

R Training 6

Rachel Kenny

6/23/2020

R Markdown

Loading packages & data

```
# Load packages
library(here)

## here() starts at /Users/rachelkenny/Documents/IRC/R Code/IRC_R_Training
library(tidyverse)

## -- Attaching packages ----- tidyverse 1.3.0 --
## v ggplot2 3.3.1      v purrr  0.3.4
## v tibble  3.0.1      v dplyr  1.0.0
## v tidyr   1.1.0      v stringr 1.4.0
## v readr   1.3.1      v forcats 0.5.0

## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
library(janitor)

##
## Attaching package: 'janitor'
##
## The following objects are masked from 'package:stats':
##
##   chisq.test, fisher.test
library(readxl)

# Load agua chinon veg data

ac_data_raw <- read_excel(here("data", "OCWR_AC_2019_Data.xlsx"))
ac_data <- clean_names(ac_data_raw)
ac_data$polygon_id[which(ac_data$polygon_id=="5M Buffer")] <- "5M BUFFER"

# Load arthropod data
arth_data <- read_csv(here("data", "IRC_restoration_arthropod_data.csv")) %>%
  clean_names()

## Parsed with column specification:
## cols(
```

```
##   Type = col_character(),
##   Management = col_character(),
##   year_restoration = col_double(),
##   Year = col_double(),
##   Transect = col_character(),
##   `Evenness - functional` = col_double(),
##   `Richness - taxa` = col_double(),
##   `Richness - ant` = col_double(),
##   detritivores = col_double(),
##   `herb chew` = col_double(),
##   `herb suck` = col_double(),
##   parasitoid = col_double(),
##   pollinator = col_double(),
##   predator = col_double(),
##   scavenger = col_double()
## )

# Load cougar example data
cougar <- read_csv(here("data", "example_cougars.csv")) %>%
  clean_names

## Parsed with column specification:
## cols(
##   Location = col_character(),
##   Camera = col_character(),
##   `2017` = col_double(),
##   `2018` = col_double(),
##   `2019` = col_double()
## )
```

REVIEW: Long and wide format data

It is sometimes necessary to rearrange the format of data in order to suit the needs of the analysis or data visualization.

Data is sometimes arranged in wide or long format. In wide format, categorical data is grouped. Wide data is often easier to read and interpret, but can sometimes lead to issues in conducting analyses or visualizing data in R. In long format (think long vertically), every row represents an observation belonging to a particular category.

Data is rearranged according to the key and the value. The key is an attribute and explains what the data describes. The value is the value of the observation described by the key.

Using the `tidyr` package, we can convert easily between long and wide format.

Spread

- `Spread()` is used to separate data from two columns into many, based on the number of categories. In other words, go from long to wide format. The key column is used to assign categories to the header of each column, and the values get arranged by category.
- `spread(data, key, value)`

Gather

- `Gather()` is used to take more than one column, and consolidate them into two columns based on the key column attributes. In other words, go from wide to long. The key column contains the categorical data, and the value column contains the corresponding values.
- `gather(data, key, value)`

Check out these links for more information: <https://github.com/rstudio/cheatsheets/blob/master/data-import.pdf> <https://garrettgman.github.io/tidying/>

REVIEW: Spread and gather example 1

```
# What kind of data is the cougar?
View(cougar)

# How can we convert the data format?
cougar_gtr <- gather(cougar, "year", "total_cougars", 3:5)

View(cougar_gtr)

# Now how can we convert it back to its original format?
cougar_spd <- spread(cougar_gtr, "year", "total_cougars")
View(cougar_spd)
```

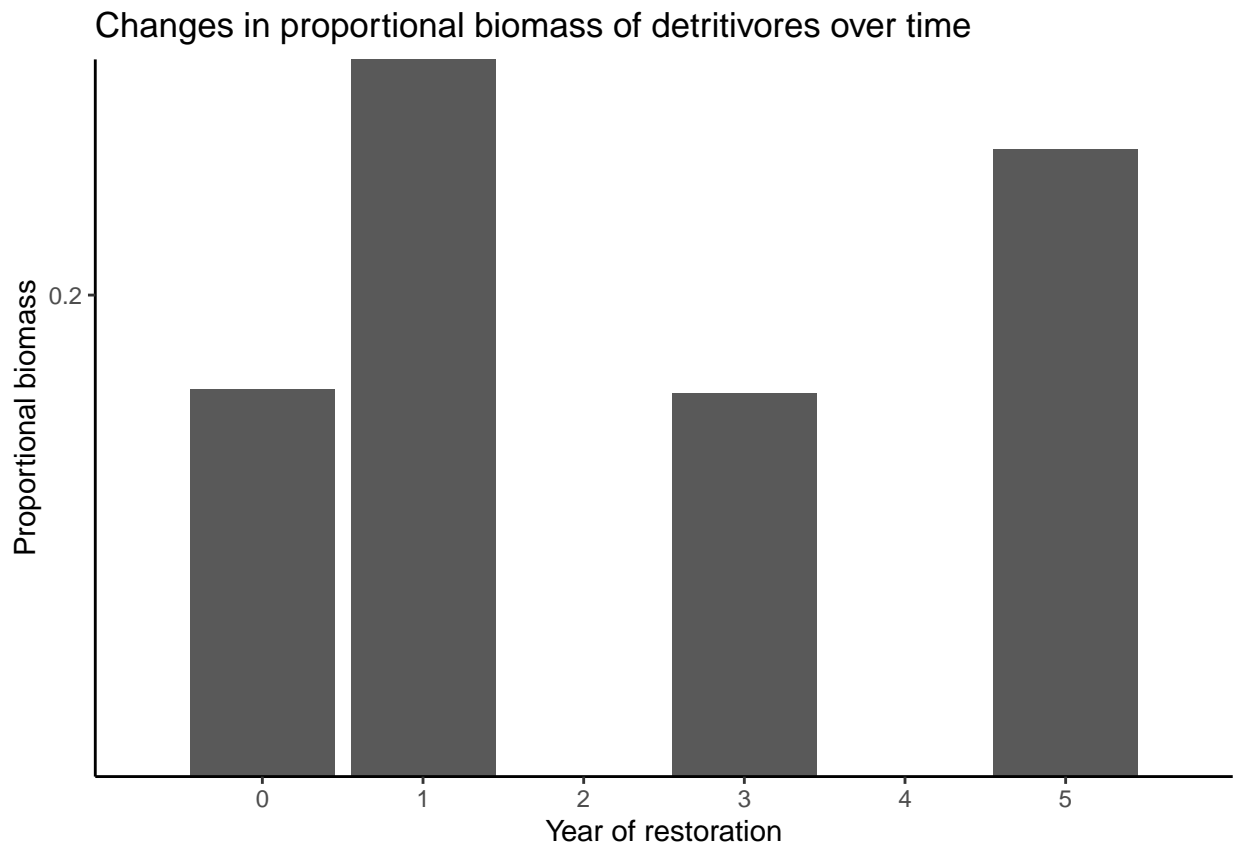
REVIEW: Spread and gather problem

```
# Problem - we want to create a plot showing the changes in proportional biomass of various functional
View(arth_data)

plot_det <- ggplot(arth_data) +
  geom_bar(aes(x = year_restoration, y = detritivores), stat = "summary", fun.y = "mean") +
  theme_classic() +
  ggtitle("Changes in proportional biomass of detritivores over time") +
  xlab("Year of restoration") +
  ylab("Proportional biomass") +
  scale_y_continuous(expand=c(0,0), breaks = c(.2,.4,.6,.8,1))+
  scale_x_continuous(expand=c(0.1,0), breaks = c(0,1,2,3,4,5))

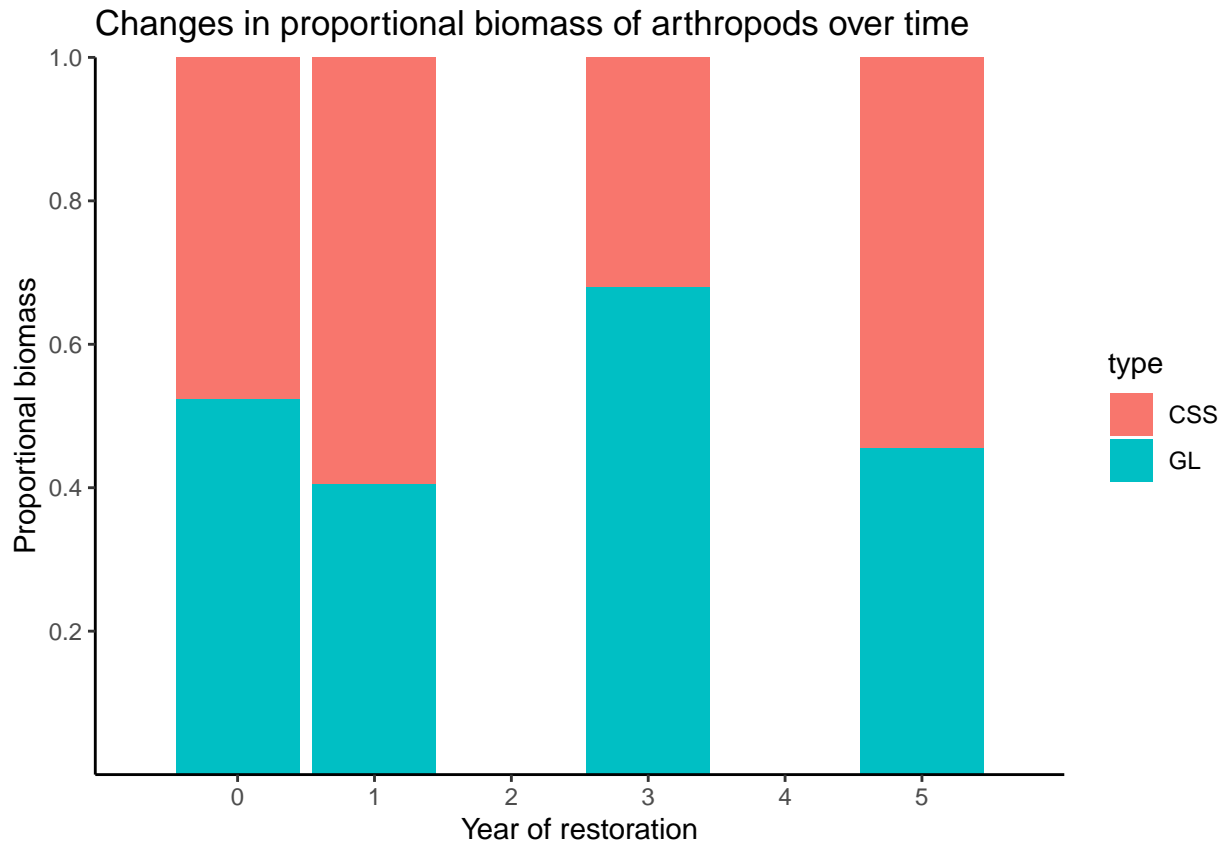
## Warning: Ignoring unknown parameters: fun.y
plot_det

## No summary function supplied, defaulting to `mean_se()`
```



```
# In order to make a ggplot with grouping by functional group, we would need one column with all the va
plot_det2 <- ggplot(arth_data) +
  geom_bar(aes(x = year_restoration, y = detritivores, fill = type), position = "fill", stat = "summary") +
  theme_classic() +
  ggtitle("Changes in proportional biomass of arthropods over time") +
  xlab("Year of restoration") +
  ylab("Proportional biomass") +
  scale_y_continuous(expand=c(0,0), breaks = c(.2,.4,.6,.8,1)) +
  scale_x_continuous(expand=c(0.1,0), breaks = c(0,1,2,3,4,5))

plot_det2
```



REVIEW: Spread and gather solution

```
# Let's select the relevant columns. I always retain the ID (transect in this case) and then we want to
arth_data_2 <- arth_data %>%
  select(transect, year_restoration, detritivores, herb_chew, herb_suck, parasitoid, pollinator, predator)
  gather("functional_group", "proportional_biomass", 3:9)

View(arth_data_2)

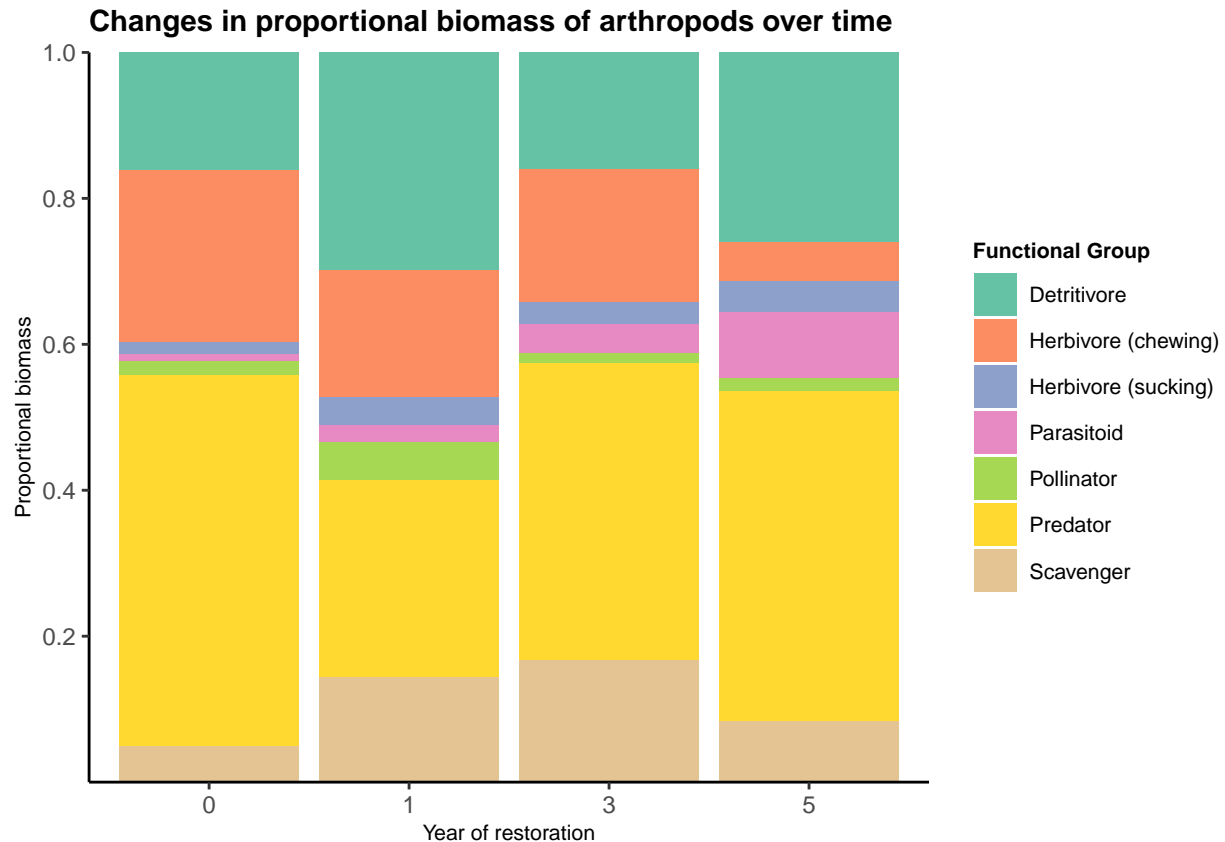
# Now we're ready to plot! Use your long form dataframe for the ggplot. Note, we want to treat restoration as a factor

arth_data_2$year_restoration <- as.factor(arth_data_2$year_restoration)

# Create a ggplot
plot_arth <- ggplot(arth_data_2) +
  geom_bar(aes(x = year_restoration, y = proportional_biomass, fill = functional_group), position = "fill") +
  theme_classic() +
  ggtitle("Changes in proportional biomass of arthropods over time") +
  xlab("Year of restoration") +
  ylab("Proportional biomass") +
  scale_y_continuous(expand=c(0,0), breaks = c(.2,.4,.6,.8,1)) +
  scale_x_discrete(expand=c(0.2,0)) +
  scale_fill_brewer("Functional Group", palette="Set2", labels=c("Detritivore", "Herbivore (chewing)", "Herbivore (sucking)", "Parasitoid", "Pollinator", "Predator")) +
  theme(plot.title = element_text(face="bold", size = 11),
```

```
axis.title = element_text(size=8),
legend.title = element_text(size=8, face="bold"),
legend.text = element_text(size=8))
```

plot_arth



REVIEW: Aggregate function

The aggregate function can help aggregate data according to grouping. In the example below, we want to

```
View(cougar)
```

```
cougar2 <- aggregate(cougar[3:5], list(cougar$location), mean)
```

```
View(cougar2)
```

Understanding the mitigation vegetation code - calculating absolute & relative cover

```
# Retain only the point-intercept data ("T.PI") and the columns for polygon ID, transect, pin number, a
ac_data_cvr <- ac_data %>%
  dplyr::filter(data_type == "T.PI") %>%
```

```

select(polygon_id, transect, pin_number, native_non_native)

cover2 <- distinct(ac_data_cvr) %>%
  count(polygon_id, transect, native_non_native) %>%
  spread(native_non_native, n)

View(cover2)

#Reorder columns so that non-native comes before no native veg, and set any NA values equal to zero
cover2 <- cover2[,c(1,2,3,5,4)]
cover2[is.na(cover2)] = 0

View(cover2)

# For polygons with multiple transects, average the values per polygon id, and then round all numbers
cover3 <- aggregate(cover2[3:5],list(cover2$polygon_id), mean)
View(cover3)

colnames(cover3) <- c("polygon_id", "avg_native_count", "avg_non_native_count", "avg_no_native_count")
cover3[,-1] <- round(cover3[,-1],1)

View(cover3)

# ABSOLUTE COVER
# Multiply values by 100 and divide by 50 because there are 50 sample points (pins) per transect
cover3_table <- mutate(cover3, absolute_cover_native = avg_native_count*100/50)
cover3_table <- mutate(cover3_table, absolute_cover_non_native = avg_non_native_count*100/50)
cover3_table <- mutate(cover3_table, absolute_cover_no_native = avg_no_native_count*100/50)

View(cover3_table)

# RELATIVE COVER
# To get relative native cover, take absolute native cover value and divide by absolute value native +
cover3_table <- mutate(cover3_table, relative_cover_native = absolute_cover_native*100/(absolute_cover_native + absolute_cover_non_native))
cover3_table <- mutate(cover3_table, relative_cover_non_native = absolute_cover_non_native*100/(absolute_cover_native + absolute_cover_non_native))

# Round all values, and isolate only the relevant columns for the summary tables
cover3_table[,-1] <- round(cover3_table[,-1],1)
cover3_table <- cover3_table %>%
  select(polygon_id, absolute_cover_native, absolute_cover_non_native, absolute_cover_no_native, relative_cover_native, relative_cover_non_native)

View(cover3_table)

```