initial centroids and assign to closest cluster*  
*4 Repeat 2 & 3 until within cluster sum of squared distances is minimized, and between cluster sum of squared distances is maximized*

Let us take the *iris* data make an attempt at clustering it.

k2<-kmeans(iris[1:4],centers=2) # run the kmeans algorithm included in R  
# first we specify the data where the [1:4] command specifies which columns to evaluate  
# the centers command is where we specify the number of clusters   
k2 # let's look at our clusters

## K-means clustering with 2 clusters of sizes 53, 97  
##   
## Cluster means:  
## Sepal.Length Sepal.Width Petal.Length Petal.Width  
## 1 5.005660 3.369811 1.560377 0.290566  
## 2 6.301031 2.886598 4.958763 1.695876  
##   
## Clustering vector:  
## [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  
## [75] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2  
## [112] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  
## [149] 2 2  
##   
## Within cluster sum of squares by cluster:  
## [1] 28.55208 123.79588  
## (between\_SS / total\_SS = 77.6 %)  
##   
## Available components:  
##   
## [1] "cluster" "centers" "totss" "withinss" "tot.withinss"  
## [6] "betweenss" "size" "iter" "ifault"

# just two clusters capture 77% of the variance in the data. That's pretty good!  
k3<-kmeans(iris[1:4],centers=3) # let's try 3 clusters to see if we can improve our results  
k3 # looking at our results

## K-means clustering with 3 clusters of sizes 38, 50, 62  
##   
## Cluster means:  
## Sepal.Length Sepal.Width Petal.Length Petal.Width  
## 1 6.850000 3.073684 5.742105 2.071053  
## 2 5.006000 3.428000 1.462000 0.246000  
## 3 5.901613 2.748387 4.393548 1.433871  
##   
## Clustering vector:  
## [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  
## [38] 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3  
## [75] 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 3 1 1 1 1 3 1 1 1 1  
## [112] 1 1 3 3 1 1 1 1 3 1 3 1 3 1 1 3 3 1 1 1 1 1 3 1 1 1 1 3 1 1 1 3 1 1 1 3 1  
## [149] 1 3  
##   
## Within cluster sum of squares by cluster:  
## [1] 23.87947 15.15100 39.82097  
## (between\_SS / total\_SS = 88.4 %)  
##   
## Available components:  
##   
## [1] "cluster" "centers" "totss" "withinss" "tot.withinss"  
## [6] "betweenss" "size" "iter" "ifault"

# we get marginal improvement, increasing the variance explained by two percentage points  
# given the tradeoff, it seem slike k=2 clusters is probably best  
# recall: we want to maximize variance explained in the fewest number of clusters  
  
# now we want to update our original dataset to reflect these clusters  
group<-k2$cluster # let's isolate the vector that defines which observations belong in which cluster  
iris2<-cbind(iris,group) # combine the original data with the cluster vector  
iris2$group<-as.factor(iris2$group)  
  
# now we want to explore things a big more.   
# we know that there are 3 groups in our data: the 3 species  
# Do we have patterns in how our clusters are grouped?   
# let's make a figure to visualize this clustering  
library(ggplot2) # we'll use ggplot to make figures  
group\_fig<-ggplot(iris2,aes(Species,fill=group)) + geom\_bar()  
# we can see pretty clearly now that all of the setosa and virginica species are classified into separate groups  
# it looks like versicolor, however, is split between clusters  
# this warrants further investigation  
# let's extract our cluster centers for each variable, and inspect what makes our clusters different  
centers<-k2$centers # extract the centers  
centers # view the centers

## Sepal.Length Sepal.Width Petal.Length Petal.Width  
## 1 5.005660 3.369811 1.560377 0.290566  
## 2 6.301031 2.886598 4.958763 1.695876

# group 1 can be described as having longer sepals, and overall larger petals  
# group 2 has shorter sepals that are more wide, with much smaller petals than group 2  
# so, perhaps our versicolor species that were classified as group 2 were simply outliers  
# let's move on to PCA, and see how we can apply these clusters in a more interesting way that let's us compare them in more detail.

The essential goal of most multivariate methods is to try and reduce the dimensionality of our data in a manner that let’s us make sense of it. As humans, we cannot even visualize dimensions greater than 3, let alone explain it. We’ve developed incredible tools that leverage theorems from linear algebra to reduce matrices of multiple dimensions into a smaller subset of dimensions that retains the original variability of the data. What does this mean? Let’s go back to linear regression.

In their simplest form, linear regression models predict a y variable from an x variable, often coming in the form of . For most research objectives, this form (and its derivatives) seems to work just fine. In the multivariate case, however, we often want to explore several variables at once. We can use a technique called *Principal Components Analysis* to reduce a set of variables into a minimal set of variables through a process called *eigenanalysis*. Let’s briefly discuss this process so you have a better idea of how PCA works, and why it’s effective.

**Eigenanalysis: The Basics**  
Consider a dataset you’ve collected at one of your study sites. Let this dataset have random variables, , , , and . We can express our dataset, $\bf{X}$, in the form $\bf{X} = c\_1v\_1+c\_2v\_2+c\_3v\_3+c\_4v\_4$. Clearly, this presents issues if we want to visualize and analyze our data. In the sense of a scatterplot, each variable represents a new dimension and now we have 4 dimensions. Eigenanalysis allows us to scale this 4-D coordinate system in 1, 2, or 3 dimensions. Let’s break down some of the very basic elements of this process a bit further.

$$
\begin{matrix}
\bf{X} = c\_1v\_1+c\_2v\_2+c\_3v\_3+c\_4v\_4\\
We~want~to~express~our~data~in~linear~form~such~that~it~can~be~represented~as:\\
Y=X\beta\\
Essentially,~\beta~is~a~scaling~feature~that~takes~linear~combinations~of~all~of~our~variables\\
and~makes~new~variables:~PC1,~PC2,~etc.,~yielding~the~form:\\
Y\_1=e\_1X\_1+\cdot\cdot\cdot+e\_pX\_p\\
\cdot\\
\cdot\\
\cdot\\\
Y\_p=e\_1X\_1+\cdot\cdot\cdot+e\_pX\_p\\
\end{matrix}
$$

We are not going to concern ourselves with how to calculate eigenvalues, as we can do this easily in R. *(Note: for those interested, this stackexchange comment (*[*https://stats.stackexchange.com/questions/220243/the-proof-of-shrinking-coefficients-using-ridge-regression-through-spectral-dec*](https://stats.stackexchange.com/questions/220243/the-proof-of-shrinking-coefficients-using-ridge-regression-through-spectral-dec)*) gives a really great and intuitive explanation of eigenvalues using spectral decomposition. I find the explanation pretty intuitive and informative, and hopefully it helps you too.)* Essentially each is an *eigenvalue* (AKA Principal Component) and each is an *eigenvector* with p rows corresponding to p variables (essentially they are scale factors). Each (formally presented as ) is essentially a projection of all of the variables, scaled by their corresponding eigenvectors in a 1-dimensional subspace spanning the column space of the initial data matrix. This projection is what we see when we visualize a PCA using a biplot.

Some notes on PCA:

1. PCA is often referred to as a dimension reduction technique, though it is not a true method of dimension reduction. In reality, PCA is simply a projection method that attempts to summarize a large number of parameters (variables) into fewer dimensions.
2. PCA should typically be done on standardized and centered variables (i.e. a correlation matrix). Often, variables in a dataset can vary in scale. This can overwhelm the PCA method and create situations where a single variable is driving the summarization. Obviously this is problematic and the sensitivity of this method to such problems warrants the use of correlation matrices in PCA.
3. Eigenvalues and eigenvectors are not unique to PCA. Such values are used in a variety of other statistical approaches not covered here. Having a good sense of the role of eigenvalues and what they are will prepare you for a future that likely includes encountering them again.
4. R has two functions for PCA: prcomp() and princomp(). Each method employs different approaches for finding the PCAs but the function prcomp() is typically the preferred method due to greater accuracy. However, in practice, you shouldn’t really notice any difference.

Let’s dive in.

First, we’ll start with a very simple PCA using the Iris dataset.

library(tidyverse)  
library(ggbiplot)

## Loading required package: plyr

## ------------------------------------------------------------------------------

## You have loaded plyr after dplyr - this is likely to cause problems.  
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:  
## library(plyr); library(dplyr)

## ------------------------------------------------------------------------------

##   
## Attaching package: 'plyr'

## The following objects are masked from 'package:dplyr':  
##   
## arrange, count, desc, failwith, id, mutate, rename, summarise,  
## summarize

## The following object is masked from 'package:purrr':  
##   
## compact

## Loading required package: scales

##   
## Attaching package: 'scales'

## The following object is masked from 'package:viridis':  
##   
## viridis\_pal

## The following object is masked from 'package:purrr':  
##   
## discard

## The following object is masked from 'package:readr':  
##   
## col\_factor

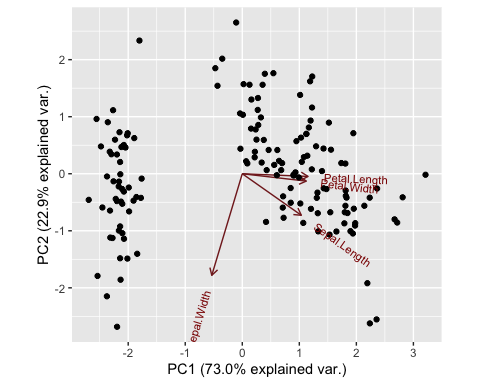
# run a PCA on the iris data   
# we set center = TRUE and scale = TRUE to make sure the variance of one variable doesn't overpower the analysis and give weird results  
iris\_pca<-prcomp(iris[1:4],center=TRUE,scale=TRUE)  
# let's look at our results  
summary(iris\_pca)

## Importance of components:  
## PC1 PC2 PC3 PC4  
## Standard deviation 1.7084 0.9560 0.38309 0.14393  
## Proportion of Variance 0.7296 0.2285 0.03669 0.00518  
## Cumulative Proportion 0.7296 0.9581 0.99482 1.00000

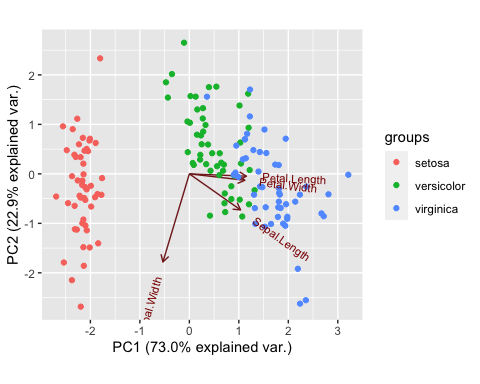
# it looks like 95% of the variance in the data can be explained with just two components!  
iris\_pca # This line of code let's us look into how each variable changes along each PC

## Standard deviations (1, .., p=4):  
## [1] 1.7083611 0.9560494 0.3830886 0.1439265  
##   
## Rotation (n x k) = (4 x 4):  
## PC1 PC2 PC3 PC4  
## Sepal.Length 0.5210659 -0.37741762 0.7195664 0.2612863  
## Sepal.Width -0.2693474 -0.92329566 -0.2443818 -0.1235096  
## Petal.Length 0.5804131 -0.02449161 -0.1421264 -0.8014492  
## Petal.Width 0.5648565 -0.06694199 -0.6342727 0.5235971

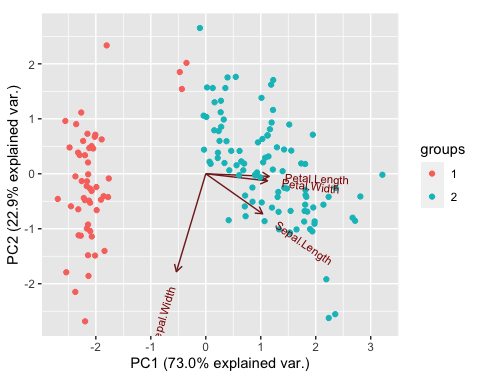
# Now, let's plot it and see what it looks like  
ggbiplot(iris\_pca,choices=1:2,scale=0.05)



# we can also color the points by species to see if our groups are clustering by species  
ggbiplot(iris\_pca,choices=1:2,scale=0.05,groups=iris$Species)



# It looks like species are definitely breaking out along PC1  
# Remember our k-means clusters above? Let's bring that data in and see if our clusters do the same thing  
ggbiplot(iris\_pca,choices=1:2,scale=0.05,groups=iris2$group)



Now, let’s use some more complicated data and show how things can get weird and beautiful.

library(tidyverse)  
library(ggbiplot)  
  
# let's start by creating a new working directory wherever you have your data stored  
#setwd("") # insert the name of your directory into the quotations and run this line  
  
# let's load in all of the data  
pd<-read.csv("masterpd.csv")%>%  
 dplyr::mutate(psipd=abs(psipd))%>%  
 dplyr::mutate(psimd=abs(psimd))  
leaf<-read.csv("leaftraits.csv")  
elements<-read.csv("CN.csv")  
  
# run the PCA on just the water potential data  
pd\_pca<-prcomp(pd[3:6],center=TRUE,scale.=TRUE) # runs the PCA on selected columns (MUST BE NUMERIC)  
# center and scale terms must be true in order to account for different data types  
pd\_summary<-as.data.frame(summary(pd\_pca)$importance)   
group<-pd$Group  
pd\_plot<-ggbiplot(pd\_pca,ellipse=TRUE,groups=group)  
  
# run PCA with combined leaf trait and water potential data  
lpd<-merge(leaf,pd)%>%  
 dplyr::select(-date,-psidif)  
lpd\_pca<-prcomp(lpd[8:13],center=TRUE,scale.=TRUE)  
lpd\_summary<-as.data.frame(summary(lpd\_pca)$importance) # 65% of the data with 2, 81% of the data with 3  
group<-lpd$Group  
lpd\_plot<-ggbiplot(lpd\_pca,ellipse=TRUE,groups=group)  
  
# run PCA on all of the data  
all<-merge(lpd,elements)%>%  
 dplyr::select(-date,-C,-Ni,-CN,-N15,-fvfm)%>%  
 dplyr::rename(Microsite=Group)%>%  
 dplyr::mutate(Microsite=ifelse(Microsite=="High Closed Canopy","High Position - Closed Canopy",ifelse(Microsite=="High Exposed","High Position - Open Canopy",ifelse(Microsite=="Low Closed Canopy","Low Position - Closed Canopy",""))))%>%  
 dplyr::rename("\u03A8pd"=psipd)%>%  
 dplyr::rename("\u03A8md"=psimd)%>%  
 dplyr::rename("\u03C1chlorophyll"=chlcont)%>%  
 dplyr::rename(LMA=lma)%>%  
 dplyr::rename("Al:As"=huber)  
all\_pca<-prcomp(all[8:13],center=TRUE,scale.=TRUE)  
all\_summary<-as.data.frame(summary(all\_pca)$importance) # 65% with 2  
all\_rotation<-as.data.frame(all\_pca$rotation)  
rownames(all\_rotation) <- rownames(all\_pca$rotation)  
  
  
# obviously adding more variables isn't making things better  
# let's look at the elemental analysis alone  
elmnt\_pca<-prcomp(elements[3:7],center=TRUE,scale.=TRUE)  
elmnt\_summary<-as.data.frame(summary(elmnt\_pca)$importance) # 71% of the data with 2  
group<-elements$Group  
elmnt\_plot<-ggbiplot(elmnt\_pca,ellipse=TRUE,groups=group)  
  
  
# it looks like the PCA on uncorrelated variables is best   
# the secrete lies with eigenvalues - highly correlated variables tend to have very small eigenvalues  
# let's make a table and inspect it a little bit  
all\_rotation<-all\_rotation%>%  
 rownames\_to\_column()%>%  
 dplyr::rename(Variable=rowname)  
  
allpca\_tab<-flextable(all\_rotation)  
  
  
# let's plot the results and see what it looks like  
  
group<-all$Microsite  
all\_plot12<-ggbiplot(all\_pca,choices=1:2,ellipse=TRUE,groups=group)

Clearly things are complicated with PCA. It seems to be most effective with a subset of the full dataset. This should make it clear that it isn’t really a dimension reduction method. What’s happening? This is called the **Curse of Dimensionality**.

We have p variables, and that p can be large or small relative to n. As we progressively add variables and p increases, the dimension of the sample space is increasing. This effectively pulls observations apart as variance tends to increase (things get more spread out!). As this distance between observations increases, it becomes more difficult to summarize all of the variables in just a subset of the sample space. This is called the curse of dimensionality and is a huge problem in inference and prediction. We won’t dive into this here, but this (SOURCE) resource explains things quite well.

**Conclusion**  
Multivariate analysis can be an important tool in ecology. It has been used for many years (see the work of P.C. Eilou for early use of ordination methods in ecology), and will certainly be used for many more to come. As ecologists we often deal with very small sample sizes (field work is hard!) in high dimensional settings. This can be problematic and we shouldn’t expect multivariate analysis to solve such problems as the curse of dimensionality. Prior to any statistical approach, it’s the job of the researcher to carefully consider which variables are important. As we saw with the PCA data, this was clearly an issue and it extends to clustering, classification, and canonical correlation.  
While we only covered a few subjects here, mostly regarding statistical *inference*, many multivariate methods have been developed for prediction. Many of us use predictive models in some capacity to conduct our research. Even if we don’t consider ourselves *modelers*, models are frequently used in the literature. Our hope is that the techniques we’ve introduced you to here will aid you in understanding, developing, testing, or even critiquing models in your field. With these tools you’ve been provided here, you should be ready to employ simple linear multivariate statistical methods on your own data, and even explore more complex approaches later.