Assignment 11

Ellen Bledsoe

2025-05-15

Assignment Details

Purpose

The goal of this assignment is to practice writing our own functions and using them.

Task

Write R code to successfully answer each question below.

Criteria for Success

- Code is within the provided code chunks or new code chunks are created where necessary
- Code chunks run without errors
- Code chunks have brief comments indicating which code is answering which part of the question
- Code will be assessed as follows:
  - Produces the correct answer using the requested approach: 100%
  - Generally uses the right approach, but a minor mistake results in an incorrect answer: 90%
  - Attempts to solve the problem and makes some progress using the core concept, but returns the wrong answer and does not demonstrate comfort with the core concept: 50%
  - Answer demonstrates a lack of understanding of the core concept: 0%
- Any questions requiring written answers are answered with sufficient detail

Due Date

April 15 before class

Assignment Exercises

1. Writing Functions–Part 1 (15 points)

a. Copy the following function (which converts weights in pounds to weights in grams) and replace the `________` with the variable names for the input and output. Remove the comments and run the code to add the function to your environment.

```
# convert_pounds_to_grams <- function(________) {
#     grams = 453.6 * pounds
#     return(________)
# }
```

Use the function to calculate how many grams there are in 3.75 pounds.

b. Copy the following function (which converts temperatures in Fahrenheit to temperatures in Celsius) and replace the `________` with the needed commands and variable names so that the function returns

the calculated value for Celsius. Remove the comments and run the code to add the function to your environment.

```
# convert_fahrenheit_to_celsius <- ________(________) {
#     celsius = (fahrenheit - 32) * 5 / 9
#     ________(________)
# }
```

Use the function to calculate the temperature in Celsius if the temperature in Fahrenheit is 80°F.

c. Write a function named `double` that takes a number as input and outputs that number multiplied by 2. Run it with an input of 512.

d. Write a function named `prediction` that takes three arguments, `x`, `a`, and `b`, and returns `y` using `y = a + b * x` (like a prediction from a simple linear model). Run it with `x` = 12, `a` = 6, and `b` = 0.8.

```
###Code solution for Writing Functions

print("1a")
```

```
## [1] "1a"
```

```
pounds_to_grams <- function(weight_lbs){
  weight_g <- weight_lbs * 453.6
  return(weight_g)
}
pounds_to_grams(3.75)
```

```
## [1] 1701
```

```
print("1b")
```

```
## [1] "1b"
```

```
convert_fahrenheit_to_celsius <- function(fahrenheit) {
    celsius = (fahrenheit - 32) * 5 / 9
    return(celsius)
}
convert_fahrenheit_to_celsius(80)
```

```
## [1] 26.66667
```

```
print('1c')
```

```
## [1] "1c"
```

```
# 3. Write a function named `double` that takes a number as input and outputs
#    that number multiplied by 2. Run it with an input of 512.

double <- function(number){
  doubled <- number * 2
  return(doubled)
}

double(512)
```

```
## [1] 1024
```

```
print("1d")
```

```
## [1] "1d"
```

```
# 4. Write a function named `prediction` that takes three arguments, `a`, `b`,
#    and `x`, and returns `y` using `y = a + b * x` (like a prediction from a
#    simple linear model). Run it with `a` = 6, `b` = 0.8, and `x` = 12.

prediction <- function(a, b, x){
  y <- a + b * x
  return(y)
}
prediction(6, 0.8, 12)
```

```
## [1] 15.6
```

2. Use and Modify (15 points)

The length of an organism is typically strongly correlated with its body mass. This is useful because it allows us to estimate the mass of an organism even if we only know its length. This relationship generally takes the form:

mass = a * lengthˆb

Where the parameters `a` and `b` vary among groups. This allometric approach is regularly used to estimate the mass of dinosaurs since we cannot weigh something that is only preserved as bones.

The following function estimates the mass of an organism in kg based on its length in meters for a particular set of parameter values, those for *Theropoda* (where `a` has been estimated as `0.73` and `b` has been estimated as `3.63`; Seebacher 2001).

```
get_mass_from_length_theropoda <- function(length){
  mass <- 0.73 * length ^ 3.63
  return(mass)
}
```

a. Use this function to print out the mass of a Theropoda that is 16 m long based on its reassembled skeleton.

b. Create a new version of this function called `get_mass_from_length()` that takes `length`, `a` and `b` as arguments and uses the following code to estimate the mass `mass <- a * length ^ b`. Use this function to estimate the mass of a Sauropoda (`a = 214.44`, `b = 1.46`) that is 26 m long.

```
###Code solution for Use and Modify

print("2a")
```

```
## [1] "2a"
```

```
get_mass_from_length_theropoda <- function(length){
  mass = 0.73 * length ** 3.63
  return (mass)
}

get_mass_from_length_theropoda(16)
```

```
## [1] 17150.56
```

```
print("2b")
```

```
## [1] "2b"
```

```
get_mass_from_length <- function(a, b, length){
  mass = a * length ** b
```

```
  return (mass)
}

get_mass_from_length(214.44, 1.46, 26)
```

```
## [1] 24955.54
```

3. Writing Functions–Part 2 (15 points)

a. Copy the following function (which converts weights in pounds to weights in grams and rounds them). Replace the `________` with the variable names for the input and output. Replace `__` with a number so that by default the function will round the output to one decimal place.

```
# convert_pounds_to_grams <- function(________, numdigits = __) {
#   grams <- 453.6 * pounds
#   rounded <- round(grams, digits = numdigits)
#   return(________)
# }
```

Use the function to calculate how many grams there are in 4.3 pounds using the default for the number of decimal places.

b. Write a function called `get_height_from_weight` that takes three arguments, `weight`, `a`, and `b`, and returns an estimate of `height` using `height = a * weight ^ b` (a prediction from a power model). Give it default arguments of `a` = 12 and `b` = 0.38. There should be no default value for `weight`. Use the default argument values (by passing only the value of `weight` to the function) to calculate `height` when `weight` = 42.

c. The function in (b) assumes that the weight is provided in grams. Use the functions from (3a) and (3b) in combination to estimate the height for an animal that weighs 2 pounds using the default value for `a`, but changing the value for `b` to 0.32.

```
### Code solution for Writing Functions 2

print("3a")
```

```
## [1] "3a"
```

```
convert_pounds_to_grams <- function(pounds, numdigits = 1) {
  grams <- 453.6 * pounds
  rounded <- round(grams, digits = numdigits)
  return(rounded)
}

convert_pounds_to_grams(4.3)
```

```
## [1] 1950.5
```

```
print("3b")
```

```
## [1] "3b"
```

```
get_height_from_weight <- function(weight, a = 12, b = 0.38){
  height <- a * weight ^ b
  return(height)
}

get_height_from_weight(42)
```

```
## [1] 49.66106
```

```
print("3c")
```

```
## [1] "3c"
```

```
convert_pounds_to_grams(2) |>
  get_height_from_weight(b = 0.32)
```

```
## [1] 106.0831
```

4. Default Arguments (15 points)

This is a follow up to Use and Modify.

Allowing `a` and `b` to be passed as arguments to `get_mass_from_length()` made the function more flexible, but for some types of dinosaurs we don't have specific values of `a` and `b` and so we have to use general values that can be applied to a number of different species.

Rewrite your `get_mass_from length()` function from Question 2 so that its arguments have default values of `a = 39.9` and `b = 2.6` (the average values from Seebacher 2001).

a. Use this function to estimate the mass of a Sauropoda (`a = 214.44`, `b = 1.46`) that is 22 m long (by setting `a` and `b` when calling the function).
b. Use this function to estimate the mass of a dinosaur from an unknown taxonomic group that is 16m long. Only pass the function `length`, not `a` and `b`, so that the default values are used.

```
print("4a")
```

```
## [1] "4a"
```

```
get_mass_from_length <- function(a = 39.9, b = 2.6, length){
  mass = a * length ** b
  return (mass)
}
get_mass_from_length(a = 214.44, b = 1.46, length = 22)
```

```
## [1] 19554.33
```

```
print("4b")
```

```
## [1] "4b"
```

```
get_mass_from_length(length = 16)
```

```
## [1] 53911.93
```

5. Combining Functions (20 points)

This is a follow up to Default Arguments, Question 4.

Measuring things using the metric system is the standard approach for scientists, but when communicating your results more broadly it may be useful to use different units (at least in some countries).

Write a function called `convert_kg_to_pounds` that converts kilograms into pounds (`pounds = 2.205 * kg`).

Use that function and your `get_mass_from_length()` function from Question 4 to estimate the weight, in pounds, of a 12 m long Stegosaurus with `a = 10.95` and `b = 2.64` (The estimated `a` and `b` values for *Stegosauria* from Seebacher 2001).

```r
convert_kg_to_pounds <- function(kg){
  pounds = 2.205 * kg
  return(pounds)
}

get_mass_from_length(a = 10.95, b = 2.64, length = 12) |>
  convert_kg_to_pounds()
```

```
## [1] 17055.37
```

6. Writing Tidyverse Functions (20 points)

First, load either the `tidyverse` or `ggplot2`, if you haven't already done so in this document.

Run the code below to create a data frame named `count_data` with columns named `state`, `count`, `area`, and `site`.

```r
count_data <- data.frame(
  state = c("AZ", "AZ", "AZ", "AZ", "NM", "NM", "NM", "NM", "NV", "NV", "NV", "NV"),
  site = c("A", "B", "C", "D", "A", "B", "C", "D", "A", "B", "C", "D"),
  count = c(9, 16, 3, 10, 2, 26, 5, 8, 17, 8, 2, 6),
  area = c(3, 5, 1.9, 2.7, 2, 2.6, 6.2, 4.5, 8, 4, 1, 3))
```

a. Write a function takes takes two arguments: (1) a data frame with a `count` column and an `area` column; and (2) a column in that data frame to color the points by.

   Have the function make a plot with `area` on the x-axis and `count` on the y-axis and the points colored by the column you provided as an argument. Set the size of the points to 3. Use the function to make a scatter plot of count as a function of area for the `count_data` data frame with the points colored by the `state` column.

   Save your plot as a .png file using `ggsave()` in the appropriate sub-directory.

b. Use the function from (a) to make a scatter plot of count as a function of area for the `count_data` data frame with the points colored by the `site` column.

   Save your plot as a .png file using `ggsave()` in the appropriate sub-directory.

c. Create a new function based on the function you created in (a); in this new function, allow the x and y axes to be any column in the data frame, specified as arguments in the function.

   Remake the same plot as from (b) using your new function. Save your plot as a .png file using `ggsave()` in the appropriate sub-directory.

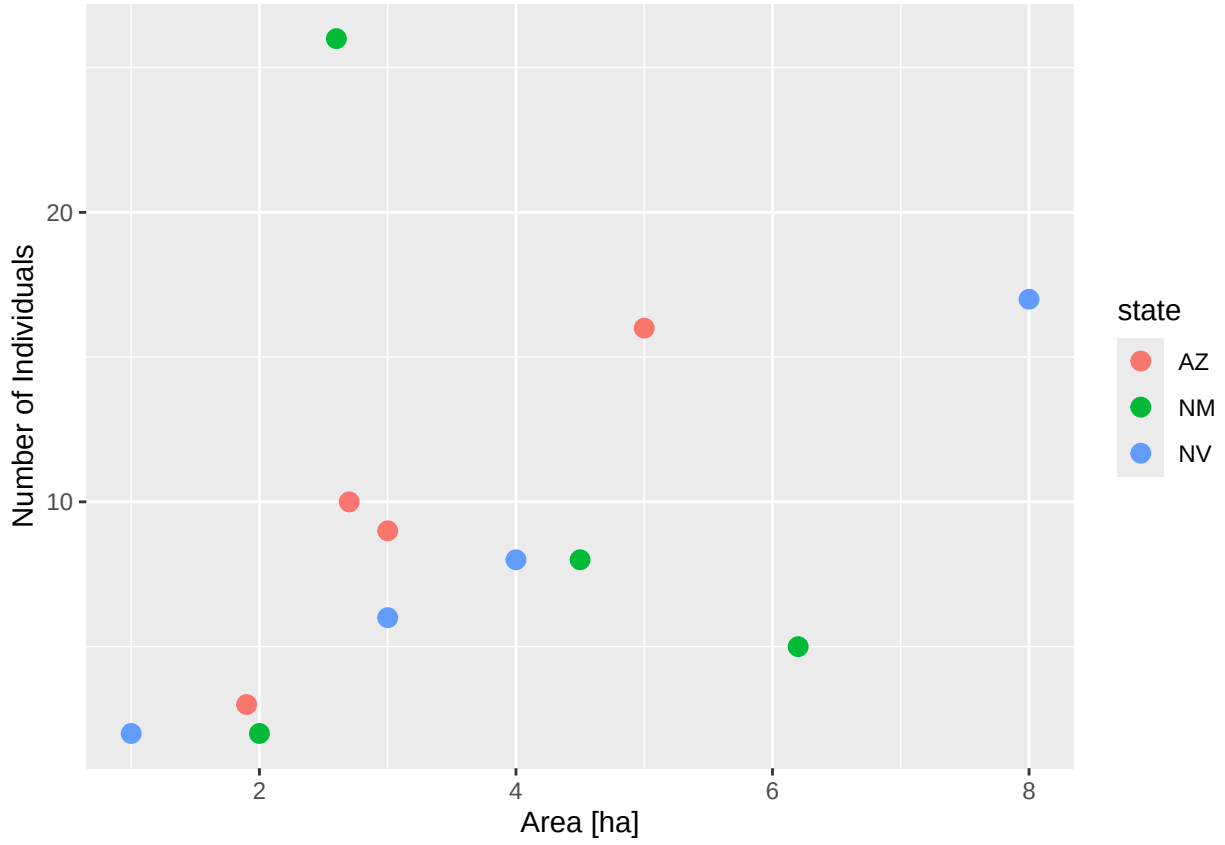
```r
# Solutions to Writing Tidyverse Functions
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.2     v tibble    3.2.1
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.0.4
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
print("6a")
```

```
## [1] "6a"
```

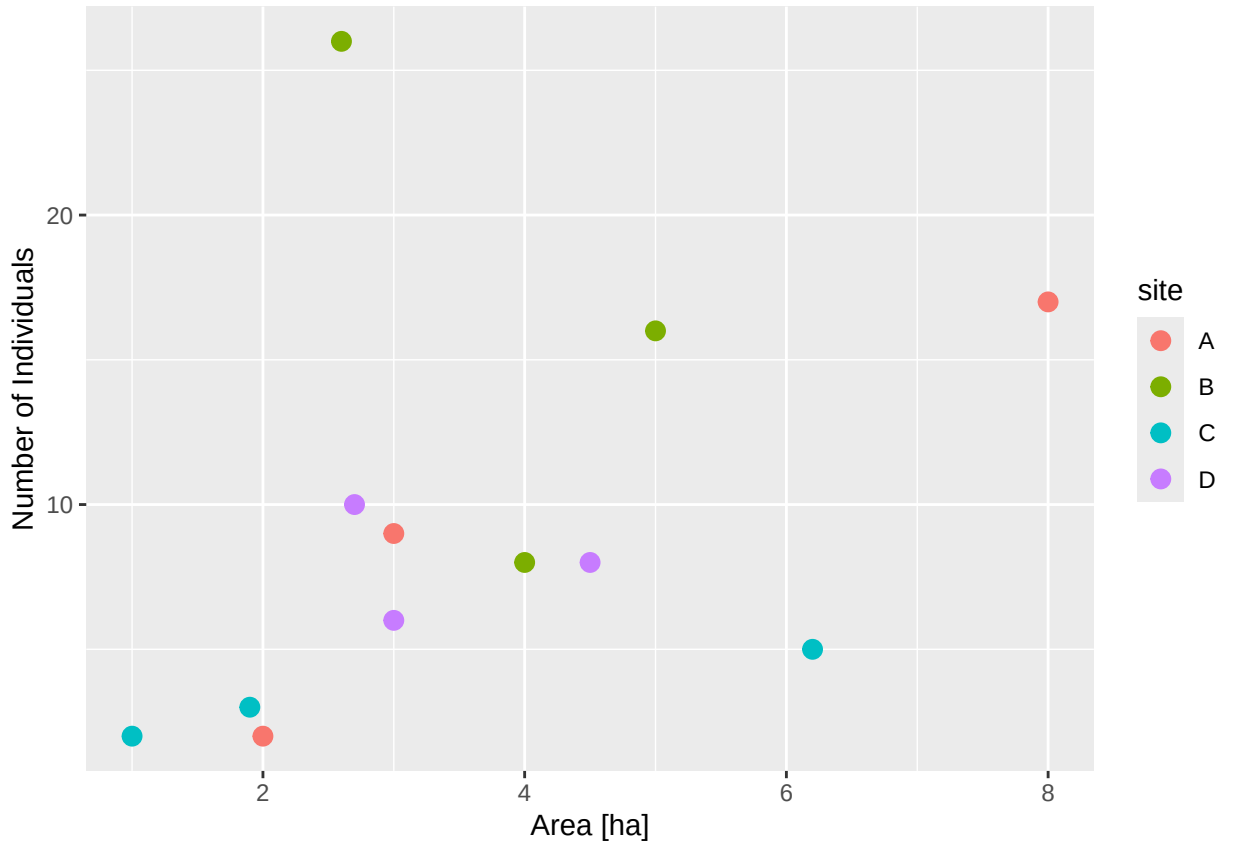
```
make_plot <- function(df, color_col) {
  ggplot(data = df, mapping = aes(x = area,
                                  y = count,
                                  color = {{ color_col }})) +
    geom_point(size = 3) +
    labs(x = "Area [ha]", y = "Number of Individuals")
}

make_plot(count_data, state)
```




```
print("6b")
```

```
## [1] "6b"
```

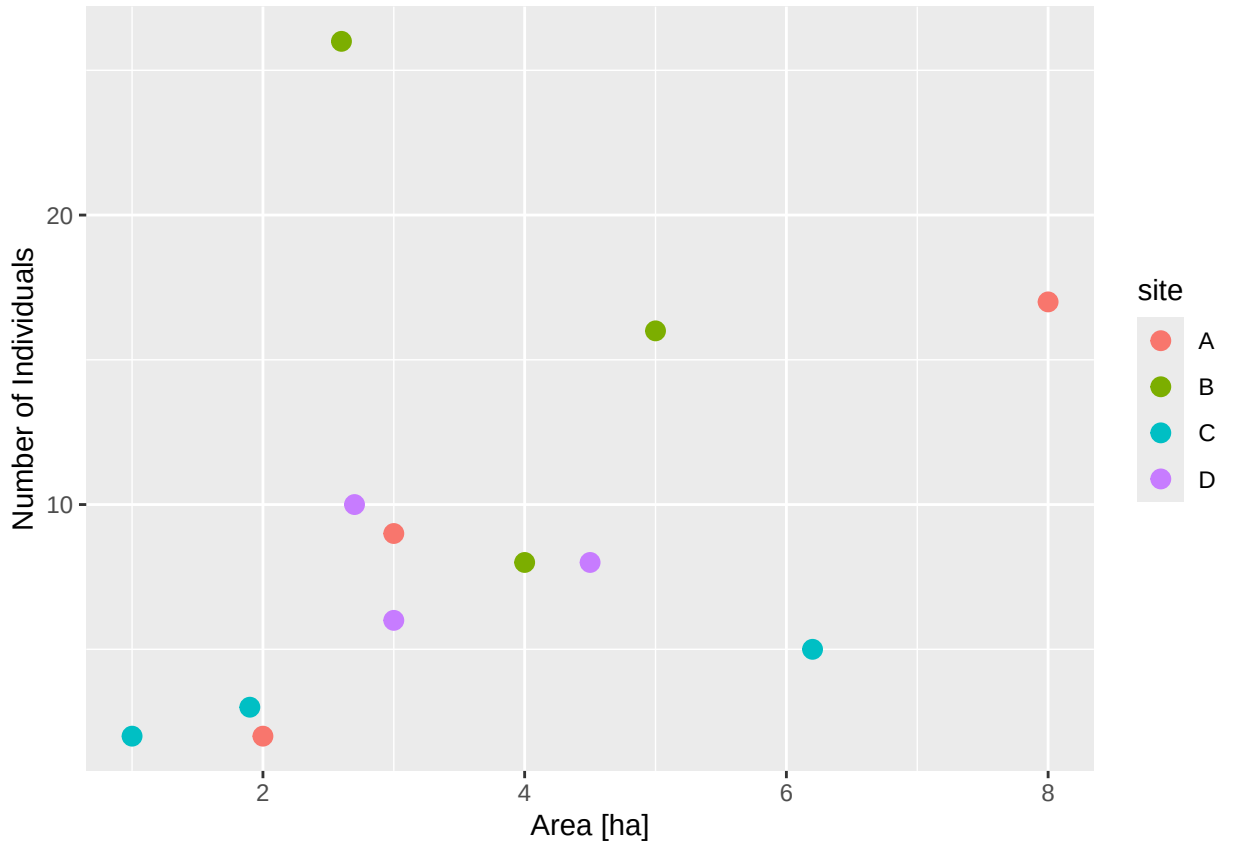
```
make_plot(count_data, site)
```




```
print("6c")
```

```
## [1] "6c"
```

```
make_plot2 <- function(df, x_axis, y_axis, color_col, x_lab, y_lab) {
  ggplot(data = df, mapping = aes(x = {{ x_axis }},
                                  y = {{ y_axis }},
                                  color = {{ color_col }})) +
    geom_point(size = 3) +
    labs(x = x_lab, y = y_lab)
}
make_plot2(count_data, area, count, site, "Area [ha]", "Number of Individuals")
```



7. Portal Species Time-Series Challenge (OPTIONAL)

If surveys.csv, species.csv, and plots.csv are not available in your workspace download them (modify the `destfile` arguments to point to the correct sub-directory):

```
download.file(url = "https://ndownloader.figshare.com/files/2292172",
  destfile = "data/surveys.csv")
download.file(url = "https://ndownloader.figshare.com/files/3299474",
  destfile = "data/plots.csv")
download.file(url = "https://ndownloader.figshare.com/files/3299483",
  destfile = "data/species.csv")
```

a. Write a function that does the following:

- Takes the following four arguments:

  (1) a data frame (where each row is one individual and there is a genus and a species column)
  (2) a column to use as a `time` column (e.g., year)
  (3) a `genus_name` argument for choosing which genus to plot
  (4) a `species_name` argument for choosing which species to plot

- Makes a plot using `ggplot2` with the time on the y-axis and the count of the number of individuals (i.e., the number of rows) observed for that time for the species indicated by the genus_name and species_name arguments. The plot should display the data as blue points (with `size = 2`) connected by blue lines (with `linewidth = 1`). The y-axis label should read "Number of Individuals." The theme should be "classic."

b. Use your function, and the data in `surveys.csv` and `species.csv`, to plot the time-series for `time = year`, `genus_name = "Dipodomys"` and `species_name = "merriami"`.

c. Use your function, and the data in `surveys.csv` and `species.csv`, to plot the time-series for `time = month`, `genus_name = "Chaetodipus"` and `species_name = "penicillatus"` (this plot will show the average seasonal pattern of *Chaetodipus penicillatus* abundances)

d. Use your function, and the data from `plots.csv`, `surveys.csv` and `species.csv`, to plot the time-series for `time = year`, `genus_name = "Chaetodipus"` and `species_name = "baileyi"` only on the "Control" plots.

```
### Code solution for Portal Species Time-Series

surveys <- read_csv('data/surveys.csv')
```

```
## Rows: 35549 Columns: 9
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): species_id, sex
## dbl (7): record_id, month, day, year, plot_id, hindfoot_length, weight
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
species <- read_csv('data/species.csv')
```

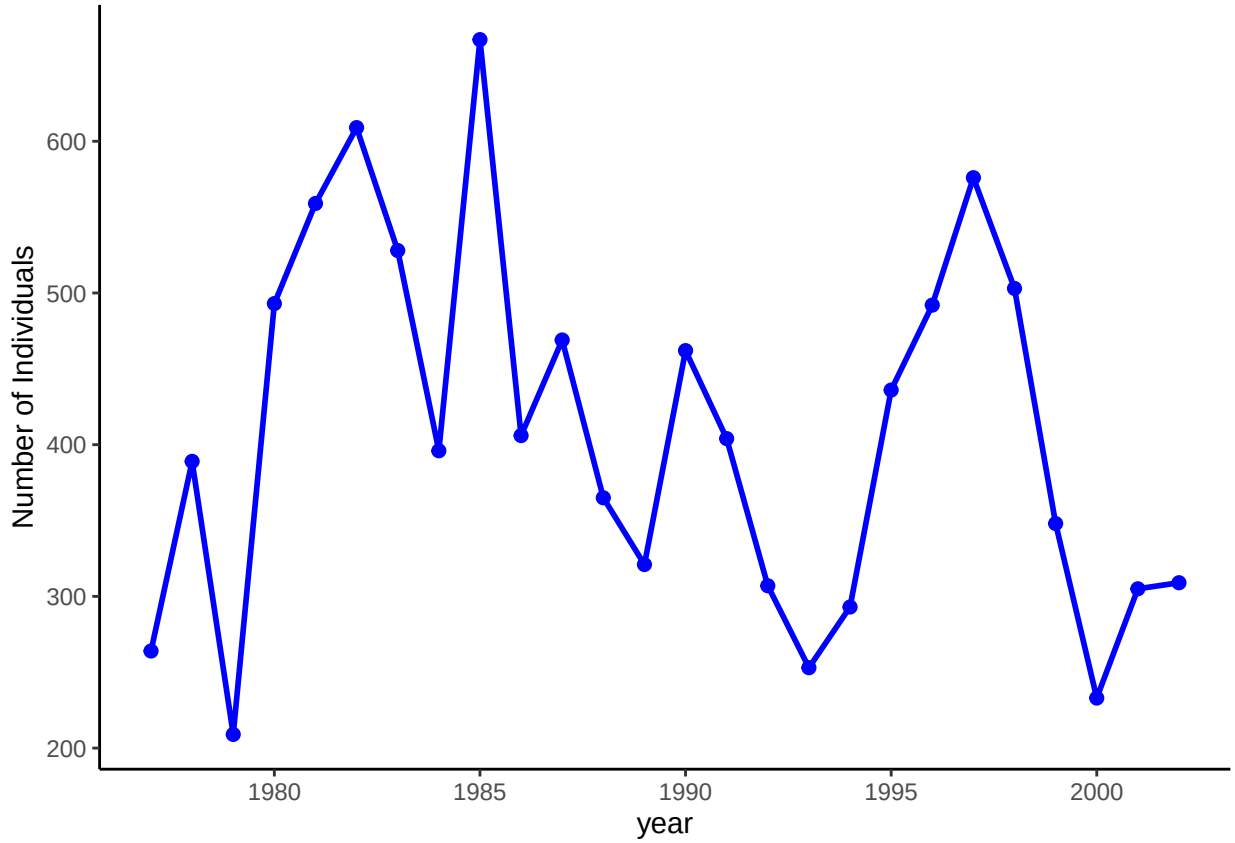
```
## Rows: 54 Columns: 4
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (4): species_id, genus, species, taxa
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
plots <- read_csv('data/plots.csv')
```

```
## Rows: 24 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): plot_type
## dbl (1): plot_id
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
make_sp_time_series_plot <- function(data, timestep, genus_name, species_name){

  time_series <- data |>
    filter(genus == genus_name, species == species_name) |>
    group_by({{ timestep }}) |>
    summarize(count = n())

  ggplot(time_series, aes(x = {{ timestep }}, y = count)) +
    geom_line(linewidth = 1, color = "blue") +
    geom_point(size = 2, color = "blue") +
    ylab("Number of Individuals") +
    theme_classic()

}
```

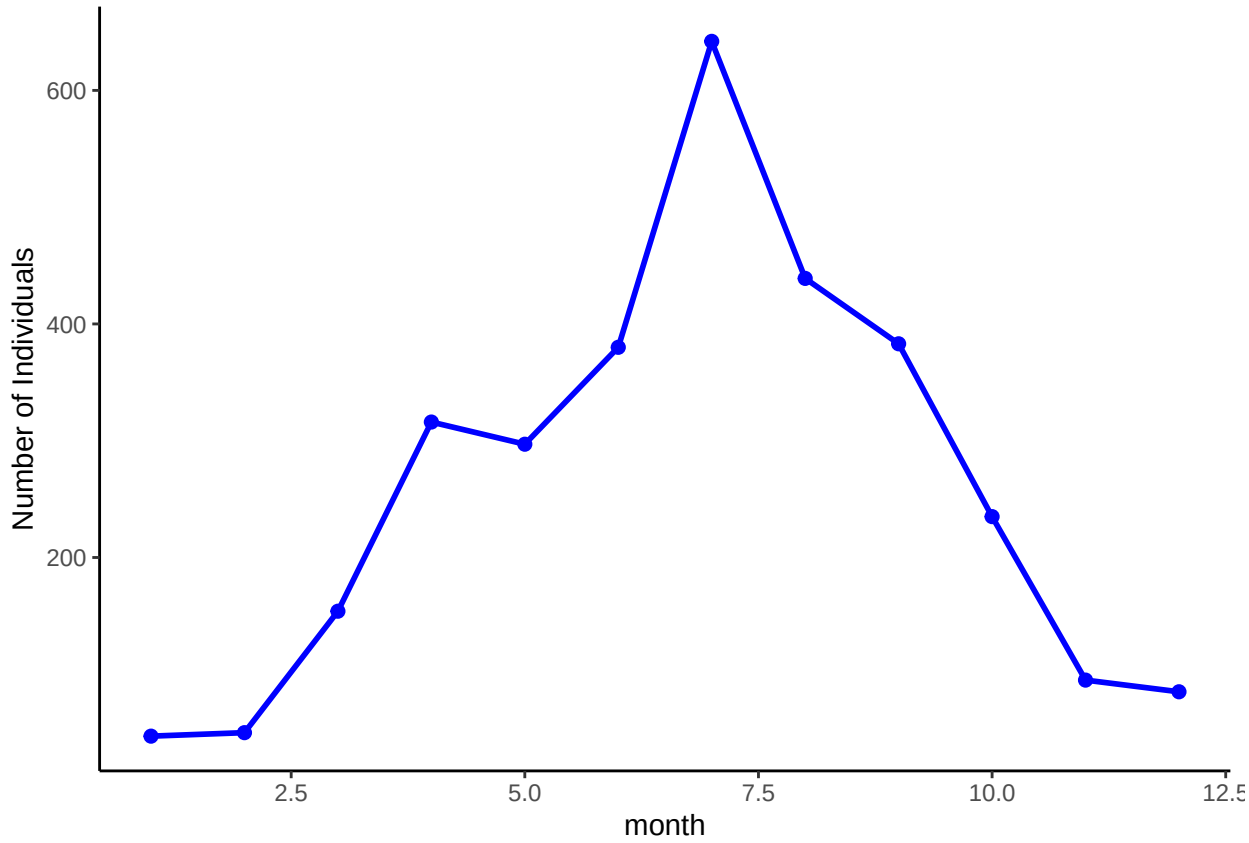
```
combined <- inner_join(surveys, species, by = "species_id")

make_sp_time_series_plot(combined, year, "Dipodomys", "merriami")
```




```
ggsave("output/Functions-portal-species-time-series-R-1.png")
```

```
## Saving 6.5 x 4.5 in image
```

```
make_sp_time_series_plot(combined, month, "Chaetodipus", "penicillatus")
```

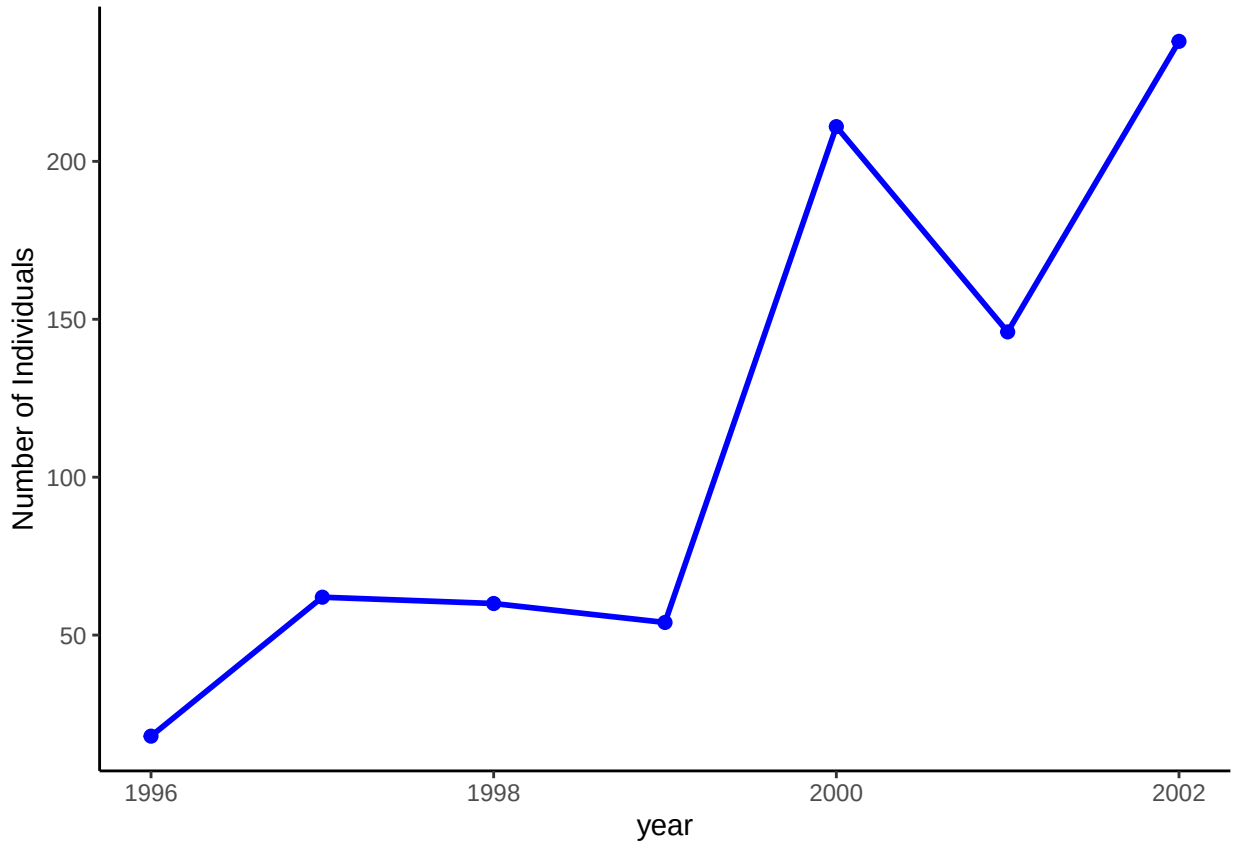



```
ggsave("output/Functions-portal-species-time-series-R-2.png")
```

```
## Saving 6.5 x 4.5 in image
```

```
combined |>
  left_join(plots) |>
  filter(plot_type == "Control") |>
  make_sp_time_series_plot(year, "Chaetodipus", "baileyi")
```

```
## Joining with `by = join_by(plot_id)`
```




```
ggsave("output/Functions-portal-species-time-series-R-3.png")
```

```
## Saving 6.5 x 4.5 in image
```