```r
#CODE Workshop Introduction to R
#Instructor: Molly Simonis
#Dates: 2 Aug 2023 and 3 Aug 2023
#Times: 1 pm – 3:30 pm Central Time
#Location: Online (Zoom link TBD)


#If you have made it this far in R and RStudio, you are ready to
#code!

#To start, in R Scripts (like this file), any text will have a #
#in front of the line. This is read by R as text only (no function).
#However, without a # in front of the line, R reads that as code.

#Day 1
#R as a calculator

#In its most basic form, R can be used as a calculator

#Type 2 + 2 into the console and hit Enter/Return
#Type 15 / 4 into the console and hit Enter/Return
#Notice how outputs are presented in the console



#Objects, assignment arrow and Global Environment

#We can also save these outputs as objects
#In the console, type in weight_kg <- 55 OR
#Execute the command by placing your cursor on line 22,
#and hit Ctr+Enter (or for Macs, Cmd + Return)

weight_kg <- 55

#Notice that we have now assigned (using <-, or the assignment arrow)
#the number 55 as an object called weight_kg. This object is now
visible
#in your Global Environment panel

#Objects can be given almost any name such as x, current_temperature,
#or subject_id. Here are some further guidelines on naming objects:
#
#1. You want your object names to be explicit and not too long.
#2. They cannot start with a number (2x is not valid, but x2 is).
#3. R is case sensitive, so for example, weight_kg is different from
Weight_kg.
#4. There are some names that cannot be used because they are the
names
#   of fundamental functions in R (e.g., if, else, for, see here for a
complete list).
```

```
#   In general, even if it’s allowed, it’s best to not use other
function names
#   (e.g., c, T, mean, data, df, weights). If in doubt, check the help
to see if
#   the name is already in use.
#5. It’s best to avoid dots (.) within names. Many function names in R
itself have
#   them and dots also have a special meaning (methods) in R and other
programming
#   languages. To avoid confusion, don’t include dots in names.
#6. It is recommended to use nouns for object names and verbs for
function names.


#You'll notice that there is no output in the console when you
assigned
#55 to your weight_kg object. This is because the number 55 is now
stored
#within your object weight_kg.
#Type your object name (weight_kg) into the console and hit Enter or
Return
  #As you are typing your object name in the console, notice that R is
smart
  #and remembers the object you created. Let R be smart to avoid
mistakes
  #typing object names and functions. As you start to type the object
name,
  #use the arrows keys on your keyboard to chose the correct object
you want
  #to call and hit Tab to select, OR use your mouse to point and click


#You can also perform calculations with your objects
#Convert weight_kg to lbs

weight_kg * 2.2

#How might you save this value for weight in lbs?
#Use your object to create another object

weight_lbs <- weight_kg * 2.2



#Functions in R

#R also has tons of basic functions
sqrt(10)

round(3.14159)
```

```
#Arguments within functions in R

#Many of this functions also have arguments you can change within them
#Type round(3.14159, into the console and hit the Tab key
#You'll notice a little note pop up that provides the organization of
#arguments available within the function.
#When we first used the round() function, we provided what 'x' is (the
number)



#Using help files in R

#Lets check out what the 'digits' argument is for using the help file
#for the round() function
?round()

#Notice a help file for the function pops up in the Help tab of the
bottom
#right panel in RStudio
#Using the information in the help file, what is the digits argument
used for?

#Rewrite this code to round pi to 2 decimal places on your own




#Vectors and Data Types
#Vector creation

#Objects in R can store multiple values.
#A vector is the most common data type in R, and it contains a series
of values
#To create a vector, we use the c() function, which stands for
'combine'
#Lets make a vector with multiple weights in grams
weight_g<- c(50, 60, 65, 82)
weight_g
#Notice now, the output has multiple values stored in the object

#You can also create a vector with characters
#characters will always be represented between quotations in R
animals<- c("mouse", "rat", "dog")
animals

#You can also add values to your vectors using the vector you just
```

```
created
weight_g<- c(30, weight_g, 90)
weight_g

#Make your own additions to the 'animals' vector





#Exploring properties of data in R

#Lets explore the properties of our vectors we created
#How long are our vectors? Use the function length()
length(weight_g)
length(animals)

#What kind of object are our vectors? Use the function class()
class(weight_g)
class(animals)

#What is the structure of our vectors? Use the function str()
str(weight_g)
str(animals)
#The structure of your data can also be found in your Global
Environment

#What kind of objects will the following vectors be? Does this
surprise you?
#Hint-- use the function class() to determine the data type of the
objects
num_char <- c(1, 2, 3, "a")
num_logical <- c(1, 2, 3, TRUE)
char_logical <- c("a", "b", "c", TRUE)
tricky <- c(1, 2, 3, "4")





#Take a break




#Subsetting vectors

#What if we want to know the 2nd value in one of our vectors?
#For that we use subsetting.
#There are multiple different ways to subset in R, but the most basic
```

```
form
#uses brackets
#Check out your full 'animals' vector output
animals
#Now call for only the 2nd animal listed in the vector
animals[2]

#Can you call for different values in other vectors by subsetting with
brackets?

#You can also create another vector by calling for specific values one
of your
#vectors you already made
more_animals <- animals[c(1, 2, 3, 2, 1, 4)]
more_animals




#Conditional subsetting

#You can also determine values within a vector based on conditions
#start with the following vector
weight_g <- c(21, 34, 39, 54, 55)
#How might you ask R to tell you which values greater than 50 in your
'weight_g' vector?

#Is this output different than what you expect?
#How would you interpret this output?

#In order to get the actual values within the vector that are greater
than 50,
#we have to create a conditional subsetting arguement that indicates
we want
#the values and not the logicals
weight_g[weight_g > 50]

#You can also you boolean symbols to make conditional subsets
#Use & for AND, | for OR
weight_g[weight_g > 30 & weight_g < 50]

weight_g[weight_g <= 30 | weight_g == 55]
#Note that here, > for “greater than”, < stands for “less than”,
#<= for “less than or equal to”, and == for “equal to”.
#The double equal sign == is a test for numerical equality between the
left and
#right hand sides, and should not be confused with the single = sign
#(which performs variable assignment similar to <-).
```

```
#What if we want to search for multiple terms in a vector that doesn't
get as messy
#and isn't as repetative as using & and |?
#The function %in% allows you to test if any of the elements of a
search vector are found:
animals <- c("mouse", "rat", "dog", "cat", "cat")

animals %in% c("rat", "cat", "dog", "duck", "goat", "bird", "fish")
#This code outputs logical responses (T/F) for if the animal values in
'animals' are
#also found in the other animal values you provided
#Can you interpret this output more specifically?





#Missing data

#A unique design in R is that it includes missing data in analyses
#Missing data in R is represented as NA, and most the time will be
filled in automatically
#if there are missing values when reading in objects (which we will
see in Day 2)
#To code a missing value, simply use NA
#When typing NA, let R be smart because NA is different type of value
than typing 'NA'
heights<- c(2, 4, 4, NA, 6)
heights

#Use the functions mean() and max() to determine the average height
across the values in
#the vector and the greatest height in the vector

#What happens when you try to perform functions with this vector?

#To get the functions to work, we need to ReMove the NAs from the
vector for the calculation
#We can use a common argument to do so that ReMoves NAs (na.rm = T)
#The typical default for R is to keep all the data you provide when
completing a function.
#Therefore, the typical default argument is na.rm = F, or not to
remove NAs
#But we can change that arguement within our mean() and max()
functions by changing our
#logicals to T or TRUE
mean(heights, na.rm = TRUE)
max(heights, na.rm = TRUE)

#Can you think of other ways remove NAs from the vector?
```

```r
#Using google, try to find R code online that lets you chose values in
a vector that are not NA
#Then, adapt that code to your heights vector and find the mean and
max heights







#Day 2

#Loading Data in R

#R can handle multiple different formats. The simplest formats
are .csv and .txt files
#Csv files are comma delimited files, which means each value is
separated by a comma
#Txt file are just text files, and each value is delimited by a tab

#Lets get started to day by loading data into R
#Using the 'Simonis_combined_df.csv' file in your, we will code to
read the csv file into R
read.csv('Simonis_R_Surveys_df.csv', header = T, sep = ",")

#Notice I have added arguments to indicate that the first row in our
data have names (header = T)
#and just in case, I've indicated that our values are separated by a
comma (sep = ",")

#You'll see in the console, we got an output of the data that maxed
out after ~75 rows.
#In order to use the data, we need to make it an object

#How would you assign the data you read in as an object called
surveys?
#Code it directly on line 286 of this R Script






#Inspecting data

#Great! Now that the data is loaded into R as a dataframe, inspect the
data using
#the str() function (to determine the structure of your data)
```

```
#and the head() function (to see the first few rows of the data)
#Directly code them here in this script in the lines below


#What types of data are each of your variables (e.g. are they numeric,
characters, etc?)

#Variables can take on many different classes of data.
#Yesterday we discussed a couple different classes of our vectors, but
here are a list of other common data types/classes

#1-D data classes
#numeric:       any real number(s)
#character:     strings or individual characters, quoted
#integer:       any integer(s)/whole numbers
#factor:        categorical/qualitative variables
#logical:       variables composed of TRUE or FALSE
#Date/POSIXct:  represents calendar dates and times

#2-D data classes:
#dataframe or tibble:   rows = observation, columns = variable
#list:                  holds multiple objects of multiple classes
#matrix:                array composed of rows and columns


#Back to the data
#Here are some data definitions for your variables:
#   Column                Description
#   record_id           Unique id for the observation
#   month                   month of observation
#   day               day of observation
#   year                    year of observation
#   plot_id               ID of a particular experimental plot of land
#   species_id        2-letter code
#   sex               sex of animal (“M”, “F”)
#   hindfoot_length       length of the hindfoot in mm
#   weight                weight of the animal in grams
#   genus                   genus of animal
#   species               species of animal
#   taxon                   e.g. Rodent, Reptile, Bird, Rabbit
#   plot_type           type of plot

#You can also directly view your table by clicking on it in your
Global Environment
#or by using the following code:
View(surveys)

#There are many other ways to inspect your data
#Use Google to determine other ways to inspect your data in R. What
codes did you find?
```

```
#Write those codes below






#Subsetting data

#We learned how to subset vectors in Day 1. We can use that same
concept to subset dataframes
#This time, since dataframes are 2-D, we will use a comma to separate
the rows and columns we want to call
#The format for subsetting with brackets is df[row, column]

#here calls the 1st row of the 1st column
surveys[1, 1]
#see what other row/column combos you can call. Code them in the lines
below






#We can also subset for multiple rows and columns at a time.
#If I wanted to call rows 1-3 of columns 5-6, I would use the code:
surveys[1:3, 5:6]
#where the : is means 'through'


#The $ operator can help us to get values from an entire column within
our dataframe
#In your console, let R be smart by begining to type 'surveys', and
select your 'surveys' dataframe object
#Then type the $ operator and hit tab
#chose a column name from the dataframe you'd like to call and hit
enter
surveys$species_id






#Factors in our data

#Factors are a special type of character class that indicate a
variable is categorical and qualitative
#In our 'surveys' data, we have multiple variables that should be
factors but are currently held in the
#character class
#Lets make sure our data is formatted correctly
```

```
#to make change a variable from character to factor classes, use the
function factor()
#and reassign that variable you want to change
surveys$sex<- factor(surveys$sex)

#recheck the structure of that variable now
str(surveys$sex)

#by default, R will always sort factors (and charaters) in
alphabetical order
levels(surveys$sex)

#To change that, you can use the factor() function with a 'levels = '
argument
surveys$sex<- factor(surveys$sex, levels = c('M', 'F', ''))
#This is important when you want a value or 'level' of your
qualitative data to represent
#a baseline for comparison. For example, check out the unique values
in your plot_type variable
unique(surveys$plot_type)
#What would we want our baseline plot type be for any comparative
analyses in the future?

#In the lines below, change your plot_type variable to a factor with
the baseline plot_type
#as the first level

#In the lines below, change all other character variables to factors
in 'surveys'
#There is no need to change levels in the rest of these variables



#Lets now take a quick look at how male and female animals are
distributed across our data
plot(surveys$sex)
#You should see a simple bar plot pop up in the Plots tab of your
bottom right panel in R studio
#What do you notice?

#There are multiple individuals captured within the dataset where sex
was undetermined upon capture
#Lets make that level more descriptive in our dataframe
#Remember when we re-leveled our factor so the levels were as follows:
1) 'M', 2) 'F' and 3) ''?
#We will use subsetting of those levels in order to rename our blank
values to 'undertermined'
levels(surveys$sex)[3]<- 'undertermined'
```

```
#recheck your levels
levels(surveys$sex)
#replot your quick bar plot on your own


#We have worked with data that we imported ourselves, investigated it
and updatted formatting where needed.
#But you can also make a dataframe without importing from a file
#On your own, create 3 vectors of the same length and make them into a
dataframe.
#Feel free to use Google to find code to adapt, or





#Take a break





#Using tidyverse for data manipulation

#Subsetting and transforming data using tidyverse

#Select and filter functions

#To start, we will need to install a package
#Packages in R have different functions that meet a particular topic.
There are thousands of packages
#you can use in R and they are relatively simple to install
#We will download the 'tidyr' and 'dplyr' packages
install.packages('tidyr')
install.packages('dplyr')
#Now, turn the package on
library(tidyr)
library(dplyr)

#Using these packages, we can subset and filter our data in different
ways that can sometimes be more intuitive
#Use the select function to chose specific variables you want to view
in your 'surveys' dataframe
select(surveys, plot_id, species_id, weight)

#You can also use select to chose not to show certain variables
select(surveys, -species_id, -record_id)

#Using the function filter() can give you entries in your dataframe
that match a specific condition
filter(surveys, year == 1995)
```

```
#Practice using the select and filter functions on your own to explore
their application







#Pipes

#What if you want to select and filter at the same time? So, from
THESE variables I only want values that meet THESE OTHER conditions.
#We can use pipes to solve this problem, which allow you to nest
functions
#Pipes are coded as %>%, and you can use the shortcut Ctr + Shift + M
(Windows) or Cmd + Shift + M (Mac)
surveys %>%
  filter(year == 1995) %>%
  select(species_id, weight)

#try on your own using pipes to subset the survey data to output
weights less than 5 and out for all variables except plot_id
#Type your code in the following lines







#Transforming data

#We can transform our data where needed by using the function mutate()
#Remember when we transformed values in our vector from kg to lbs in
Day 1? We can use that same concept in this function
#Lets transform the weight variable in our 'surveys' dataframe from g
to kg
surveys %>%
  mutate(weight_kg = weight / 1000)


#You can take this a step further by placing additional
transformations within the function to be performed in a proceeding
order
#by using a comma to separate those arguments
#Starting with the previous function, can you take our transformation
a step further by converting kg to lbs?
#Type your code in the lines below
```

```
#Thinking back to how we eliminated NAs in the past from our vectors,
how might you chose to eliminate NAs when subsetting?
#Hint--add in a filter() function
#Type your code in the lines below






#Now, what if you only want to visualize the first few rows of that
output? How might you add another pipe to get that outcome?
#Type your code in the lines below






#Sometimes it is important to summarize your data. For example, say
you want to provide a quick update to your colleague
#on the average weights for each known sex of animal you have
captured.
#To get them that answer, we can use the group_by() and summarize()
functions in-line with our pipes
surveys %>%
  group_by(sex) %>%  #perform functions by the levels in the sex
variable
  summarize(mean_weight = mean(weight, na.rm = T)) #create a new
vector for average weight for each sex without unknown values


#What if your colleague wants average weights by sex and taxa?
#You can add additional variables to group by-- just separate with a
comma
#Type your code in the lines below





#If your colleague wants a range of average weights by sex and taxa
(min and max), how might you update this code to include
#those values in your output?
#Type your code in the lines below
```

```
#If our colleage also wants sample sizes of our groupings, we can find
our n's by using the n() function within our summarize
#function. Add the n() function (no need to put anything into the
parantheses of this function) into your summarize function
#and have it called 'sample_size'
surveys %>%
  group_by(sex, taxa) %>%
  summarise(mean_weight = mean(weight, na.rm = T),
                                    max_weight = max(weight, na.rm =
T),
                                    min_weight = min(weight, na.rm =
T),
                                    sample_size = n())






#Take a break





#Data Visualization

#Plotting with ggplot2

#Now that we have taken a lot of time to subset, filter, sort and
transform our data, lets make some visuals!
#Data visualization is a really important step in any science or
science communication field because it should tell your
#audience everything they need to know

#First install ggplot2 and turn the package on
#Type your code in the lines below




#Similar to how we were subsetting using tidyr and dplyr, ggplot2
builds your visualizations and plots in layers
```

```
#In the first layer of code, we tell ggplot what data we are using and
what our axes are
ggplot(data = surveys, aes(x = weight, y = hindfoot_length)) #aes()
stands for aesthetics

#You can see that we have plotted our axes, but we need to add more
layers to actually visualize the data
#The layers we use to plot our data on the axes are known as geoms
#To visualize hindfoot_length ~ weight, lets make a scatter plot by
using geom_point
ggplot(data = surveys, aes(x = weight, y = hindfoot_length)) +
  geom_point()

#You can also use geom_jitter if you don't want the points to look so
uniform in a horizontal line
ggplot(data = surveys, aes(x = weight, y = hindfoot_length)) +
  geom_jitter()

#Those are some wild clusters of data we can see from the scatter
plot! I wonder if there is any sort of pattern we can
#pull out from those clusters. Let try to color the points by one of
our factor variables within the 'surveys' dataframe.
#To do so, we can add a 'color = ' argument into our aesthetics
ggplot(data = surveys, aes(x = weight, y = hindfoot_length, color =
taxa)) +
  geom_point()

#Oh yeah-- remembering back to our summary plots, we only had weights
for rodents. What other variable might parse out some
#of these clusters visually?
#Type your code in the lines below, changing the variable you color
your points by




#Now, using the ggplot2 cheat sheet and/or Google, find other ways you
can change the visualization of these points
#Hint-- look for arguments to place in the point layer (e.g. 'alpha =
', 'size = ', 'shape = ', etc.)
#Type your code in the lines below
```

```
#Boxplots are also a great way to visualize data when you have a
qualitative (factor) and quantitative variable.
#Boxplots provide visualization of the median of the data, data
quartiles and outliers

#First, since we know most of our completed data are for rodents only,
lets subset our 'surveys' dataframe for only rodents
#what are our unique taxa values?
surveys_rodents<- surveys %>%
  filter(taxa == 'Rodent')

#Lets also remove rows with NAs so visualization is easier for us
today.
surveys_rodents<- na.omit(surveys_rodents)

#Now, lets plot a boxplot of weight ~ species
ggplot(data = surveys_rodents, aes(x = species_id, y = weight)) +
  geom_boxplot()


#What if we want to visualize both the raw data points and the box
plots?
#Easy! We just add two geom layers. Here is my preference for how I
typically plot both of these layers
ggplot(data = surveys_rodents, aes(x = species_id, y = weight)) +
  geom_jitter(alpha = 0.25, color = 'darkgray')+
  geom_boxplot(alpha = 0.5, linewidth = 1) + #alpha is the
transparency of geom object
  theme_bw() + #remove the gray background
  labs(x = 'Species ID', y = 'Weight (g)')





#Explore other geoms and different ways to change visuals using
ggplot2
#You can use Google, the ggplot cheat sheet, etc to come up with your
own ways to visualize data
#Are there specific plots you want to make in the future?
#Ask questions and share with the group!
```