BINF 5003: Data Mining, Modeling, and Biostatistics

Week 7

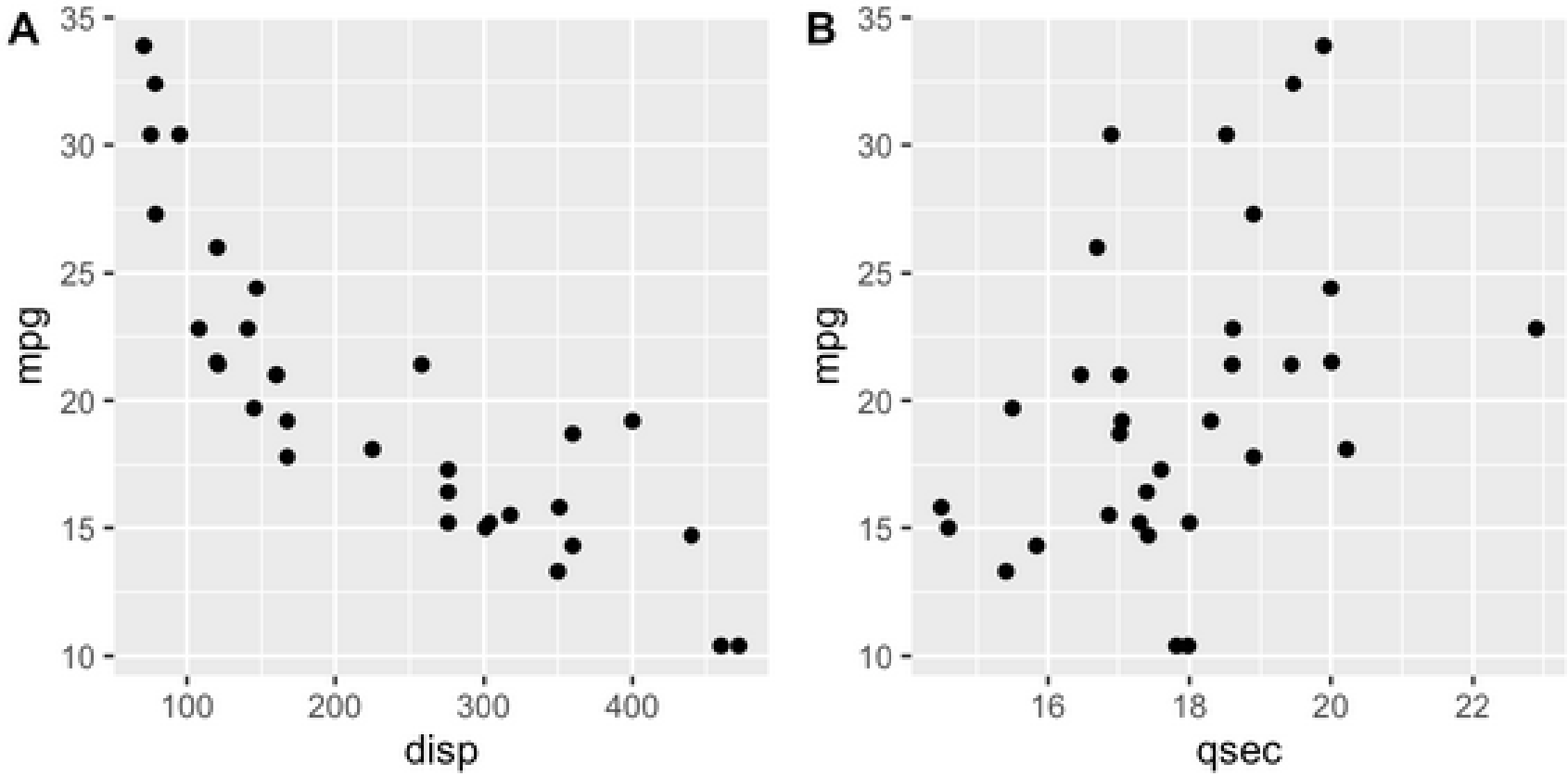
Module 4 – Data Modeling

Overview

- Multi-panel plots
- Revisiting repetitive tasks
- Automating sets of tasks with For Loops
  - Compare and contrast with generalizable
- Tips for troubleshooting code

Cowplot – multipanel plots

Fully arrange your figures before exporting



Cowplot – multipanel plots

- Save each panel of the figure into their own object
- Organize the figure by rows and then combine
- Can integrate labels
- Promote consistency of figure formats (same colour, size, shape) and labels

Revisiting: Repetitive tasks

- When you do a task 3 or more times, it is work considering:
  - Optimizing your code to be generalizable
  - Writing a custom function
  - Automating the task

- Save time
- Prevent errors when copy and pasting or typos

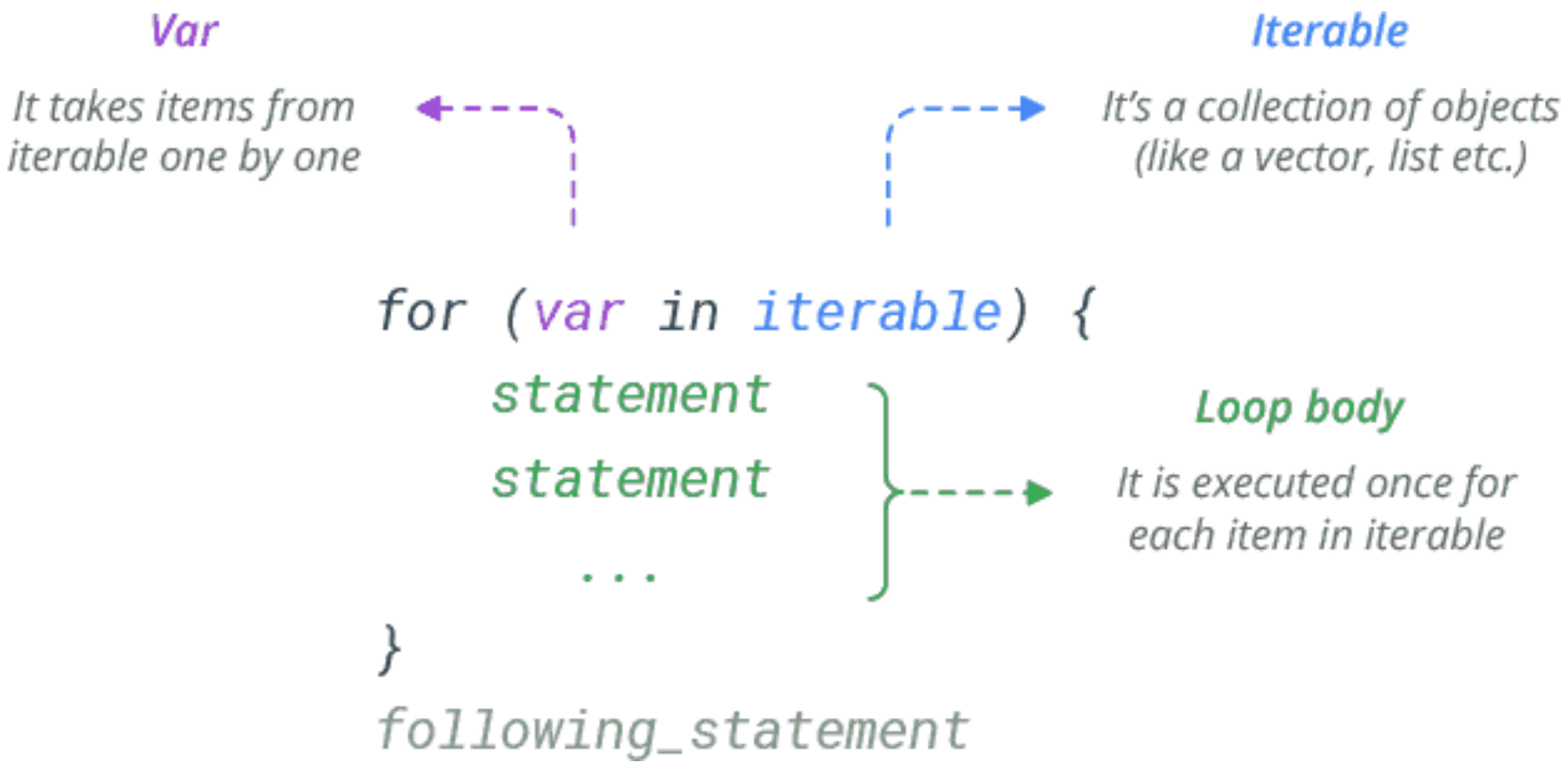
For Loops

- Rather than apply a function multiple times, a `for loop` will iterate through all items in a list

- Need to know all the options you want to apply the code to at the start

```
> fruits
[1] "orange"     "apple"      "banana"     "grapefruit"
[5] "starfruit"
> for (x in fruits) {
+     print(x)
+ }
[1] "orange"
[1] "apple"
[1] "banana"
[1] "grapefruit"
[1] "starfruit"
```

Automating repetitive tasks with loops



https://www.learnbyexample.org/r-for-loop/

Syntax

- Function: `for()`
  - Single argument in the format of `[placeholder] in [list]`
  - Also interacts with the code in the curly brackets

- Curly brackets hold longer lines of code
  - Not necessary if the command is only one line long

What can you iterate through?

- Vectors are the most straightforward

- Operate on the results of a command
  - E.g., unique(object$columnName)
  - E.g., 1:5

Testing a loop

- Break down the task
  - Make sure the loop is iterating through your list of options as expected before applying the code you want to execute

```
> for (x in fruits) {
+     print(x)
+ }
[1] "orange"
[1] "apple"
[1] "banana"
[1] "grapefruit"
[1] "starfruit"
```

```
> for (x in fruits) {
+     print(paste0("My favorite fruit is a/an ", x))
+     }
[1] "My favorite fruit is a/an orange"
[1] "My favorite fruit is a/an apple"
[1] "My favorite fruit is a/an banana"
[1] "My favorite fruit is a/an grapefruit"
[1] "My favorite fruit is a/an starfruit"
```

Local environment in the loop

- The placeholder `var` exists only in the loop and is not created as an object in the global environment

- Temporary objects that are created when running code within the curly brackets may also not exist in the global environment

- Commonly need to use the `print()` or `return()` function to bring an output to the global environment

What can break a loop

- Inconsistent formatting in the data
  - Need to tidy the data before operating with a loop
- Errors/typos
- Exceed the amount of available memory

Advance Options

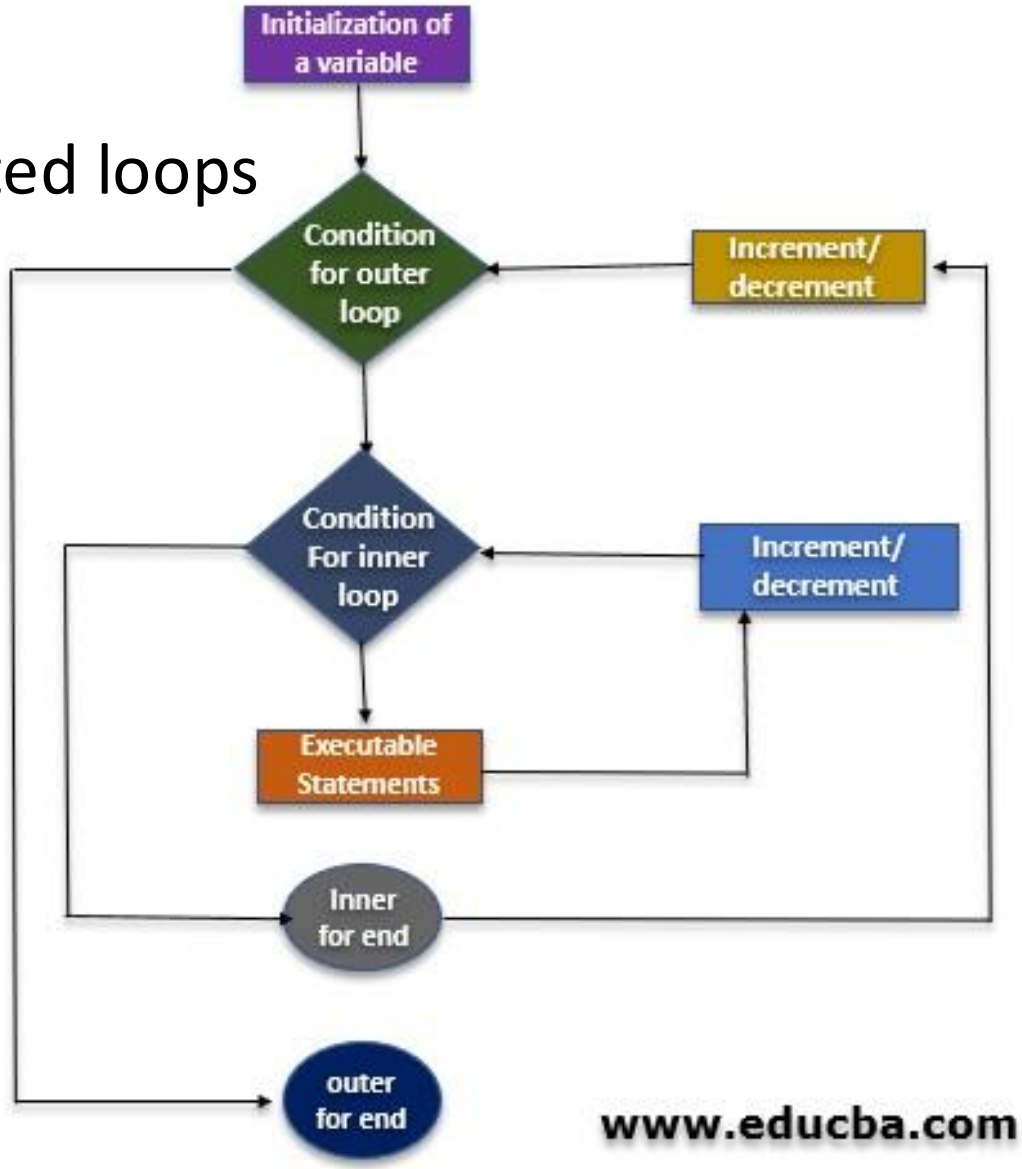
- If else statements

```
if (condition) {
    statement
    statement
       ...
} else {
    statement
    statement
       ...
}
following_statement
```

True branch — *This is executed if the condition is true*

False branch — *This is executed if the condition is false*

- Nested loops



Limitations of For loops

- Relatively computationally expensive
  - Not advised for large tasks
  - Specific functions are often more optimized for larger tasks

- Need to know the number of iterations at the start

Tips for writing For Loops

- Optimize your code before you start moving it into a loop
  - Recommend making it generalized before moving it into a loop as well!
- Try it on a smaller dataset first
  - E.g., object[1:5, 1:5]
- Check if a function already exists for the purpose
- Be cautious if you are overwriting any values/objects

Wrap-up

- When you are repeating a task 3 or more times, it is worth investing time to automate it
  - Generalized code
  - Custom functions
  - For loops

- For loops need to know the full set of values to iterate through at the start

- Start with a simple foundation, test and add more layers of complexity as you build your code