R basics workshop notes

AZ2CH Teaching Team

2023-06-04

Basic R

09:00 Introduction

The goal of this book is to provide a 'how-to' guide to connect you to the world of data science. We focus on the fundamentals of the R programming language and its applications in biology.

In Biology, there are two dominant programming languages: **Python** and **R**. We focus on **R** because it is more commonly used in published statistical analyses in Biology, and it is a bit easier to learn. As you will see, it is very easy to walk through the fundamentals and generate graphs and statistical analyses with just a few hours of practice.

Learning a new language is not easy. Learning a programming language is not easy either. Here are a few specific tips to become fluent in R:

1. **Get organized and PLAN.** Use a personal calendar and schedule sufficient time to deal with error messages. This is important to accept, though it can be difficult: troubleshooting your code often takes more time than planning and writing it, especially when you are starting to learn.
2. **Apply what you learn.** You will start to develop a toolbox of coding techniques from day one. Look for opportunities to apply them whenever you can. Try to reframe small projects or tasks in terms of what you can address with your R toolkit. Even if it takes a lot longer to code than to use other methods, the extra time will reinforce your coding skills, saving time in the long run. Take time to think about what coding tools you can apply.
3. **Experiment.** Try new things, make mistakes, solve problems.
4. **Devote time.** Set aside large blocks of time (2+ hours), to immerse yourself in your coding lessons or project.
5. **Focus. Eliminate distractions.** Turn off your notifications. Put your phone and computer on 'airplane mode'. Do whatever it takes to work without interruption.
6. **Learn to Troubleshoot.** If you get stuck, Google: "How do I _____ in R". Look for answers from a website called Stack Overflow: https://stackoverflow.com/. If you can't figure out what an error means, paste it into Google. Again, look for answers from Stack Overflow.
7. **Socialize.** Find a coding support group or find a few others to form your own group. Discuss problems and successes. Read other people's code to see how they tackle problems. Rarely is there one single 'right' way to code something.
8. **Git 'er done.** When you are starting out, the 'right' way to code is whatever it takes to get the code to do what you want. Don't let perfection be the enemy of the good: messy code that works is 100% better than efficient code that never runs.

9. **Improve.** As you get more comfortable you can start to think about cleaner, clearer, more efficient ways to code. As you advance, look for ways to do the same thing faster and with fewer lines of code.

10. **Embrace Failure.** Every error is a learning opportunity.

11. **Read** the documentation for the function or package you are using. Don't worry if you don't understand everything. Be sure to take the time to read it slowly and try to understand as much as you can. Try searching online for terms or phrases that are not familiar to you. You will come across these again in the future, so you are investing time now for future payoff. In addition to the built-in help in R, often the repository on The Comprehensive R Archive Network (CRAN) (https://cran.r-project.org/) or Bioconductor (https://www.bioconductor.org/) will include vignettes or tutorials as pdf files with worked examples.

09:15 R and RStudio

Install R

Install the latest version of R: https://cran.r-project.org/

Versions are available for Windows, MacOS and Linux operating systems. Immediately we can see one of the advantages of learning to code in R – we can move code across computing platforms quite easily, as long as R is installed there.

Install R Studio

https://rstudio.com/products/rstudio/ download/#download

R Studio is an Integrated Development Environment (IDE). Once you install R Studio, go ahead and run the program.

Helpful Tabs

Environment keeps track of all of the objects in your program- ming environment.

History keeps track of the code you have run.

Files similar to the Finder (MacOS) or File Explorer (Windows), starting with the working directory.

Plots are where your plots are created.

Packages show which packages you have installed, and which have been loaded.

Help provides documentation for R functions.

Console is important enough to get its own section.

The console is one of the most important tabs in R Studio. It's usu- ally the main tab that opens on the left when you first start R Studio. You'll see a little chevron (>) with a cursor after it. This is the R Console, which is the part of R Studio that actually runs the R program. Everything in this window shows you what would happen if you ran the code outside of R Studio, for example on a high perfomance com- puting cluster like the ones maintained by Compute Canada, Microsoft Azure, Amazon Web Services, or Queen's University's own Centre for Advanced Computing. Everything in R studio is built around helping you to perform tasks in R, as shown through the R Console.

R Projects

Working with relative paths can get a little bit confusing, especially if you start using setwd() (not covered in this section). A good way to avoid confusion is to make an R project in R Studio

File–>New Project–>New Directory–>New Project

Then make a name for your project and choose where you want to save it on your computer. Now quit R studio and look inside the new directory that you just cre- ated on your computer. You will see a file with the name of your project followed by .Rproj.

Let's start with a simple R script.

To open a R script, go to File –> New File –> R Script.

To run a script, you have to send the text from the script tab to the console tab.

1. Copy and paste manually.
2. Highlight the code you want to run and click the Run button on the top-right corner of the script tab. The run button sends the highlighted text from the script to the console. If you click the Run button without highlighting text, it will send whatever text is on the same line as your cursor.
3. If you press **Ctl + Enter** (Windows) or **Cmd + Return** (Mac) it will do the same thing – this is the shortcut for the **Run** button.
4. There are other options if you press the tiny triangle next to the **Run** button, including **Run All**.
5. **Ctl/Cmd + Shift + Enter/Return** is a shortcut for **Run All**.

Packages

Packages in R contain functions – small programs that contain func- tions you can use. A few are loaded automatically when you start R, including the stats and base packages. Let's go ahead and install some useful packages.

To install the packages, open R Studio and look for the Console tab. Type this into your console:

```
#install.packages('rmarkdown')
#install.packages('dplyr')
#install.packages('knitr')
```

The install.packages() function downloads the package and saves it on your computer. You only need to do this one time, though you may want to do it periodically to update to the latest version of the package.

Once a package is installed on your computer, it will be available to run in R with the library() command.

9:30 R Markdown

R Markdown is a powerful format for quickly making high-quality re- ports of your analysis. You can embed code and all kinds of output, including graphs, and output them to a Word Document, PDF or web- site.

File-> New-> R Markdown

set up

Let's load these packages.

At the top of this window is a little icon that says 'Knit'. The Knit icon, knits together your text file into a rendered html document.

If you want to output this as a pdf file, then you can simply choose Print, and then Print to PDF in your web browser.

YAML Header

Every R Markdown file starts with a YAML header, which contains some basic information about the file. A YAML header is generated automatically when you make a new .Rmd file in RStudio, but not all elements are needed. Depending on what options you choose, it might look something like this:

The main advantage of R Markdown (.Rmd) over regular Markdown (.md) is the ability to easily print, format, and execute embedded R code for graphs, tables, and calculations.

Plain Text

Plain text is converted into paragraph format.

To start a new paragraph, press enter twice. This is important – if you only press enter once, then the two paragraphs will knit together into the same paragraph.

Similarly, if you inlcude more than two lines between paragraphs, these will be ignored when you render the R Markdown document.

Try adding paragraphs of text separated by pressing enter 1, 2 and 3 times. Then, knit to html to see how these are rendered in the final output.

Formatted Texts

You can format text with * or _

*italics*or_italics_: italics

boldor___bold___: bold

Use greater-than sign for block quotes, eg. > TIP: quote

Headers

Add headers with up to six hash marks (#). Each additional # denotes a sub-heading of the previous (sub)heading.

Header 1

Sub-Header = Header 2

Sub-Sub Header = Header 3

Sub-Sub-Sub Header = Header 4

others

Use two dashes (–) for short–dash (a.k.a. 'n-dash').

Use three dashes (—) for long — dash (a.k.a. m-dash).

Links

Links have a special format. The text you want the user to see goes in square brackets, followed immediately by the file or html link in regular brackets, with no space in between. You can use both web links and relative path links.

Colautti Lab Website

This should produce a link if you are reading this electronically:

Instead of linking, you can embed the image directly by adding an exclamation point. The text in square brackets becomes the figure caption.

Figure 1: Linked .png file

Embed R Code

You can format text to look like code using the back-tick character.

`Use single tick mark to invoke code formatted text.`

The Back Tick is a strange looking character usually located on the same key as the tilde (~) on English keyboards. Don't confuse the back tick with the single quotation mark.

You can incorporate blocks of R code using three back ticks with r in curly brackets. Write some code, then add three more tick marks to signal the end of the code chunk. By defaule, your code will run when you convert your R Markdown to html, showing the output. This is a great way to include graphs and the output of statistical models.

```
#your code goes here
```

Ctl-Alt-i is a nice shortcut in R Studio for adding code chunks quickly.

For macs, use Command + Option + i.

You can name your code chunks by adding a name right after the r separated only by a space. Naming code chunks is very handy for trou- bleshooting. When you knit your file, the name of each chunk is listed in the Render tab in R Studio. If there is an error, you can see which code chunk is causing the error. Note that the name cannot contain spaces, and it can be followed by a comma to specify options for the code chunk.

{r code-chunk-name, eval=F}

9:45 Are Birds Real? Dataset Overview

Let's import our data

Let's return the first (**head()**) or last (**tail()**) parts of the data to make sure the correct data has been loaded. We can also use this function to check out the headers in the dataset.

You can click on **MyData** in the environment panel to get the full view of the dataset.

Let's look at the dimensions and the summary of the data.

From a quick glace, we know that there are 1000 entries (rows) and 14 different columns of different variables such as IndID, which stands for individual ID, Species, N_Eggs (number of eggs), Body_Size_g (body size in grams), etc... The first four columns give information about the samples, and locations of the samples and the rest of the columns are different measurements.

Let's start with some basics. Using basic R functions, we can obtain some simple statistics: **sum()**, **mean()**, **median()**, **var()**, **sd()**.

Let's look at the mean of the egg sizes.

! the out put is **NA** This indicates the mean of the egg size in our data set is not available. The most common culprit for these sort of outputs are missing values in the column.

Let's check to see if there are any missing values.

```
#This looks at the whole data set and determines if there is a missing value or not. FALSE means no mis

#Let's narrow this down to just the egg size.

#We can clearly see here that there are numerous TRUE returns, which all indicate missing values.


#This will tell you which rows have missing values.
```

In order to properly calculate the mean, we have to remove the missing values.

Using **dplyr** functions, we can explore our dataset a bit further. Some useful functions include: **filter()** for rows, **slice()**, **select()** for columns, **arrange()**, and many more!

Using dplyr - pipe operator %>%

using a pipe operator allows us more flexibility, and allows us to combine functions tha can be used on top of each other.

We can filter specificially for rows of data that are missing its egg size values.

```
 #all the rows that are not NAs
 #all the rows that are NAs
```

Let's clean up our data to omit rows with missing values, and use this dataset going forward.

Let's try some simple functions using the pipe operator.

```
#filtering out all the rows which its number of eggs is greater than or equal to 5.
#filtering out all the rows that are collected at SiteA
#filtering for both SiteA and Family Cardinalidae
#filtering for both egg number greater than or equal to 5 and matching the family Cardinalidae using a
#filtering for individuals with number of eggs above or equal to 5 OR below 3, using the | symbol.
```

This is an example of arranging the egg size in increasing order and removing missing values in the egg size column.

Let't take this a little further. First, remove the missing values. Second, let's group the samples by families. Now, let's calculate the mean and the standard deviation for egg size for each family.

We can take this and create a new data frame containing these information.

Pay close attention to the Family names. There are some incosistent caplitalizations. Let's go ahead and clean this column up.

There is a very handy 2-page 'cheat sheet' (https://www.rstudio.com/ wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf), which you can also access through R Studio under the Help menu: Help -> Cheatsheets. There are several other links here too, including **ggplot** and **dplyr**.

Visualizing raw data can be a very helpful way when starting your data analysis. As a preview, here is a simple histogram displaying the distribution of number of eggs in our data set.

Working with dates

dates

As biologists, we often work with dates or time. We may want to ana- lyze the date a sample was collected, or a measurement was taken. But dates are often encoded in formats that don't fit neatly into the usual data types that we've worked with so far. The **lubridate** package pro- vides a convenient framework for switching between human-readable dates and mathematical relationships among them – for example, the number of days, minutes, or seconds between two time points.

```
#install.packages("lubridate")
```

It can be convenient to automatically include the current date or time, especially when you are producing reports in R.

We can get the date as a date object in the year-month-day format,

And we can get the date along with the time as a datetime object, in the year-month-day hour:minute:second timezone format.This is an important distinction, because the datetime object extends on the date object to include hours, minutes, seconds, and time zone.

Human-readable dates come in many different forms, which we can encode as strings. Here are some examples that we might see for en- coding the end-date of the most recent 5,126-year-long Mesoamerican calendar used by the ancient Maya civilization:

The lubridate package has a number of related functions that corre- spond to the order of entry – d for day, m for month, and y for year:

Notice the flexibility here! Some dates have dashes, dots, spaces, and commas! Yet, all are easily converted to a common object type. On the surface, these objects look like simple strings, but compare the struc- ture of the date object with its original input string:

Notice how the first is a simple chr character object, whereas the sec- ond is a Date object. The date object can be treated as a numeric vari- able, that outputs as a readable date. For example, what if we want to know what the date 90 days before or after?

When using date objects, R will even account for the different number of days in each month, and adjust for leap years!

As with any function in R, we can apply ymd() to a column in a data frame or any other vector of strings:

The elements must have the have the same order as the function, but surprisingly they don't have to have the same format:

Note the warning and the resulting output. The first, third, and fourth elements are converted, even though they have different formats. The second element is replaced with NA because the order is not year, month, day, as required by ymd().

datetime

The datetime object adds a time element to the date. Just as there are different functions to translate different date formats, there are different date time functions.Each date time function starts with one of the three date functions we used earlier, but then adds time elements after an underscore _ to define nine different functions. Here are a few examples:

10:30 BREAK

Visualizations

11:00 Into to basic visualizations

In this first visualization section, we will learn how to use base R to make simple graphs and how to customize them. Will use The Are Birds Real data which we have called `MyData`. ### Why use R for graphing?

```
R has many tools for graphing and graph customization. It can also be used to create professional looki
```

Accessibility

Before we start, we should keep in mind that the colours you choose are important. For example, in Western European countries there is a cultural bias towards associating red spectrum colours (red, orange, yellow) with hot or danger. While blue spectrum colours (blue, cyan, purple) are associated with cold or calm. Using blue for hot temperatures and red for cold temperatures on a graph may confuse people.

In addition, many people have some form of colour blindness making it difficult or impossible to see certain colours. The package viridis is helpful for making graphs with colour palettes that are colourblind friendly, but this can only be used with ggplot2.

Also note that both spellings of "colour" (colour or color) work in R.

Data Setup

Loading the data: You can most likely skip this step since we worked on the same dataset in the previous section.

```
#MyData <- read.csv("AreBirdsRealData.csv", header=TRUE)
```

Look at the data.

Making a quick plot using base R

We will be creating a scatter plot using two continuous variables. Looking at the data we can see that body size and egg size are an integer and numeric respectively data, making them continuous.

Customizing the Graph

Adding Axis Labels and Title We get a scatter plot, but it doesn't look the best. Let's change the axis labels and add a title.

Figure 2: COFFEE

Changing Point Shape Let's change the shape of the points. Using the pch argument you can change the shape of the points. Using '?points' you can look up all the different point shapes available in the help window.

We are going to use 'pch = 19'.

Changing Point Colour Let's now change the colour of the points. Using the argument 'col' we can change point colour. We can change point colour in two different ways.

1. Using the colour name ("blue")
2. Using the Hex value of the colour ("#0000FF")

Method 1: Using the colour name.

Method 2: Using the colour's hex value.

We can find a list of all available colours in R using colours().

There are also 5 pre-made colour palettes in base R: 'rainbow()', 'heat.colors()', 'terrain.colors()', 'topo.colors()', and 'cm.colors()'.

You can also use the packages 'RColorBrewer' and 'wesanderson' to get more pre-made colour palettes.

Point Size We can also change the size of the points using the argument 'cex'. Changing the number with 'cex' allows you to increase or decrease point size.

Increase point size by setting cex = 3

11:15 Intro to ggplot2 for custom visualizations

In this section, we will learn the basics to using the package ggplot2 and some basic ways to customize graphs.

Getting Started with ggplot2

Install the `ggplot2` package using the `install.packages()` function the first time you want to use it. This installs it on your local computer, so you only need to do it once – though it is a good idea to re-install periodically to update to the most recent version.

```
#install.packages("ggplot2")
```

Once it is installed, you still need to load it with the `library` function if you want to use it in your code.

Using ggplot2

The ggplot function has two main components:

1. The ggplot function that defines the input data structure. Often has a nested aesthetic function 'aes()' to define your plotting variables (x and y variables). And a 'data =' parameter to define input data (the data you want to use for the graph).
2. And the 'geom_<name()>' function that defines geometry output. This tells R what kind of graph you want (bar, histogram, scatter).

Additionally, when adding new lines of code to the graphs an '+' is needed to connect them together.

Let's try making the same scatter graph from before using ggplot2.

We get the same graph from before.

Customizing graphs with ggplot2

Like before with base R, simple graphs made with ggplot2 don't look very nice. But we can change that.

Adding a Title and Axis Labels Adding a title and axis labels are like to base R but there are a few differences. First you use the function 'labs()'. Second the x-axis title, y-axis title, and main title are 'x =', 'y =', and 'title =' respectfully.

Changing point colour To change the colour of points we use the parameter 'colour ='. Like with base R you can use the colour name to specifiy which colour to use.

But different from base R is the function 'rgb()' for 'colour =' parameter. Using the 'rbg()' function you can make any colour you want. This function takes three values corresponding to the intensity of red, green, blue light respectively. Uses value range from 0 (no colour) to 1 (brightest intensity).

Something else that ggplot2 can do is assign colours as a set of values defined in vector (the input data in 'aes()').

As you can see, birds from different sites are now different colours.

Changing Point Shape Just like with colour you can use a parameter to change point shape. Using 'shape =' you can change the shape of the points to one of the 26 shapes available. These shapes use the same number code that we used before when graphing with base R.

And like before you can specify a certain variable in the data as shape.

Birds from different sites are represented as different shapes.

Changing point Size And finally point size can be changed by using the parameter 'size =' with a number to increase or decrease point size.

And again, variables can be specified to have different point sizes.

Different sites now are represented with different point sizes.

Themes You might want to use a fixed set of customization variables to use for all of you graphs. Instead of individually changing each graph you can use or create a theme and then apply it to all your graphs.

Themes are added to plots using the function 'theme_<name()>' where the name part is the name of the theme you want to use.

The package ggplot2 already has a set of 7 pre-made themes that you can use.

They are: 'theme_gray()', 'theme_bw()', 'theme_linedraw()', 'theme_light()', 'theme_dark()', 'theme_minimal()', and 'theme_classic()'.

There is also 'theme_void()' which is a completely empty theme.

Let's use 'theme_classic'.

As you can see, the all the grid lines are gone and the background is now white instead of gray.

You can modify the existing pre-made themes or create your own custom theme.

You can also use other people's custom theme and modify that to suit your needs. The theme 'theme_pub()' is a modified version of the pre-made 'theme_classic()' theme. This theme is formatted to create graphs and figures geared towards presentations or publication.

```
source("http://bit.ly/theme_pub")
```

Let's use this theme on our scatter plot.

As you can see this theme changed the formatting of the axis titles and main title, in addition to removing the grid lines and changing the background colour. Once again, this theme can be modified.

To set a theme for the entire code you can use the function 'theme_set()' and the theme will be used for all graphs you create so you don't have to add the theme manually to each graph.

For more information on ggplot2 and all the customization options available, there is a cheat sheet available on RStudio's website:

https://posit.co/resources/cheatsheets/?type=posit-cheatsheets/ ## 11:30 Histograms and Boxplots

We can then look at the first few rows of data using the head() function

Usually when we only have a **single continuous** variable to graph, then we are interested in looking at the frequency distribution of values. This is called a **frequency histogram**. The frequency histogram shows the distribution of data – how many observations (y-axis) for a particular range of values or *bins* (x-axis). Now we can create the histogram. Regardless of the type of graph we are creating in ggplot2, we always start with the ggplot() function, which creates a canvas to add plot elements to. It takes two parameters.

The first argument is a data frame. Here we want to use home_data. The second argument is a mapping from columns in the data frame to plot aesthetics. This mapping must call the aes() function. Here we map the price column to the x-axis.

This won't draw anything useful by itself. To make this a histogram, we add a histogram geometry using geom_histogram().

Note the two mai Also note the warning message about `binwidth` in `stat_bin()`. This is not a problem, it is just R telling us that it chose a default value that may not be idea. We can try different values of `binwidth=` in the `geom_histogram()` function to specify the width of the bins along the x-axis.

we will change the colors of the histogram. We can customize the color of the outlines of each bar using the color attribute, and we can change the fill of the bars using the fill attribute of geom_histogram(). We will fill the bars with blue and change the outline color to white.

You can customize the colors of the histogram bars. Here each color represents a bird family.

Next, we add titles and labels to our graph using the labs() function. We set the x, y, and title attributes to our desired labels.

If we want to move the legend on our graph, for instance, when we visualize the condition in different colors, we can use the theme() function and the legend.position attribute.

Finally, we can visualize data in different groups in separate graphs using facets. This will split the visualization into several subplots for each category. This can be done using the facet_grid() function. We will visualize the distributions of prices by different condition values below.

If we have a categorical and a continuous variable, we usually want to see the distribution of points for the two variables. The box plot is a handy way to quickly inspect a few important characteristics of the data:

1. **median**: middle horizontal line (i.e. the 50th percentile)
2. **hinges**: top and bottom of the boxes showing the 75th and 25th percentiles, respectively

3. **whiskers**: vertical lines showing the highest and lowest values (excluding outliers, if present)
4. **outliers**: points showing outlier values more than 1.5 times the inter-quartile range (i.e. 1.5 times the distance from the 25th to 75th percentiles). Note that not all data sets will have outlier points.

If you would like to flip these box-plots from a vertical orientation to a horizontal orientation, the code is almost exactly the same. The only addition is adding + coord_flip() to the end of the phrase.

You can change labels

You can also add colors

We can add multiple categories with color showing the 3 sites for each bird family

11:45 Tansformations

There are cases that transforming our data can help it reach normality and linearity of your data, which is one of the assumptions we will need to take into account when performing linear models. The transformations we choose depends on the type of data we are dealing with,but in biological data, transforming one or both of your x and y axes with log() or exp() will often get you close to a linear relationship.

Let's explore how our data changes when we log transform it and visualize it using histograms.

Let's start by looking at our egg size, we can see that the data is skewed to the left, a log tranformation will be useful.

we can log transform the variable directly by adding log(). However, we could also create a new colum with our log transformed data uisng dplyr() and the mutate(log()) command. For now let's look at the first approach:

The log transformation really helps with the skewness of this variable!

Regresion Lines

The last thing we will look into is adding regression lines to our scatter plots, this will be also useful when we are visualizinng the data for the linear models (next section).

For this, we can build up from what we have learned in the previous sections: Let's start by plotting Body and egg size, we will add some color and transparency to the points.

Now, let's add a regression line:

If we don't specify the method ggplot uses the "loess" method. However there are more options of regression lines. You can look at the help for the `?geom_smooth()` command to learn more. For now, we will use just a simple "lm" method.

You can now see that we have a linear regression line and a shaded area that displays the confidence interval. However, you can remove this by specifying FALSE in the `se` command.

12:00 LUNCH

Linear Models

```
library(ggplot2) # plotting library
library(dplyr) # data management
```

Figure 3: LUNCH

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(lmtest)
```

```
## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
library(MuMIn)
library(MASS)
```

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

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

```
library(lme4)
```

```
## Loading required package: Matrix
```

1:30 Intro to linear models

PPT Slides

1:40 Basic linear models with lm()

Linear regression

Linear regression uses a continuous predictor

`Continuous Response ~ Continuous Predictor`

Question: Is egg size related to body size?

Recall that a key assumption for linear models is that our data follows a normal distribution. We saw in the visualizations tutorial that both egg size and body size seem to be log-normally distributed, so before we perform our analysis, we should transform our data.

The default output of the `lm` function is not too informative. It just shows us the input formula and then the estimated coefficients. We can use the `summary()` function to extract more information about the model

The **Estimate** column gives us the equation for the line of best fit, with the intercept and slope. Recall the linear equation:

$$y = mX + b$$

In the mathematics of linear models, this exact same equation is usually written as:

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$

In the output of our model, we see the intercept (β_0) is 2.215 and the slope (β_1) is 1.022 which give us the values of the predicted line. (ϵ_i) is the residual error which represents the difference between observed values in our data and predicted values from our model.

The **Pr(>|t|)** column represents a t-test on the slope and intercept under the null hypothesis that `slopThe = 0` and `intercept = mean(Y)`. We can see that both are significant

We can also see the **Adjusted R-squared** value, which is an estimate of the amount of variation explained by the statistical model, ranging from 0 (no variation explained) to 1 (100% explained).

The **F-statistic** tests the overall fit of the model by comparing variation. In this case you can see that our model is significant. In a linear regression, a t-test on the slope and an f-test are equivalent. We can see more detail about the F-test with the `anova()` function.

The data suggests a statistically significant relationship between the egg size of birds and body size of birds.

ANOVA

Question: Is body size different among taxonomic families?

Similar to linear regression we have an estimate column, and for each estimate we also have standard error, t-value and probability.

We also have R-squared and F-values describing the fit of the model, along with an overall p-value for the model.

BUT there is one important difference. Our predictor variable X_i is categorical in this model, rather than continuous. Therefore, the **(Intercept)** column is the mean of a reference category (usually alphabetically the first one- in this case, Calcariidae) and every other estimate is simply the deviation from the reference category mean. The **Pr(>|t|)** column tests whether each family mean is significantly difference than the reference family mean (Calcariidae). Additionally, you should be careful interpreting this column as it does not correct for additive Type I error rates. More on this later.

In this case, the p-value from your F-test tells you that at least one of these group means significantly differs from the others which is not necessarily that informative. We would need to do a post-hoc test that corrects for Type I error rate and tells us specifically what families differ from eachother.

1:50 Advanced linear models

What if we have 2 or more predictors?

We can simply add additional predictor X terms to our linear model

$$Y_i = \beta_0 + \beta_1 X_{1,i} + \beta_2 X_{2,i} + \beta_j X_{j,i} ... + \epsilon_i$$

Multiple regression

Question: Does body size of a bird differ with its egg size and duration of its song?

Multifactor ANOVA

Question: Is body size different between families or genus?

ANCOVA

ANCOVA is short for ANalysis of COVariance, meaning it is is an ANOVA (nominalpredictor *cat*) combined with a covariate (continuous predictor *con*).

$$Y_i = \beta_0 + \beta_{cat} X_{cat,i} + \beta_{con} X_{con,i} + \epsilon_i$$

Question: Is body size related to egg size or taxonomic family?

Interactions

Interaction terms in a linear model are sometimes referred to as *non-additive* effects, because they represent deviations from what would be expected under an additive model. Put simply, the response to one predictor variable changes based on another predictor variable.

Question: Is body size related to egg size? Does this relationship change depending on taxonomic family?

It is also possible to compute interactions between continuous variables and categorical variables although these can be tricky to visualize.

Take a minute to think about how hard it would be to add third variable with 3-way interactions to the example we covered above. It can get very tricky to interpret 3-way interactions, but here is a handy trick that you can use.

Consider the interaction term A:B:C with each letter representing a continuous or categorical variable. If there is at least one categorical variable, then a significant 3-way interaction tells you that you can run independent models. For example, let's say A represents three experimental groups. If there is a significant 3-way interaction, then you can just create separate data.frame objects for each group, and run simpler 2-way interaction models for each of the three groups, giving you 3 models in total. If you have more than one categorical variable, for example if A and B are both categorical, then you can choose to split *either* into different groups from A *or* B. It doesn't matter which, so you can just choose the one that makes the most sense to you.

2:00 QA/QC and model selection

QC Tools

Two key assumptions of our linear models are:

residual error follows a normal distribution with a mean of zero

AND

error variance is independent of predictors

There are a few convenient ways to check this with our original univariate linear regression `LinMod` and ANOVA `AOVMod`

This model should meet our assumptions, but let's check. There are three main ways to do this, all using the residuals of our model, which we can calculate manually, or use the `residuals` function on our model object.

Histogram This looks normal, and is centered around zero, but what is a normal distribution supposed to look like? We can use our `qnorm` function to generate the expected quantile of each observation. We could then plot the observed vs. expected values.

This is called the quantile-quantile or q-q plot

Q-Q Plot The Quantile-Quantile (Q-Q) Plot bins the residuals the same way we would for a histogram, but then calculates the number of observations we expect in each bin for a normal distribution. If the data are perfectly normal, then all the points will fall along the 1:1 line

We can see some very slight deviation from the line at the extreme values, where we have a small sample size

Homoskedasticity The last test is to make sure that variance is constant across our predictors. We can do this for each predictor variable alone, as well as the overall prediction from the model. To do this, let's add residuals to the dataset:

Note how the variance is similar range, centred at zero across all values of x in both graphs. We should not see any correlation between the x- and y- axes, or any large change in variance.

LRT

The likelihood ratio test gets its name from the statistic:

$$-2\ln(\frac{\theta_0}{\theta_1})$$

Where θ is the likelihood of a model, and the subscripts 0 and 1 are for two nested models. The model with **less parameters** goes in the numerator, and the model with **more parameters** goes in the **denominator**

The likelihood of the model (θ) is quite complicated, but for now just understand the model: Know that you can calculate the likelihood of any model by taking the natural log of the ratio of the two likelihoods (one from each model), and then multiplying by negative 2 gives you the **likelihood ratio**. You can test this value against a χ^2 distribution, with degrees of freedom equal to the difference in the degrees of freedom (number of predictors).

Let's consider our ANCOVA again. Does the interaction model fit the data better than the additive model?

AICc

The most common information criterion is the **Akike Information Criterion** or **AIC**. The mathematical definition of **AIC** is:

$$AIC = 2k - \ln(\theta)$$

where k is the number of parameters in the model, and $ln(\theta)$ is the natural log of the likelihood of the model. Yes, the same likelihood as the Likelihood Ratio Test (LRT)!

The 'best' model is the one with the **lowest** AIC

Another common and important criterion is AICc. This is just the AIC adjusted for small sample size:

$$AICc = AIC + \frac{2k^2 + 2k}{n - k - 1}$$

Where n is the sample size. As n increases, the denominator of the second term moves toward zero. So as sample size increases, AICc approaches AIC. Note also that k is in the denominator too, so we need larger sample sizes when including more complicated models.

2:10 Generalized linear models with glm()

Linear models are run in R using the `lm()` function, as we saw detailed in the *Linear Models Tutorial*. In that tutorial, we also saw a wide range of 'classical' statistical models that differ only in the type of predictor variables (e.g. ANOVA for categorical and regression for continuous).

We also learned how to inspect our residuals to make sure that they don't violate model assumptions. One of the most important is that the error term follows a normal distribution with a mean of zero.

Generalized Linear Models are very similar in the way they are programmed, except that they use the `glm()` function in R, rather than `lm()`, and they can be used when our residuals DO NOT follow a normal distribution. If you understand how `lm` works, then `glm` will be very easy to understand. If you are still struggling with `lm` then it's a good idea to go back and review/practice before continuing.

All **Linear Models** are **Generalized Linear Models** with a normal error distribution, but additionally we can use other error distributions like the *Poisson*, and the *binomial*. We specify the error distribution with the `family=` parameter in `glm()`.

Contrast with lm()

It should be very similar, but you'll notice that there is no p-value for the overall model. But, there is an AIC that we can use for model selection, or we can do the likelihood ratio test.

Now what if our residuals are not from a random normal distribution?

Poisson distributions

Poisson variables are common for count data (e.g. N offspring) but also can apply to log-normally distributed data with continuous measurements (e.g. biomass).

Question: Is number of eggs related to average egg size? Does this change among taxonmic family?

Logistic distributions

A **Logistic Regression** is similar to a linear regression in `lm()` except that the response variable is binary rather than continuous. This means that values of the response variable must be either 0 or 1. This is also known as a Bernoulli variable.

Question: Is overwinter survival related to average body size? Does this change among taxonmic family?

2:20 Linear mixed effects models with lmer()

Too many parameters is one problem. Another problem is the fact that we don't have a balanced sample (compare N of individuals for each family, above). When our samples are imbalanced, we can get big standard errors on our estimates, making them unreliable for families with smaller sample sizes.

A third problem would arise if families differ in their variability. In our example data, we ask about the relationship between body size and egg size, but if we look at the variability in the relationship among individuals, some families may a more variable response, and some have a less variable response. This would be heteroskedasticity – another violation of linear models.

Finally, there is a conceptual problem: We aren't really interested in these specific families. Instead, we might assume that we randomly sampled these from a much large population of bird families. The above models don't account for the random sampling of genetic families, only the residual error among randomly sampled individuals.

Random Effects

A random effect assume that the levels of a predictor are chosen at random from a larger population. Just as we sampled individual observations from a larger population in the *Distributions Chapter* we can sample genetic families from a larger population of genetic families, and we can sample individuals within a family from a larger population of individuals from the same family.

That's where the term 'random effects' comes from: we are assuming that the different levels of a random effect are a random, unbiased sample from a larger population of effects.

Instead of estimating each individual mean for a random effect, we assume the effects are drawn from a random distribution. As a result, we estimate a single term – the variance – rather than a separate term for the effect of each family.

Linear Mixed Model

The **Linear Mixed Model (LMM)**, aka the **Linear Mixed Effects (LME)** model is just a Linear Model (LM) with the addition of random effects.

Mixed Model

Question: Is body size related to egg size?

No p-value! How do we tell if predictors are significant?

Accounting for the random variation between Genus groups does improve the fit of this model!

2:30 - Break

Figure 4: BREAK

Multivariate analysis

3:00 Into to machine learning

Why Use Machine Learning in Biology? The machine learning models covered in this book are the ones most relevant to biologists. As biologists, it can be helpful to start by asking What do I want to do with my data? Machine learning methods may be particularly useful if your goals fall into one or more of these three classes:

1. **Compress:** You want to simplify complex data to make it easier to interpret.
2. **Cluster:** You want to group subjects based on observations that they share.
3. **Classify:** You want to classify subjects based on observations about them.

Compression **Compression** is a way to simplify data. You may be familiar with compression of large jpeg images, mp3 audio files, or mpeg video. **Dimension reduction** is one common and powerful approach to data compression with many applications in biology. A dimension is just technical jargon for a *measurement* or *observation*. Consider the following example:

1. You are an environmental researcher working with dozens to hundreds of climatic variables. Each variable could be a dimension of data.

The dimensionality of our data is be equal to the number of columns of observations in a `data.frame` or `tibble` object.

Dimension reduction methods attempt to simplify our data from many original dimensions to a few of the most informative dimensions. However, not all columns are appropriate for dimension reduction. Codes that you create for the experiment are typically not appropriate. These would include ID codes for study individuals, study sites, treatment, and other descriptive information.

Clustering Sometimes we are less interested in the specific columns of data or predictor variables in a statistical model. Rather, we want to know how our individual rows (i.e., subjects) are related to each other based on the observations they share. For example:

1. In population genetics we are often interested in how individual populations relate to each other based on shared alleles.
2. In community ecology, we might want to see which geographic locations have similar species assemblages.

Classification Classification is where machine learning really shines. For example in previous sections, we saw how to use the link function to transform a linear model equation function to a probability function in a logistic regression (GLM with a binary response variable.). This is a form of classification since our probability is a prediction about the likelihood that the subject belongs to the group encoded as 1 in the response variable. Machine learning models for classification expand on this idea to make more robust predictions.

We can use ML methods like cross validation to make predictions are robust to different error distributions, unlike the GLM framework, which requires a defined error distribution. Also, we can use models to classify subjects into multiple categories, rather than the simple binary categories needed for logistic regression.

Overview of Machine Learning A machine learning model is one that uses computation to optimize a model and draw inferences from data. It is a very broad area of research that has a lot of practical implications for biologists. In general, the more complex a machine learning model, the more observations are needed to build a robust model.

Discriminative models Machine learning tends to focus on discriminative models that emphasize prediction over parameter estimation. As a result, there tend to be fewer assumptions about the underlying distributions of the input data. Models that make no assumptions about the underlying population distribution are called distribution free models.

Supervised vs Unsupervised Machine learning models are often grouped into these two categories. The difference is related to the prediction, which is usually a grouping variable. A model is supervised when the grouping response variable is used to train the model. A good example of a supervised machine learning model is a linear discriminant analysis. A model is unsupervised when there is no grouping variable. A good example of an unsupervised machine learning model is a principal component analysis.

Bias vs Variance A key concept in machine learning is the trade off between the variance and bias of a model.

The bias of a model is the difference between the average predicted value of a model and the true value of the target variable. A model with a high bias will consistently miscategorize the data. The sources of bias are discussed in the Experimental Design chapter of R STATS Crash Course.

The variance of a model describes the uncertainty or inconsistency in a machine learning model. A machine learning model would have a high variance if, for example, organisms with similar height or length were predicted to have very different weights.

A more complex model can reduce bias but increase variance. A model that is overfit to the data would have a low variance but a high bias and the primary goal of machine learning models is to find the 'sweet spot' between bias and variance.

Overfitting Overfitting occurs when a model is really good at predicting the specific data used to generate the model, but then performs poorly when predicting new data.

Data Splitting When we use machine learning models to make predictions, we usually want to make broader predictions about a population. In the ideal scenario we would collect data, make or 'train' the model, then collect new data to test the model. If we cannot run two separate experiments, then we can try to simulate this by splitting the data into a training dataset and a validation dataset. We use the training dataset to train/make the model and the validation dataset to test the model.

Cross Validation (CV) Cross validation is a computational method related to data splitting, excpet that we start with the entire dataset, then:

1. remove one or more observations
2. train the model on the remaining data
3. put the data back in and remove the next data point(s)
4. repeat for the entire dataset
5. average the models

This is known as leave-one-out cross-validation or LOOCV

Too Many Predictors Generally, we want the number of predictors to be much less than the number of observations. But many biological datasets are the opposite, such as gene expression for thousands of transcripts measured in a few dozen individuals.

Even though we have more models than predictors, our model will perform better if we choose a few of the best predictors rather than throwing everything into the model.

Feature Selection A feature in machine learning lingo is just a predictor, so feature selection just means choosing predictors, and there are many approaches to automating this. The best way is to use your biological knowledge of the system to select features; however, we may need to automate this task. In a simple case, we can apply linear models and use the p-value as a cutoff to decide which features to include.

Dimension Reduction Dimensionality is just another fancy word for number of features. When 2 or more features are colinear or we expect some collinearity for biological reasons, then we can use dimensions reduction methods. Principal Component Analysis is a common and robust approach that is the basis for many machine learning models.

Scaling Scaling is important when our measurements are on different scales, which is true for most biological variables. Scaling usually involves two components that can vary across different features: mean and variance. Equations for scaling generally adjust for one of these.

The basic rule of thumb is to only use unscaled data when you want to incorporate differences in the mean and variance among your features.

The `normalize()` function in R us a useful took for scaling columns in a data frame. More advanced transformations are not covered here but are worth investigating for complex datasets.

3:15 Principal Components Analysis (PCA) with princomp()

Set Up

Be sure to set up R and R Studio. You should have these installed already. Please install the following packages: `dplyr` and `ggplot` and the custom plotting theme:

```
#source("http://bit.ly/theme_pub") # Set custom plotting theme
#theme_set(theme_pub())
```

This is the raw data but we should be using the CLEAN data for this

```
#MyData <- read.csv("AreBirdsRealData.csv", header = T)
```

Data Overview

We will continue to use the same data set used for all the previous sections. However, we will be using some new predictor variables, these variables are bird song traits which we will use to answer the following question:

1.
   - Collective answer -

Now let's explore our data really quickly to understand the traits we are working with:

for this section we will focus on the following variables:

1. **IndID**
2. **Site**
3. **Family**
4. **Genus**
5. **Species**

6. **Duration**
7. **RepeatLen**
8. **PitchRange**
9. **AmplitudeRange**
10. **Ramping**

In this case we can think of the song as a multivariate trait with $D = 5$ dimension (song characteristics).

Principal Component Analysis

Overview Principal Components Analysis (PCA) is an example of an unsupervised machine learning model. However, PCA has long been used by ecologists and evolutionary biologists as a dimension reduction tool. A common approach is to visualize a few principal components to look for structure in biological data. PCA also forms the basis for more discriminant analysis – a form of supervised machine learning.

PCA is one type of multivariate analysis and related methods are commonly used to analyze all kinds of biological data including plant and animal communities, microbiomes, gene expression, metabolomics & proteomics, and climate.

The **Principal Components** of a PCA are new vectors that are calculated as linear combinations of the original vectors of measurements – the columns in our data frame.

Think of a linear model where we try to predict a single response variable Y by adding together coefficients of N individual predictors $X_1...X_N$. PCA is similar, except that instead of predicting a measured variable Y, we are trying to redefine N new component axes $PC_1...PC_N$ that are:

1. Linear combinations of the input *features* $X_1...X_N$ (features = in LM lingo)
2. Uncorrelated with each other. The correlation coefficient between any two PC axes should be close to zero.

One common goal of PCA is to reduce the **dimensionality** of our data, by redefining many **correlated** predictors into one or a few **uncorrelated** principal component axes. These PC axes are themselves vectors of data representing different combinations of the input features.

Correlated Traits PCA works best when there is collinearity among the predictors. We can see this by looking at the pairwise correlations. To do this, let's first make a separate data set, one for each of the main measurements. We can use the `select()` function from `dplyr` and specify the columns we will use:

We can calculate a correlation matrix using `cor()`, to see how the features are correlated, we will omit the first 5 columns since these are categorical ID variables.

We can see some pretty high correlation coefficients. We can square these to get R-squared values, which we can interpret as the amount of variation in one metric that can be explained by the other. We might also want to round to make the values easier to compare

We can see there is quite a bit of variation year-to-year but we can predict 0.1-63% of the variation.

Visualizing these correlations is a bit tricky. Normally, we should do this to look for outliers and nonlinear relationships among variables, but for the sake of time we'll skip this step.

PCA Running PCA in R is very straight-forward. But there are a few important considerations if we were to deal with missing data and scaling.

Missing Data One major limitation of PCA is that it can't handle missing data. One approach is to remove any rows with missing observations. However, if even just one observation is missing for any column, then the entire row must be removed from the analysis. When there are many columns of data, this can significantly deplete our sample size, and compromise the interpretation of the PCA. Instead we can impute the data:

Imputation: We only have a few hundred rows of data so we want to avoid removing any observations that we have. Rather than delete entire rows, we can try to 'impute' the missing data using the following approaches:

1. One simple method is just to replace the `NA` with the mean, median, or mode of the column of data.
2. A more complicated method is to use some kind of predictive model like a linear model.

Scaling A typical next step in a machine learning pipeline would scale our *features* (i.e., *predictors*) using z-scores or another method. However, the PCA function we use in R has a built-in scaling function that we can use.

Now we are ready to run the PCA.

The princomp() Function The `princomp()` function in base R is all we need for a principal components analysis.

Scaling In `princomp()`, we can use the `cor=T` parameter to calculate principal components from the correlation matrix, rather than the covariance matrix. This is equivalent to standardizing to z-scores.

That's it! One simple line gives us a Principal Components Analysis. To start, examine at the structure of our princomp output.

We can see that the output is a *list* object, with some important components that we'll explore here.

Principal Components Here's where Principal Components Analysis gets its name: Every PCA analysis will return a number of principal component vectors (aka 'axes') equal to the number of input features. In this case, we have five columns of Song data. This results in four PC axes (Comp.1 to Comp.5), which are `scores` in the output of a `princomp` list object.

A key characteristic of PC axes is that they are **uncorrelated** with each other.

Here we see the correlation coefficients of the PC axes, rounded to 3 decimal places. Note that the diagonal is 1 representing the correlation of each PC axis with itself. The off-diagonals are zero, meaning that the correlation coefficient is < 0.001.

Compare this matrix to the correlation matrix for the original `Song` data, shown above. We have redefined linear combinations of the original data to take the original 5 correlated features and redefine them as 5 uncorrelated features.

In other words, we have rescaled four **correlated** measurements into four **uncorrelated** PC axes.

Eigenvalues Each of the five PC axes has an eigenvalue, which we can see in the summary:

The first row shows the `Standard Deviation` for each PC axis This is literally just the standard deviation of the values of each PC:

Compare this value to the `Standard deviation` for Comp.1 above.

Squaring the standard deviation gives the variance, which is also known as the PC eigenvalue:

The eigenvalue of a PC axis is just the amount of variation (variance) in the observed data that can be explained by the PC axis. Notice above that the `sd` declines from PC1 to PC5. This is always the case: PCs are sorted by their eigenvalues, from highest to lowest.

The second row shows the `Proportion of variance`, which is how much of the total variance (= sum of all eigenvalues) is represented by each Principal Component axis.

The third row is just the `Cumulative Variance`, calculated by summing the variances. So the first column is the same, the second column is the first + second, and so on.

If we are using PCA for dimension reduction, we can use the **Proportion of Variance** explained by each PC to help us decide how many PCs to keep for downstream analysis.

A common method for this is to just plot the variance across eigenvectors.

Scree Plot Looking back at the list items for SongPCA above:

The x-axis is the PC number, ranked by eigenvalue. The y-axis is the eigenvalue, representing the amount of variation explained.

This is called a **Scree Plot**. It can help us choose the number of principal components to keep in the analysis. We look at the shape and try to find the PC axes that account for most of the variance. Visually, we can see this as the change in slope.

In this case, we can see a big drop from 1 to 2 and then a much slower decline from 2 through 5. This is a good indication that the first eigenvector (PC1) captures most of the variation in Song characteristics. The above table tells us that it's more than half (Proportion of Variance = 54%).

Eigenvectors & Loadings Each principal component axis is just a vector that is calculated by a linear transformation of the input features. You should now have a clear idea of how we can define a linear model to predict `Y` from one or more `X` predictors, with an overall intercept and individual slopes/means for each `X`. The eigenvector is similar except that there is no intercept. Instead of a measured `Y` variable, we have five PC axes, each one a different linear combination of the original song columns. If we take these four coefficients and put them into a new vector, we get something called the **eigenvector**. In PCA lingo, the coefficients are called **loadings** and the **eigenvector** is just a vector of loadings.

Think of eigenvector loadings as a measure of how much a particular measurement 'loads' onto a given PC axis. Features with higher magnitude have a stronger influence on the PC and the sign gives the direction.

We can extract the first column of coefficients as the first 5 elements:

The PC axis is just calculated by multiplying each loading by its corresponding feature vector, and then summing them together. We can confirm by plotting. We also have to make sure we scale each measurement to z-scores since we used `cor=T` in the PCA. The `scale` function in R is good for this:

Remember that PC axes are just rescaled versions of the input data, so we can treat these as new variables, even adding them back to our dataset:

There are a lot of other things we could explore here. For example, we could use PC1 or PC2 as a predictor or response variable in a linear model.

Projection You may remember back to high school math or physics that a **vector** is defined by a **direction** and **magnitude**. We have already seen that the **eigenvalue** is the magnitude of an eigenvector and represents the amount of variation captured by a principal component axis.

So what is the direction of an eigenvector? It's a direction in multivariate space, with dimensions equal to the number of input features. Of course, more than 2 dimensions can't be visualized on a computer screen or report, and more than 3 dimensions is a completely foreign concept to the human brain. But we can still visualize multiple dimensions by using **projection**.

The 3D space is **projected** onto 2D space. We can do the same thing for higher dimensions of a PCA. Rather than do these calculations by hand, we can use the `autoplot` function from the `ggfortify` package

And use `loadings=TRUE` to project the eigenvectors to see which measurements contribute most to PC1 vs PC2

Compare the direction of these projected eigenvectors to the loadings in the output for SongPCA, above. We can see that PC1 is affected by PitchRange, Durantion, Ramping, and RepeatLen whereas component 2 is mostly affected by Amplitude Range and mildly by RepeatLen, Ramping and Duration.

QA/QC PCA has important assumptions that should be checked. As with linear models, we may want to transform some of the input variables to help meet model assumptions. The key assumptions is that input variables are **multivariat normal.** This means that if we graph each pair of input variables, they should form a 'shotgun' pattern that is approximately circular (or oval) with the highest density of points in the middle. As Principal Components are linear combinations of input variables, outliers and non-normal distributions tend to bias the loadings of particular PC axes.

Another important assumption is that the major axes of variation are the axes of interest. In the examples above, we saw how variation along PC1 is largely due to genetic variation while PC2 is largely due to environmental variation. But it may not always be like this. For example, we may have multiple axes of environmental variation and genetic variation might be spread across many axes. In such a case, we may want to rescale or 'rotate' our axes in a way that maximizes variation among populations. Although we can't do this with PCA, there are other methods based on PCA that we can try.

3:30 Linear Discriminant Analysis (LDA) with lda()

Linear Discriminant Analysis

With our bird data, we can run a Linear Discriminant Analysis to predict bird species based on the variables associated with their song such are duration and repeat length. The lda() function from the MASS package is pretty striaght forward. There are a couple of ways we can specify the model. First is to use an equation similar to a linear model (Y ~ .) where y is the column name of the categorical variable and . means 'all other columns of data'. We also need to specify the data parameter in lda()

```
library(MASS)

#MyData <- read.csv("./data/AreBirdsRealData.csv")
```

This chunk of code fits a model that will predict the family of the bird based on the duration, repeat length, pitch range, amplitude range, and ramping of their song. Now lets inspect the data using the str() function.

We can also use the summary function. However, unlike linear models the summary function for an LDA object summarizes the object itself, rather than the summary of the model.

Lets look at some of the summarized objects in more depth. The counts vector show the sample size of each group

Scaling shows the factor loadings, known as the LD eigenvectors. In an LDA we would have the number of observations - 1 as the number of LD eigenvectorss for all features. Contradictory to a PCA where we would have an axis for each feature. The loadings for LD1 show how each feature contributes to the LD1 axis, just like the PC eigenvector loadings. The difference is that LDA loadings are calcualted to distinguish groups, whereas PC loadings are calculated to explain variation across all the features.

Here we can see that the higher values of LD1 are determined by Ramping, and the lower values Duration, Repeat Length, Pitch Range, and Amplitude Range.

Predictors We can use the predict function with the LDA object to generate additional information, including the LDA equivalent of the PC scores.

The x object here is the predicted score for the LDA axis

The class object is the predicted category

Confusion Matrix We can generate a confusion matrix by comparing the predicted vs observed categories of each bird (i.e., row of data)

From this table we can calculate our model accuracy, which happens to be 99% with only one contradictory prediction

Posterior Probabilities The posterior object gives the posterior probability for each category. This is a concept in Bayesian statistics that is not covered in this workshop. For now, just know that these are probabilities for assigning each observation to each group. You can use it to measure the confidence of the model, with values close to 0% or 100% indicating high confidence of belonging to one group or the other.

3:45 Visualizions and interpretations

We can use what we have learned throughout this course to plot and interpret the PCA and LDA that we have constructed.

PCA Plot

Try to construct a PCA plot with the aesthetics you think will help you visualize your data better.

Here is a example:

Creating a color pallete:

```
mycolors <- scale_colour_manual( values=c("#555E7B", "#B7D968", "#B576AD", "#E04644", "#FDE47F",
                                          "#7CCCE5", "#452632", "#F0A830", "#C6BFB7", "#EBEBB2", "#4F74
```

Interpretation We can see that song traits are representative of the family groupings. We can cluster families based on their song traits. We can also see there is more overlap between certain families along PC1 and that the elipses overlap for some families. We can look back at our PC loading and see that pitch and duration might be the variables influencing that phenomenon. We could conclude that families like Paridae and Hurundinidae have quite distinct song traits from families like Icteridae and Cardinalidae. What other interpretations can you develop from this plot? ## LDA Plot

Interpretations

Beyond PCA and LDA

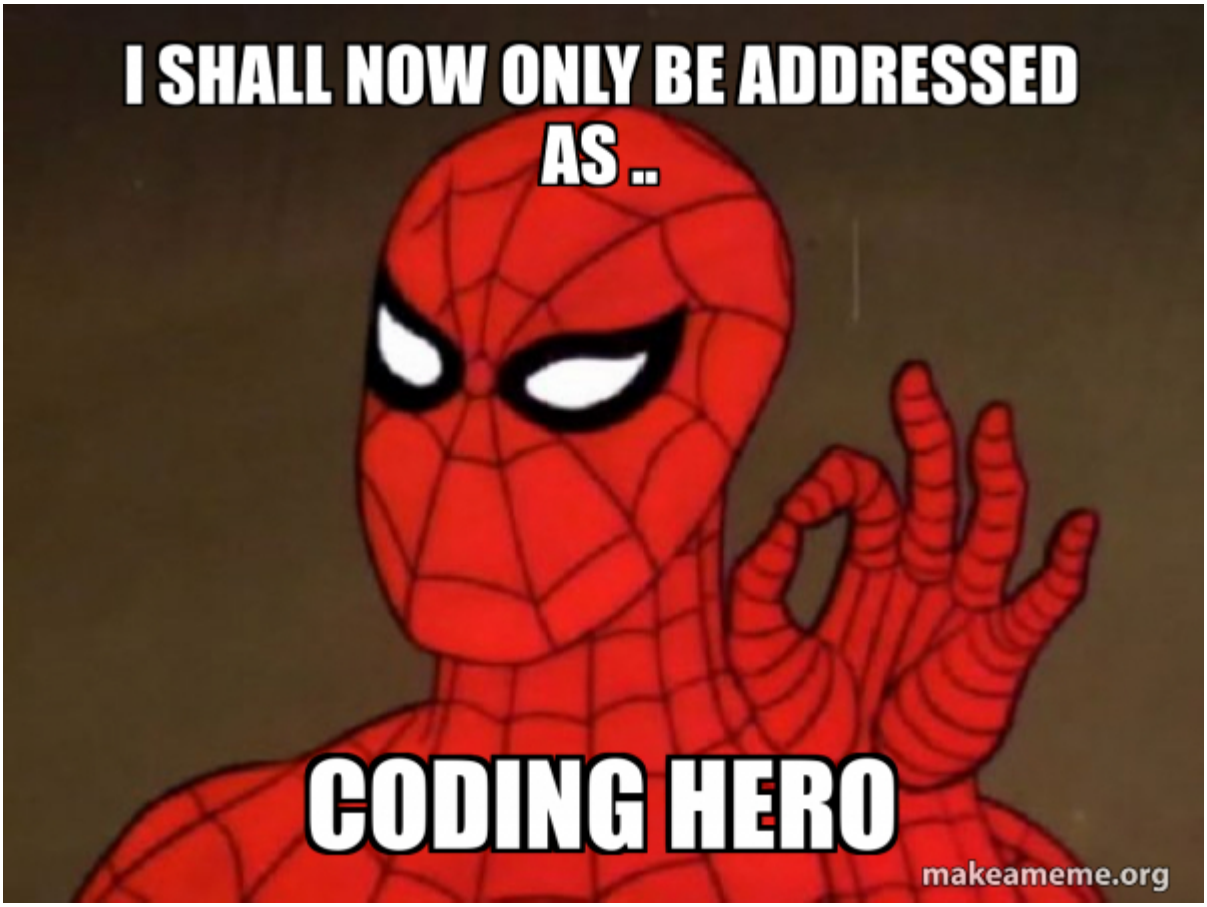
Here are a few examples of other multivariate analyses.

- **UMAP** – Uniform Manifold Approximation and Projection (UMAP) is a newer algorithm for dimension reduction that has gained in popularity. Although similar to PCA in the goal of dimension reduction, it allows for non-linear transformations of the data to try to group individuals (i.e., rows of data) based on their overall (dis)similarity. The `umap()` function from the `umap` package in R can be

used for this purpose, and the coding is very similar to PCA (e.g. `SongUMAP<-umap(Song)`). However, note that there is no scaling option, like the `cor=T` parameter in `princomp`. **You must scale your predictors manually (e.g. as z-scores) before applying the `umap()` function**. Be careful!

- **Factor Analysis** – FA builds on PCA by adding unobserved variables called factors. For example, we may have gene expression data from a set of particular pathways, and we want to use factor analysis to look at the behaviour of the smaller subset of pathways rather than the many genes themselves.
- **Correspondence Analysis** – Think of this as 2 separate PCAs that you then compare to figure out how one set of data affects the other. A good example of this is community ecology where one set of data is the species community and the other set includes a bunch of environmental variables.

Wrap-up



But remember. . . coding is a constant learning process and when things don't work out right away, using online resources as well as the R Crash Course books can help you troubleshoot!


WHEN YOUR R
CODE DOESN'T WORK
AND ALL YOU CAN DO IS
CONTEMPLATE YOUR EXISTENCE
imgflip.com