**Schoodic Institute at Acadia National Park**

**Program R Cheat Sheet**

# Table of contents

1. Keyboard shortcuts
2. Getting started
   1. Packages
   2. Reading data into R
   3. Assignment to the global environment
   4. Exploring data
3. Manipulating data
   1. The tidyverse
   2. select function
   3. filter function
   4. mutate function
   5. group\_by and summarize functions
   6. join functions
4. Data visualizations
   1. Tables
   2. Figures
5. Getting help and further learning

# 

# 

# 

# 

# Keyboard shortcuts

| **Description** | **Windows & Linux** | **Mac** |
| --- | --- | --- |
| Undo | Ctrl + z | Cmd + z |
| Redo | Ctrl + Shift + z | Cmd + Shift + z |
| Cut | Ctrl + x | Cmd + x |
| Copy | Ctrl + c | Cmd + c |
| Paste | Ctrl + v | Cmd + v |
| Comment out selection | Ctrl + Shift + c | Cmd + Shift + c |
| Run current line/selection | Ctrl + Enter | Cmd + Return |
| Insert assignment operator | Alt + - | Option + - |
| Insert pipe operator | Ctrl + Shift + m | Cmd + Shift + m |

# Getting starting

## Packages:

1. packages are groups of functions you will need to perform specialized actions
2. install.packages(“package name”)
   1. installs the package
3. library(package name)
   1. calls the package into use
   2. note the lack of quotations; they are not needed for this function

## Reading data into R:

1. read.csv(“file path/file name.csv”) - reads .csv files
   1. don’t forget the file extension at the end of the file name
   2. in base R
2. read\_excel(“file path/file name.xlsx”, sheet = 1) - reads excel files
   1. the sheet argument allows you to tell R which tab in your excel file you want
   2. you need the “readxl” package to use this function
3. read.delim(“file path/file name.txt”) - reads text files
   1. in base R

## Assignment to the global environment:

1. use the assignment operator (<-) to save an object to the environment window
   1. the arrow always points to the name you wish to give the object
   2. for example
      1. raw.data <- read.csv(“file path/file name.csv”)
      2. now this .csv data is read in and saved as a data frame called raw.data

## Exploring data:

1. names(raw.data)
   1. will tell you the names of all your columns
2. head(raw.data)
   1. outputs the first few rows of your data frame in the console window
3. tail(raw.data)
   1. returns the last few rows of your data frame
4. str(raw.data)
   1. shows the structure of the raw.data data frame
5. class(raw.data)
   1. gives the class of your data frame which usually is typically “data.frame”
   2. you can also you the $ operator to check the class of columns inside the data frame
      1. class(raw.data$first.column) might return something like “numeric”
   3. to learn more about the types of data in R click [here](https://www.tutorialspoint.com/r/r_data_types.htm)!

# Manipulating data

## The tidyverse:

1. the tidyverse is a group of packages that you need to install and use in your scripts
2. it is the basis for all our manipulation and future visualization of data
3. first we need to know about the pipes operator (%>%):
   1. this is how we can make a pipeline, taking the data frame and performing multiple different actions on it at once
   2. the pipe operator tells R to carry the last action forward to perform the next action
4. let’s get into the different types of function you will want to use from the tidyverse
5. always produces a tibble (a tidy R version of a table) until you save the object with the assignment operator

## select function:

1. data.frame %>% select(column.name)
   1. as you can see the data frame is piped into the select function which will now return a data frame with only the column that you wrote inside the function
2. if you wanted to select multiple columns:
   1. data.frame %>% select(column.name, other.column, third.column)
3. if you want to remove one column:
   1. data.frame %>% select(-column.name)
4. if you want to remove multiple columns:
   1. data.frame %>% select(-c(column.name, second.column))
   2. here the c() is a function that combines the columns into a list that should be removed

## filter function:

1. data.frame %>% filter(species == “setosa”)
   1. the filter function takes your data and returns a simplified version removing the stuff you don’t want
   2. species is the column, and “setosa” is the value in the column you want
   3. in this example, we would return a tibble with only the data associated with rows that have setosa in the species column
   4. note the required 2 equal signs to make this work
2. if you want to filter out the setosa species keeping all the other species data:
   1. data.frame %>% filter(species != “setosa”)
   2. the “!=” translates to “not equal to”
3. if you want to filter by multiple species:
   1. data.frame %>% filter(species == “setosa” | species == “versicolor”)
   2. the “|” symbol translates to “or” and enables you to list multiple filters
   3. now we would get a tibble with the setosa and versicolor species data and nothing else
4. if you wanted to filter by setosa and see what sepal.lengths are greater than 5:
   1. data.frame %>% filter(species == “setosa” & sepal.length >= 5)
   2. now we get only setosa data where the sepal.length column has a value of 5 of more
   3. we can use any combination of =, <, > symbols to filter

## mutate function:

1. mutate allows you to create new columns or change any column in your dataframe
2. this function should only be used when wanted to produce a column that is the same length (same number of rows) as the existing data frame
3. if you wanted to get means by species or something that would compress the data, you want to use the summarize() function
4. let’s change the class of a column with the wrong class
   1. say the species column is somehow incorrectly formatted as a logical and we want it to be of class character:
      1. data.frame %>% mutate(species = as.character(species))
   2. you can also do multiple mutates at once:
      1. data.frame %>% mutate(species = as.character(species), sepal.length = as.numeric(sepal.length)
   3. the first argument you write before the equal sign is the name of the column, so if you want to alter an existing column use the same name, and if you want to make a new column you name it something different.
5. if you want a new column that is sepal.length divided by sepal.width:
   1. data.frame %>% mutate(sepal.lw = sepal.length/sepal.width)

## group\_by and summarize functions (or summarise):

1. to summarize your data, you first want to tell R what you groups you want so summarize the data for
2. in the case of the iris data, we might be interested in getting the mean and median of sepal.length
3. to do this, we first need to tell R to group by the species:
   1. data.frame %>% group\_by(species)
   2. this groups by all unique values in species which is why it is important to keep data entry consistent or deal with inconsistencies prior to this step
   3. use the distinct function to check:
      1. data.frame %>% select(species) %>% distinct()
4. now we select the column of interest:
   1. data.frame %>% group\_by(species) %>% select(sepal.length)
   2. the species column is kept automatically because that’s our grouping variable
5. then calculate the mean and median:
   1. data.frame %>% group\_by(species) %>% select(sepal.length) %>% summarize(sl.mean = mean(sepal.length, na.rm = TRUE), sl.median = median(sepal.length, na.rm = TRUE))
   2. summarize uses the same general format as mutate where the sl.mean is the new column name and mean() is the function performed on sepal.length with an additional argument na.rm set to TRUE so that we remove NAs in the data
      1. if you didn’t add the na.rm argument then you would get NA instead of an actual number
6. this is a perfect example of why the pipes operator; with technically one line of code we can get a nice tibble with the mean and median for each species!

## join functions:

1. joins are different functions you can use to combine multiple data frames into one
2. if you want to join data frame “y” on data frame “x”:
   1. data.frame.x %>% left\_join(data.frame.y, by = “column1”)
   2. the “by =” argument specifies the column that both data frames share
   3. you need to have a column that is the same in both data frames you wish to join
3. if you want to merge the opposite way; data frame “x” joined on data frame “y”:
   1. data.frame.x %>% right\_join(data.frame.y, by = “column1”)
4. the right and left joins will automatically leave out data
   1. for example, if you are joining y onto x, R uses the x data frame as the basis for the join
   2. so, it will only take the matching data from y and paste it appropriately where it matches in data frame x
   3. therefore, if you have extra data in y that doesn’t match x, that data will not come through
5. if you want all data from both data frames that you are trying to join use a full\_join():
   1. data.frame.x %>% full\_join(data.frame.y, by = “column1”)
6. if you want to join two data frames that have the exact same column structure but contain different sets of data, you can use the bind\_rows() function
   1. data.frame.x %>% bind\_rows(data.frame.y)
   2. again, these two data frames must have the same columns
      1. i.e., both have the column “species” and “sepal.length”, but contain different sepal lengths for different species
      2. you can bind these rows into one data frame
   3. remember not only do the columns have to be the same type of data, they also need to have the exact same column names

# Data visualizations

## Tables:

1. once you have manipulated your data to a point at which you would like to make it a table in a report, you should save it to the global environment using the assignment operator (<-)
   1. using the mean and median example from earlier:
   2. sepal.stats.table <- data.frame %>% group\_by(species) %>% select(sepal.length) %>% summarize(sl.mean = mean(sepal.length, na.rm = TRUE), sl.median = median(sepal.length, na.rm = TRUE))
2. now that it is saved we can export this data frame to our outputs folder using the write.csv() function
   1. write.csv(sepal.stats.table, “outputs/sepal\_stats\_table.csv”, row.names = FALSE)
   2. sepal.stats.table is the data frame we want to export, next is the path and desired file name in quotations, and then the row.names argument set to FALSE stops the output table from having numbered row names in its own column
3. after exporting the table, open it in excel for final formatting before copying it into the report, presentation, manuscript, etc.

## Figures:

1. to create almost any figure you will need to produce, you should use the ggplot2 package
2. ggplot works very similarly to the piper operator but instead of the pipe (%>%) you use a “+” symbol
   1. for example let’s plot a simple box plot of the means for three iris species:
   2. read.csv("data/iris\_data\_clean.csv") %>% ggplot(aes(x=sci.name, y=sepal.length, fill=species)) + geom\_boxplot()
      1. this line of code reads in the data and creates a box plot with ggplot where scientific name is on the x-axis and the sepal lengths are on the y-axis
3. there are tons of specification and adjustments that can be made with ggplot functions, see the example script (3\_data\_viz) for more details
4. once a finalized plot has been created, use the ggsave() function to write it to the outputs folder
   1. ggsave(“outputs/iris\_boxplot.png”, height = 6, width = 8)
      1. here the first bit in quotations is the path and file name that we assign to this plot, as well as the file extension
         1. you should use .png whenever possible
      2. then the height and width arguments are where you can specify the actual size out the output in inches

# Getting help

1. if you ever are stuck and need help, turn to google and the numerous forums for R help:
   1. [Stack Overflow](https://stackoverflow.com/)
   2. [R-bloggers](https://www.r-bloggers.com/)
2. you can also type help() with the package name inside the parentheses in the console of RStudio for information on the package
   1. alternatively you can write “?function.name” in the console for any function you want to read more about
3. there’s a ton to learn in the R world of ecological data, here are some great resources:
   1. [Riffomonas Project's Code Club](https://www.youtube.com/playlist?list=PLmNrK_nkqBpKY3SZiivlIGvcLX-KHmfR8) - not really ecological but a great R resource
   2. [R Programming for Data Science](https://bookdown.org/rdpeng/rprogdatascience/)
   3. [Tidy Modeling with R](https://www.tmwr.org/)