# Lesson 6 Ornithology Theme

by Lorraine Gaudio

Lesson generated on September 28, 2025

BOISE STATE UNIVERSITY

# Contents

# 1. ☒ Welcome back to R!

In lesson one through four you learned about objects, vectors, functions and how to handle missing data in R. In this lesson, you will learn about subsetting, extraction, and insertion in data frames. You rarely analyse an entire data table at once. You nearly always pull out specific rows or columns, or write back cleaned-up values. Mastering R's bracket "[ ]" syntax is therefore foundational.

To begin Lesson 6, follow these steps:

1. Open your course project for RStudio
2. Create a new file. Today, let's use □ "R Markdown" again (File > New File > R Markdown).
3. Type in the code provided in this document as you follow along with the video. Pause the video at anytime to answer learning skill prompts by digging deeper or add memo notes.
4. Complete the assignment questions in this same document.
5. Submit the RMD and RData file to the Canvas assignment for this lesson.

**Lesson Overview**

By the end of Lesson 6 you will be able to:

1. □ Remember – State R's basic subsetting template: x[rows, cols].
2. □ Understand – Describe the usefulness of creating a copy of original data.
3. □ Apply – Use logical tests + which() to select rows.
4. □ Analyze – Summarize a subset to answer a question.
5. □ Evaluate – Choose an insertion (overwrite) vs. safe copy strategy

Keep these goals in mind as you move through each section.

# 2. Build Dataframes

We'll need example dataframes to play with for this lesson. You can copy and paste the code that creates the dataframes. To learn more about how these dataframes were created, complete the Lesson 6 Point Boost!

## 2.1 Simple Dataframe

We create trial_df as a simple example. Type or copy and paste the following code in a new code chunk and run.

```
set.seed(1)
trial_vec <- sample(x = c(1:3, NA, NA, NA, NA), size = 25, replace = TRUE)
trial_vec
```

```
## [1]  1 NA NA  1  2 NA NA  3 NA  2  3  3  1 NA NA  2 NA NA  2 NA  1 NA NA NA  1
```

```
# Example: Create a new vector trial_vec2 based on trial_vec
trial_vec2 <- ifelse(trial_vec == 2, "two", "not two")
```

Now, let's create a data frame using trial_vec and trial_vec2. Type the following code in a new code chunk and run.

```
# Create a data frame with trial_vec and trial_vec2
trial_df <- data.frame(trial_vec, trial_vec2)
```

□ Check the data frame trial_df.

```
# head(trial_df)
head(trial_df) # Prints only for first 6 rows of trial_df
```

| trial_vec | trial_vec2 |
|---:|:---|
| 1 | not two |
| NA | NA |
| NA | NA |
| 1 | not two |
| 2 | two |

| trial_vec | trial_vec2 |
|---|---|
| NA | NA |

□ Are you ready? Remember to summon ?help whenever you need it.

## 2.2 Field Biology

Dataset theme: Bird Band Observations □

This theme simulates data from a field biologist observing a population of birds, where most are untagged, but a small number have been previously captured and marked with colored leg bands. This is a very common practice in ecology and behavioral studies.

We'll simulate a small inventory called field_sightings with two columns:

- Status – Describes whether the sighted bird is "Untagged" (the common case) or "Tagged" (the rare case).
- Band_Color – If a bird is untagged, the value is "None." If it is tagged, the value is one of several possible band colors.

**Copy and paste** the following code in a new code chunk.

```r
set.seed(8)
Status  <- sample(c(rep("Untagged", 4168), rep("Tagged", 302)))
Band_Color <- ifelse(Status == "Untagged", "None",
         sample(c("Red", "Blue", "Green", "Yellow",
              "Silver", "Black", "Orange",
              "White"), length(Status), replace = TRUE))
field_sightings <- data.frame(Status, Band_Color)

# View the data
View(field_sightings)
```

□ Look deeper: Why does the code have line breaks with indentation? What is the advantage of indented line breaks? When might this be useful for you? Create a memo note, demonstrate learning skill(s) used.

# 3. Bracket Syntax

□ Review Lesson 2 about indexing vectors x[ ]. Create a memo note, demonstrate learning skill(s) used. Create a memo note, demonstrate learning skill(s) used.

In this lesson we will use the same syntax to subset data frames.

- x is the object (vector, data frame, etc.)
- [ ] is the bracket operator for subsetting
- x[rows, cols] extracts specific rows and columns from x.
- rows and cols can be: indices (1, 3:5), names ("Band_Color"), or a logical vector / expression (Status == "Tagged").

Type the following code in a new code chunk and run.

```
field_sightings[1:3, ]  # □ Shows full row extraction
```

| Status | Band_Color |
|---|---|
| Untagged | None |
| Untagged | None |
| Untagged | None |

```
# R's subsetting format:  x[rows, columns]
field_sightings[1:3, 1]  # □ Shows first three rows of first column
```

```
## [1] "Untagged" "Untagged" "Untagged"
```

```
field_sightings[1:3, c(1, 2)]  # □ Shows first three rows of first and second
```

| Status | Band_Color |
|---|---|
| Untagged | None |
| Untagged | None |
| Untagged | None |

# 4. Subsetting

We'll start with trial_df from earlier. Type the following code in a new code chunk and run.

```
# Subsetting
# trial_df[ trial_df[ , "trial_vec"] == 2 , ]
```

```
direct_subset <- trial_df[ trial_df[ , "trial_vec"] == 2 , ]
head(direct_subset)
```

| | trial_vec | trial_vec2 |
|---|---|---|
| NA | NA | NA |
| NA.1 | NA | NA |
| 5 | 2 | two |
| NA.2 | NA | NA |
| NA.3 | NA | NA |
| NA.4 | NA | NA |

□ Identify: which rows are extracted?

- TRUE (2 == 2)
- FALSE (1 == 2)
- FALSE (3 == 2)
- NA (NA == 2)

□ **Goal: isolate only the tagged birds.**

□ **Step 1** – create a logical test using [ ]. Type the following code in a new code chunk and run.

```
# Step 1 — Create and store the logical test (no big printout)
logical_test <- field_sightings[ , "Status"] == "Tagged"

# un-comment the line below to see the object logical_test
# print(logical_test) #prints a long list of values

# Instead, lets make a quick table of logical_test
table(logical_test, useNA = "ifany")   # counts of TRUE / FALSE / NA
```

```
## logical_test
## FALSE  TRUE
##  4168   302
```

□ **Step 2** – Store subset of field_sightings data frame as a new data frame. Type the following code in a new code chunk and run.

```
# Step 2 — Store the subset as a new data frame
tagged_only <- field_sightings[ field_sightings[ , "Status"] == "Tagged" , ]

head(tagged_only)
```

| | Status | Band_Color |
|---|---|---|
| 18 | Tagged | Orange |
| 22 | Tagged | Silver |
| 31 | Tagged | Green |
| 41 | Tagged | Black |
| 50 | Tagged | Silver |
| 66 | Tagged | Blue |

□ Look deeper: How many *rows* did we keep? How could you check the number of rows in a data set? Create a memo note, demonstrate learning skill(s) used, including method you used to answer the question.

Type the following code in a new code chunk and run.

```
n_tagged <- nrow(tagged_only)  # □  Check number of rows
print(n_tagged)
```

```
## [1] 302
```

And 302 makes sense because of this code that helped create the dataset replicated "Tagged" 302 times.

```
# Confirm the filter worked
all(tagged_only[ , "Status"] == "Tagged")
```

```
## [1] TRUE
```

```
unique(tagged_only[ , "Status"])
```

```
## [1] "Tagged"
```

# 5. Extraction

Using which() is a strategy to handle potential NA values in your data. When the test could include NA, wrap it in which() to drop unknowns.

Type the following R script in your document script and run.

```
rows_without_na <- which(trial_df[ , "trial_vec"] == 2)

which_subset <- trial_df[rows_without_na, ]

print(which_subset)
```

```
##    trial_vec trial_vec2
## 5          2        two
## 10         2        two
## 16         2        two
## 19         2        two
```

□ Explore and Play: What is the difference between running print(which_subset) and which_subset? Create a memo note, demonstrate learning skill(s) used.

When you use a direct logical condition like trial_df[ , "trial_vec"] == 2, any NA values produce NA in the logical vector. Wrapping the condition in which() returns only the row positions where the test is TRUE, effectively dropping rows where the comparison is NA.

```
# Two conditions
row_idx <- which(field_sightings[ , "Status"] == "Tagged" &
                 field_sightings[ , "Band_Color"] == "Blue")

subset_blue <- field_sightings[row_idx, ]
```

row_idx stores the row positions where both conditions are TRUE - Status == "Tagged" - Band_Color == "Blue"

Wrapping the test in which() returns only those positions (and automatically drops any rows where the comparison was NA).

subset_blue <- field_sightings[row_idx, ] then keeps all columns but only the matching rows, preserving the original row order and row names.

Let's verify that it worked.

```
head(subset_blue)
```

|  | Status | Band_Color |
|---|---|---|
| 66 | Tagged | Blue |
| 284 | Tagged | Blue |
| 430 | Tagged | Blue |
| 675 | Tagged | Blue |
| 700 | Tagged | Blue |
| 751 | Tagged | Blue |

□ Comment (respond in one or two sentences): What changed in the data after subsetting (rows, columns, values)? What stayed the same (column names, column order, data types)?

## 5.1 Selecting Columns

□ Select columns by name or by index using indexing.

Recall that columns can be chosen by index or name.

- By index: field_sightings[ , 2] (second column)
- By name : field_sightings[ , "Band_Color"]

```
head(subset_blue[ , "Band_Color"])
```

```
## [1] "Blue" "Blue" "Blue" "Blue" "Blue" "Blue"
```

You can combine row + column ideas: band color of the first 10 birds. Type the following code in a new code chunk and run.

```
field_sightings[1:10, "Band_Color"]
```

```
##  [1] "None" "None" "None" "None" "None" "None" "None" "None" "None" "None"
```

□ Why did we use head() in the first chunk and not in the second? Create a memo note, demonstrate learning skill(s) used.

## 5.2 Selecting Multiple Columns

1. Compute a filtered row index with which(), then
2. Select multiple columns with df[row_idx, c("colA","colB", …)].

```r
#  1) Row index with two conditions
row_idx <- which(
  field_sightings[ , "Status"] == "Tagged" &
  (field_sightings[ , "Band_Color"] == "Blue" |
   field_sightings[ , "Band_Color"] == "White")
)

# 2) Apply row + multi-column selection (select multiple columns by NAME)
tagged_blue_white <- field_sightings[row_idx, c("Status", "Band_Color")]

head(tagged_blue_white)
```

| | Status | Band_Color |
|---|---|---|
| 66 | Tagged | Blue |
| 84 | Tagged | White |
| 238 | Tagged | White |
| 247 | Tagged | White |
| 284 | Tagged | Blue |
| 397 | Tagged | White |

□ Comment: What changed after subsetting—rows, columns, or both? Verify the filter worked: do all rows have Status == “Tagged” and Band_Color in c(“Blue”,“White”)? About how many rows remain compared to the full field_sightings? Why is that number smaller?

# 6. Insertion

Insertion is the process of adding or modifying values in a data frame. You can insert new values or overwrite existing ones.

## 6.1 Overwrite Values

Suppose we decide to rename “Untagged” to “Wild_Hatch” in the Status column.

□ Good practice: make a backup first! Type the following code in a new code chunk and run.

```
# Part 1
reintroduction <- field_sightings          # copy
indices_untagged <- which(field_sightings[ , "Status"] == "Untagged")
reintroduction[indices_untagged, "Status"] <- "Wild_Hatch"
```

□ Check-in: Look at the format of that code: x[rows, column] <- value. Here, we assign the character value "Wild_Hatch" into reintroduction at the rows given by indices_untagged and the "Status" column.

```
# Verify the change
unique(reintroduction[ , "Status"])
```

```
## [1] "Wild_Hatch" "Tagged"
```

□ Comment: Why is field_sightings unchanged while editing reintroduction? Why is this good practice? Create a memo note, demonstrate learning skill(s) used.

### 6.1.1 Renaming columns

```
# Part 2 — Add a new column pattern
reintroduction[ , "Tag_Flag"] <- ifelse(
  reintroduction[ , "Status"] == "Tagged", "Yes", "No"
)
head(reintroduction[ , c("Status", "Tag_Flag")])
```

| Status | Tag_Flag |
|---|---|
| Wild_Hatch | No |
| Wild_Hatch | No |
| Wild_Hatch | No |
| Wild_Hatch | No |
| Wild_Hatch | No |
| Wild_Hatch | No |

# 7. Renaming Columns

Example 1: To rename a column, you can use the colnames() function. Type the following code in a new code chunk and run.

```
# Part 2
names(reintroduction)
```

```
## [1] "Status"     "Band_Color" "Tag_Flag"
```

```
colnames(reintroduction) <- c("Band_Status", "Breeding_Cohort", "Tag_Flag")
head(reintroduction)
```

| Band_Status | Breeding_Cohort | Tag_Flag |
|---|---|---|
| Wild_Hatch | None | No |
| Wild_Hatch | None | No |
| Wild_Hatch | None | No |
| Wild_Hatch | None | No |
| Wild_Hatch | None | No |
| Wild_Hatch | None | No |

□ Identify: Which columns are renamed? □ Look deeper: Create a list of all the different ways you might identify the column names of a data frame? Create a memo note, demonstrate learning skill(s) used.

---

Example 2: You can also rename a specific column by index. Type the following code in a new code chunk and run.

```
reintroduction2 <- field_sightings     # □ Always copy first
names(reintroduction2)[which(names(reintroduction2) == "Status")] <- "Band_Status"
names(reintroduction2)[which(names(reintroduction2) == "Band_Color")] <- "Breeding_Cohort"
```

```
head(reintroduction2)
```

| Band_Status | Breeding_Cohort |
|---|---|
| Untagged | None |
| Untagged | None |
| Untagged | None |
| Untagged | None |
| Untagged | None |
| Untagged | None |

# 8. □ Assignment

Now it's your turn to practice creating and using vector objects. Follow the tasks below to complete part of the **technical skill practice assignment**.

1. Work through each task in order. Replace the ___ placeholder with your code or short written answer.
2. Run each completed line to be sure no errors appear and objects show in the Environment.
3. When finished, save your workspace and submit this R Markdown file (RMD) plus the .RData file.

## 8.1 Task 0

□ Setup: Load the built-in data set mtcars and take a quick look. Identify what the columns contain and meaning of that content.

```
data("mtcars")  # already in memory but this keeps the workflow explicit
?mtcars        # help file for variable descriptions
View(mtcars)   # spreadsheet view (optional)
```

□ Comment: Explain R's basic subsetting template. Hint: It looks like data[ rows , columns ]

## 8.2 Task 1

□ Manual vs. Automatic

1. Create Manual_Cars: all rows where am == 1 (manual), all columns.

```
Manual_Cars <- ___
```

2. Use nrow() to record how many manual cars there are (store in n_manual).

```
n_manual <- ___ # numeric count
```

## 8.3 Task 2

□ MPG by Cylinder

Extract mpg for each cylinder group into separate objects. Hint: Repeat the bracket pattern

```
Four_Cyl_MPG  <- mtcars[ mtcars[ , "cyl"] == 4 ,
Six_Cyl_MPG   <- ___
Eight_Cyl_MPG <- ___
```

## 8.4 Task 3

□ Analyze Fuel Efficiency

1. Compute the mean mpg for each object.

```
mean_4  <- mean(___)
mean_6  <- mean(___)
mean_8  <- mean(___)
```

2. □ Comment: Which cylinder group is most fuel□ efficient? Answer in one word or number.

Most efficient: ___

## 8.5 Task 4

□ Safe vs. Direct Editing

1. Make a safe copy of mtcars called cars_safe.

```
cars_safe <- ___
```

2. Rename the 'am' column in cars_safe to "Transmission_Type".

```
colnames(cars_safe)[___] <- ___
```

3. □ Comment: Explain *why* working on cars_safe (a copy) is safer than editing mtcars directly.

Answer: ___

## 8.6 Task 5

□ Practice which()

Using cars_safe, extract the rows for V□ shaped engines (vs == 0) *and* horsepower (hp) above 150. Use which() for the row indices and keep only mpg, hp, and cyl columns. Name the object hi_power_v.

1. Create row_idx for the condition above.

- cars_safe[ , "vs"] ? 0
- &
- cars_safe[ , "hp"] ? 150
- cars_safe[row_idx , c("mpg","hp","cyl")]

```
row_idx     <- ___
```

2. Subset cars_safe with row_idx and columns mpg, hp, cyl. Name object hi_power_v.

```
hi_power_v  <- ___
head(hi_power_v)
```

## 8.7 Task 6

□ Insertion Example

Add a new column called "Efficiency_Class" to cars_safe. Label "High" if mpg >= 25, "Low" otherwise.

Hint: use ifelse() and cars_safe[ , "mpg"] >= 25, "?", "?"

```
cars_safe[ , "Efficiency_Class"] <- ___
```

# 9. Save and Upload

1. You will be submitting **both** the R Markdown and the workspace file. The workspace file saves all the objects in your environment that you created in this lesson.

Or you can click the "Save Workspace" button in the Environment pane.

□ **Always save the R documents before closing.**

2. Find the assignment in this week's module in Canvas and upload **both** the RMD and the workspace file.

## 10. Today you practiced:

- Reading R’s subsetting template x[rows, cols].
- Extracting rows with logical tests and which().
- Selecting columns by index or name.
- Safely overwriting values (insertion) after making a copy.
- Renaming columns.

□ Great job! Subsetting is the gateway to every data-cleaning task.