# Need to know:

Pipes: shift + cmd + m

# Lab 1 commands:

* Basic operations/ Arithmetic
* Median, mean, ? for help
  + **Note** for mean, be aware for NA. if you want to remove the NA’s
    - Mean(vect, na.rm = T)
* “<-“ gets

# Lab 2 commands:

* **OPENING LIBRARIES**:
* library(“tidyverse”)
* library("janitor")
* library("skimr")
* library("palmerpenguins")
* Concocted
* spring\_list\_data <- c(spring\_1, spring\_2, spring\_3, spring\_4, spring\_5, spring\_6, spring\_7, spring\_8)
* <- matrix(data, nrow, byrow?)
* Creates a simple data matrix
* Use “” for non objects
* Colnames
* Rownames
* rowMeans
* rowSums
* Cbind(data matrix, vector)
* Binds a new column
* Matrix[x,y]
* Shows value for x = row, and y = column

# Lab 3 commands:

* <- data.frame(data)
  + Creates data frame
* Dim(data)
* Summary(data)
* Nrow(data)
* Ncol(data)
* anyNA(data)
* table(data$variable)
  + gives how much of different things in a variable
* filter(data, variable with operation)
* to create vector from data frame:
  + data$variable

# Lab 4 commands:

* **TO READ A CSV FILE:**
  + <- readr::read\_csv(“data/name.csv”)
    - Remember to set right directory
* Glimpse(data)
  + Gives list of variables and their classes
* Summary(data)
  + Gives statistical data along with variables
* Class(data$variable)
* Data$variable <- as.factor(data$variable)
  + Want to do when there are two of the same “groups” within a variable
  + Ex. When you have like a ‘vore’ class and a list follows 1. Carniv 2. Herbiv. 3. Carniv 4. Carniv. 5. Herbiv.
    - So you can group them later on
* Levels(data$variable)
  + Displays what type of “groups” are in the variable
* Select(data, variable[s])
  + For columns
* Filter (data, variable with operation)
  + For rows
* Sort()
* Arrange()
  + Arrange(desc()) for descending

# Lab 5 commands

*With tidyverse*

* Names(data)
  + Names of variables, good for checking bad names
* Rename(data, [new name] = “old name”)
  + Janitor::clean\_names(data) [**MUCH BETTER**]
* Mutate\_if(class, new class)
* KNOW HOW TO PIPE
* Mutate\_all(tolower)
  + Lowercases everything
* Mutate(across(c(“var1”, “var2”), tolower))
  + Lowercases across the range of variables from var1 to var2
* **IF ELSE TO REMOVE -999 VALUES**
  + mammals %>%
    - select(genus, species, newborn) %>%
    - mutate(newborn\_new = ifelse(newborn == -999.00, NA, newborn)) %>%
    - arrange(newborn)
* General mutation
  + Data %>%
    - Mutate(new variable = var1 with operation)
* Tabyl(data, variable)
  + Gives # of cells associated with the groups in the variable
* Filter(!grepl(If a “string” is in any group, variable)
  + Ex. If you want to get any race associated with humans in a list of superheroes:
    - superhero\_info %>%
      * filter(!grepl("Human", race))
* If you want to get the all variables that satisfies a condition:
  + Ex. Trying to highlight only all of Doctor Doom’s powers:
    - superhero\_powers %>%
      * filter(hero\_names == "Doctor Doom") %>%
      * select\_if(all\_vars(.=="TRUE"))
  + This will only highlight only the cells that are “TRUE”

# Lab 6 commands

* Skim(data)
  + Gives a brief overview of the dataset
* Hist(data)
  + Gives histogram of data
* Summarize( new variable = operation on a variable)
  + Think like mutate where you can find the mean, min, max and total ‘n’ of observations
  + Summarise(n\_new variable = **n\_distinct(variable)**
    - Shows number of distinct observations and creates a new column
* Group\_by(variable)
  + Groups observations in a variable and you can connect it with summarize functions
* Count(variable, sort = T/F [T for descending order])
  + Counts how many observations within a variable.
  + Ex wanting to count how many penguins there are on each island.

# Lab 7 Commands

* %>% group\_by(“var1”, “var2”) %>% summarize(n\_var =n(), .groups = 'keep')
  + Get the number of observations by grouping two variables together.
* %>% summarise\_all(~(sum(is.na(.))))
  + Shows # of NAs across all variables
* Doing a count can also show *known* NAs
* %>% na\_if("-999")
  + Converts all “-999” values to NA
* naniar::miss\_var\_summary( )
  + remember to load in “+library("naniar")”

# Lab 8 Commands

* IS THE DATA TIDY????
* pivot\_longer (“var that you want to pivot / not pivot”, names\_to = “new column name”, values\_to = “values moved to the column”)
  + Names\_to can also be concatenated ( c(“var1”, “var2” …) ) creating two separate columns.
  + for the first part, you can also use “cols = starts\_with(“”)” to select all the variables with the string you entered.
    - Also, names\_prefix = “” , will select values with that prefix and delete them
    - Names\_sep = “” will remove a
  + Values\_drop\_na will drop NAs
* Graphical user interface, text, application, email

  Description automatically generated ‘**Transform’ will basically separate values for which have been put into one cell. Then also use ‘unnest’ so the values show up**
* %>% as\_tibble()
  + converts data matrix into a data frame
* Mutate(var\_name = rownames(data\_matrix))
  + Creates a new column with the row names as values under the new column
* %>% separate (var1, *into* = c(“col\_1”, “col\_2”), sep = “any thing that separates the values like an underscore ‘\_’ or even a space ‘ ‘ “ )
  + Separate values in a column into two columns

**Before**

Graphical user interface, text, application

Description automatically generated

**After**

Graphical user interface

Description automatically generated

* %>% unite(new\_var\_1, “col\_1”, “col\_2”, sep=”\_”)
  + Basically the opposite of separate
* %>% pivot\_wider(names\_from = “col\_1”, #observations under this col will be new columns

values\_from = “col\_2” #select the observations under a column to be the values under the new columns.

Table

Description automatically generated



**After**

Table

Description automatically generated



# Lab 9 Commands

**Data visualization**:

* Basic structure:
  + ggplot(aes())+geom\_x()
    - **remember to add the “+”**
* geom\_point()
  + Scatterplot
  + Can add na.rm
* geom\_jitter()
  + Similar to point, but gives random noise to data to separate some of the points when overlapping
  + Can add na.rm
* +geom\_smooth(method = lm, se = T)
  + Gives regression
* Geom\_bar()
  + Good bar graph for counts (**DOESN’T COUNT Y-AXIS IN THE AESTHETICS**)
    - Doing geom\_bar(stat=”identity”) is literally the same as geom\_col()
* Geom\_col()
  + Allows to specify x and y-axis.
  + Coord\_flip() changes axis
* You can do a filter before setting ggplot so you can graph a certain data
* Geom\_boxplot()

# Lab 10 commands

* life\_history <- read\_csv("data/mammal\_lifehistories\_v2.csv", na="-999") %>% clean\_names()
* homerange <- read\_csv(here("data", "Tamburelloetal\_HomeRangeDatabase.csv"), na = c("", "NA", "\\"))
  + **ONLY DO THIS IF YOU KNOW THE DATA!!**
  + Changes -999 to NA and cleans the variable names
* options(scipen=999)
  + cancels scientific notation for the session
* scale\_y\_log10()
* labs(title = “”, x = “”, y = “”)
  + setting a axis = NULL, removes the axis name
* theme(plot.title=element\_text(size=rel(1.25), hjust=0.5))
  + Changes the size and location of the Title
* Aes(x = “”, fill = “”)
  + ‘Fill’ is a common grouping option and automatically generates color
* Aes(x=””, size = “”)
  + ‘Size’ adjust the size of points relative to a continuous variable
* **One useful trick is to store the plot as a new ‘object’ and then experiment with geom’s and aesthetics**
* Scatterplot mods
  + geom\_point(size = )
    - Changes dot sizes
  + Aes(shape = “var\_1”, color = “var\_1)
    - Changes the shape and color of the points respectively
* geom\_bar(position = "dodge")
  + the bars become side-by-side
* geom\_bar(position = position\_fill()) + scale\_y\_continuous(labels = scales::percent)
  + Scales the bars (using two categories) to be shown percentages / proportions

# Lab 11 Commands

* Note: You might want to mutate a certain variable to be a factor (such as year) so that they are represented as each axis on a plot
  + For example:

The variable ‘year’ was not mutated as a factor:

Chart, line chart

Description automatically generated

However, this plot has ‘year’ as a factor:

Chart, line chart

Description automatically generated

Recall you can mutate to factor using:

* + mutate(var\_1 = as\_factor(var\_1))
* theme(axis.text.x = element\_text(angle = 60, hjust = 1))
  + Adds aesthetic to texts
* Geom\_line()
  + Great for showing changes over time
  + Useful when combined with +geom\_point()
    - Geom\_point(shape = ‘x’) changes the shape of the point
* Geom\_histogram()
  + Great for showing distributions
  + Geom\_histogram(bins = ‘x’) allows to change # of bins
    - (color = ‘ ‘ ) changes the **outline** of the bins
    - (fill = ‘ ‘) changes the **interior color** of the bins
* Geom\_density()
  + Shows the area under a curve using a smoothing function
  + Great with pairing with histograms
  + Has ‘color’ and ‘fill’ feature similar to geom\_histogram()
* grDevices::colors()
  + check colors
* library(gtools)
  + quartiles <- quantcut(homerange$log10.hra)
    - table(quartiles)
  + Shows quartiles when you want to create a ‘range\_category’
* theme(legend.position = "bottom")
  + sets legend to the bottom of the graph
* library(ggthemes)
  + ls("package:ggthemes")[grepl("theme\_", ls("package:ggthemes"))]
    - Stolen themes
* ?RColorBrewer
  + Shows palettes *at the bottom*
* +`scale\_colour\_brewer()` is for points
* +`scale\_fill\_brewer()` is for fills
* colors <- paletteer::palettes\_d\_names
  + my\_palette <- paletteer\_d("palettetown::charizard")
    - barplot(rep(1,14), axes=FALSE, col=my\_palette)
      * p+scale\_fill\_manual(values=my\_palette)
* +xlim(0, 4) + ylim(1, 6)
  + Adjust x and y limits
* facet\_wrap(~migratory\_strategy, ncol=6) [left – right]
  + Makes a ribbon of panels
  + Ncol determines how many columns in a row
  + Note where the ‘~’ determines if creating a top-down or left to right
* facet\_grid(migratory\_strategy~.) [ top -down]
  + Allows control over the faceted variable; arranged in rows or columns
  + A~B makes a comparison of two categorical variables (A = rows, B = col)