Introduction to R

INFO 201

Today’s Objectives

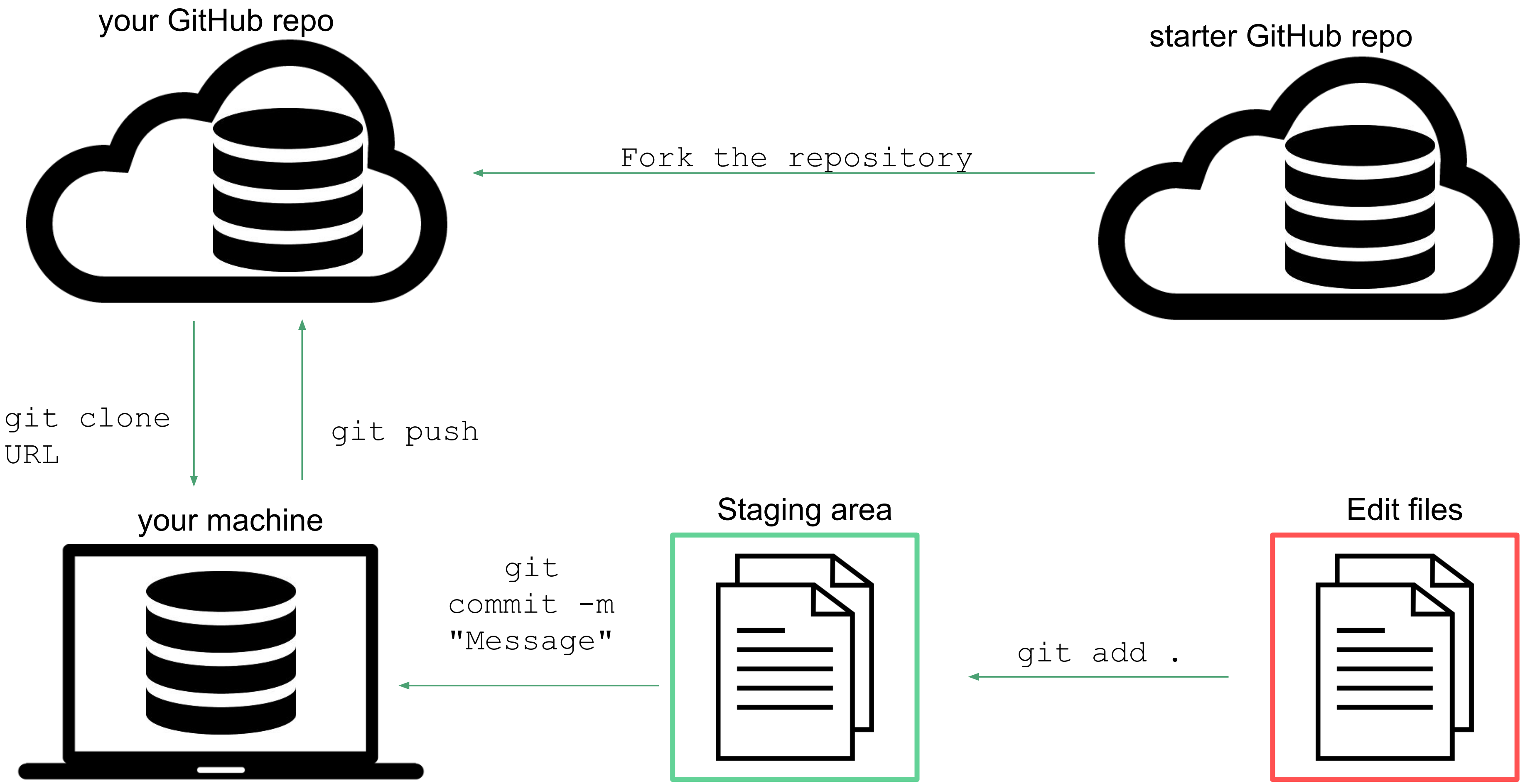
Feel confident with ***version control*** basics

Understand how/where to ***write*** and ***execute*** the R programming language

Explore using and writing your own ***functions*** in R

Practice using the ***vector*** data type in R

Version Control Review



GitHub Process

module-4

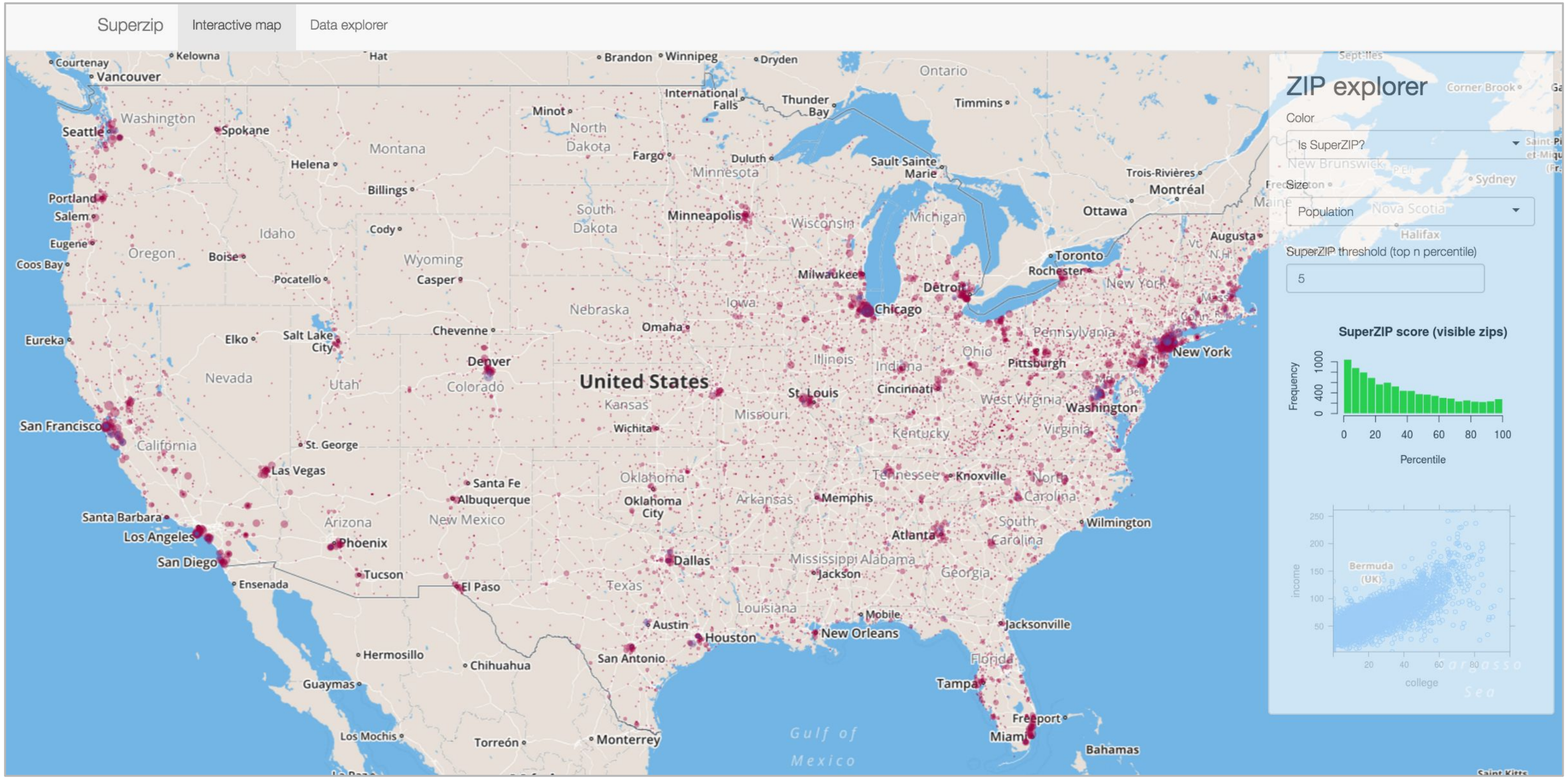
R Basics

Overview

Statistical programming language

Optimized to read, wrangle, analyze, visualize data

Primary language for the course



Color

Is SuperZIP?

Size

Population

SuperZIP threshold (top n percentile)

5

Extensible frameworks for building web applications

Running R

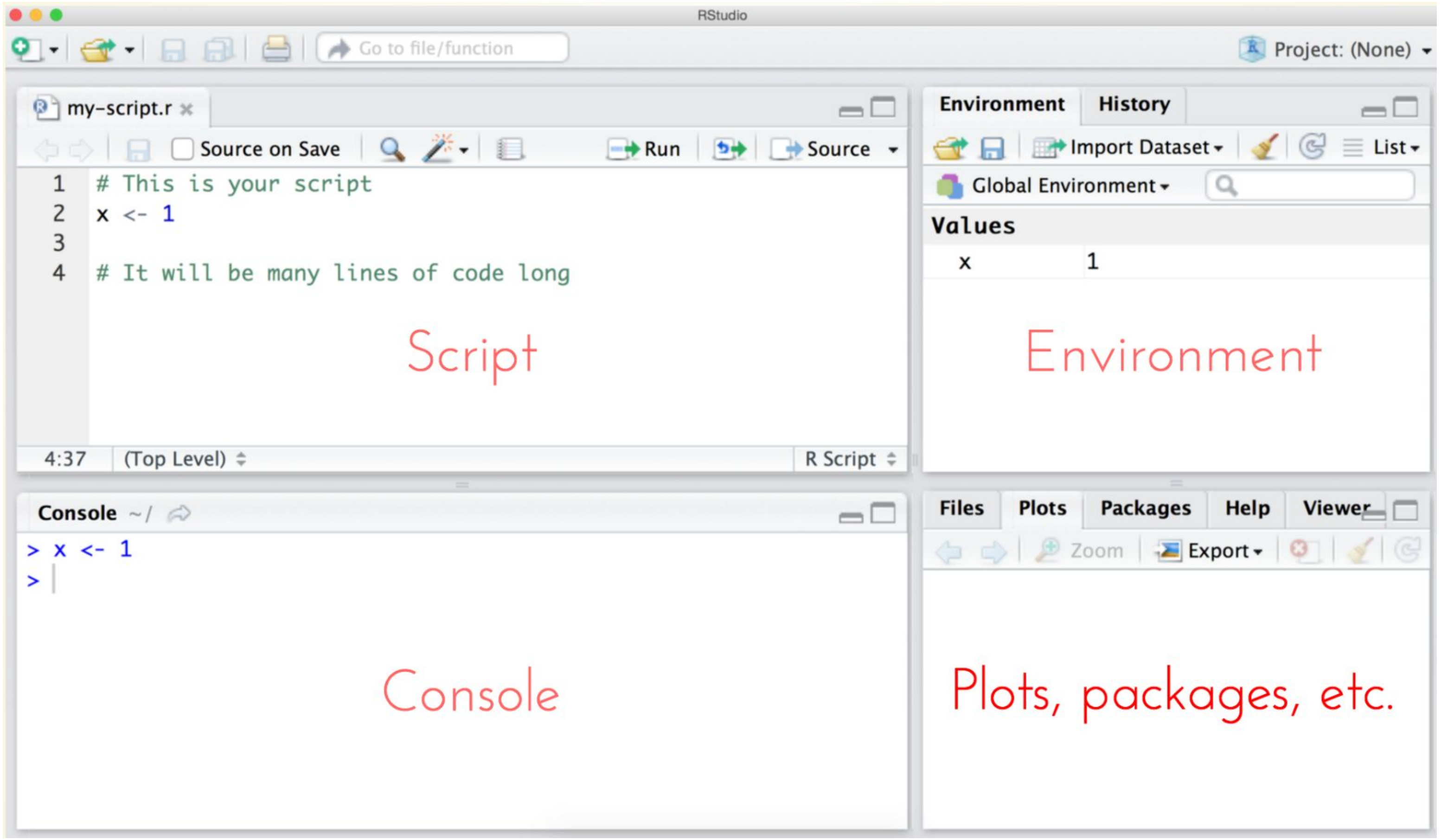
Interactive R session in your terminal

```
    r
```

Tell R to run an R Script (many lines of R code saved in a file)

```
    Rscript filename.r
```

RStudio



```
# This is your script
x <- 1

# It will be many lines of code long
```

```
x    1
```

```
> x <- 1
>
```

RStudio

Basic Syntax

Store information in a variable by assigning a variable that value

```
x <- 6
```

Supports mathematical operations

```
hours.in.a.week <- 24 * 7
```

Describe code with comments (#), which aren't interpreted by R

```
# This is a comment
```

Execute code in RStudio by hitting `ctrl + enter`

module-5

Basic Data Types

Numeric: default computational data type, all real numbers and fractions

```
temperature <- 66.4
```

Character: strings of characters in a variable, made with single or double quotes

```
name <- 'Mike' # "Mike" would be the same
```

Boolean: Logical values, if something is TRUE or FALSE

```
equality <- 3 > 2 # stores TRUE, not in quotes
```

Also ***complex*** and ***integer*** data types, though less often used (see module)

module-5

Module-5 exercise-1

Functions

Functions

An ability that you can leverage throughout a program

An executable block of code that you can use in a variety of contexts

Often accept a number of inputs (*arguments*, *parameters*)

Returns a single output

Built-in Functions

Many functions are built into the R program and can be used out of the box

To execute a function, type the function name, and pass (comma-separated) arguments into parentheses:

```
FunctionName(argument)
```

You can store the results of a function in a variable

```
result <- FunctionName(argument)
```

For example:

```
smallest.number <- min(3, 10, pi) # returns 3
```

module-6

Function Name	Description	Example
`c(a,b, ...)`	Concatenate multiple items into a vector	c(1,2) # returns 1,2
`length(a)`	Determine vector length	length(c(1,2)) # returns 2
`paste(a, b, ...)`	Concatenate characters into one value	paste("Hi", "there") # returns "hi there"
`length(a)`	Determine vector length	length(c(1,2)) # returns 2
`seq(a, b)`	Return a sequence from a to b	seq(1,5) # returns 1, 2, 3, 4, 5
`sum(a, b, ...)`	Calculates the sum of all input values	sum(1,5) # returns 6
`tolower(a)`	Returns the characters in lowercase	tolower("Hi there") # returns "hi there"

Some Handy Functions

Loading Functions

You may want to load functions that someone else has written (the beauty of open source!)

First, you'll need to download an R package (only *once*)

```
    install.packages("stringr")
```

Each time you want to use the package, you'll need to load it into R

```
    library(stringr)
```

Writing Functions

Write utilities that you may want to use throughout a program

Store functions in variables (just like any other value)

Should be named CamelCase without periods (see docs)

```
# Write a function to add two numbers together
AddNumbers <- function(a, b) {
  # Function body: perform tasks in here
  answer <- a + b

  # Return statement: what you want the function to output
  return (answer)
}
```

module-6

Module-6 exercise-1

Vectors

Vectors

Vectors are one dimensional collections of elements of the ***same type***

Elements in a vector are referenced by their position (index) starting at 1

Actually, everything is a vector:

```
x <- "hello" # create x
x            # type x into the console
[1] "hello"
```

Vectors are created by combining elements using the c function

```
colors <- c('yellow', 'blue', 'orange')
```

module-7

Indexing Vectors

Pass in the position (index) inside of square brackets

```
# Create a `colors` vector
colors <- c('yellow', 'blue', 'orange')

# Retrieve the first element
colors[1] # returns 'yellow'
```

Use a **vector of indices**

```
# Create a `colors` vector
colors <- c('yellow', 'blue', 'orange')

# Retrieve the second and third elements from the `colors` vector
colors[c(2, 3)] # returns a vector with 'blue' and 'orange'
```

Indexing Vectors

Using logical (TRUE, FALSE) indices:

```r
# Create a vector of shoe sizes
shoe.sizes <- c(7, 6.5, 4, 11, 8)

# Use a vector of boolean values to retrieve the first, fourth, and fifth elements
shoe.sizes[c(TRUE, FALSE, FALSE, TRUE, TRUE)]  # returns 7, 11, 8

# Better yet, create a boolean vector that indicates if a shoe size is greater than 6.5, then use that
shoe.is.big <- shoe.sizes > 6.5  # returns T, F, F, T, T

# Use the `shoe.is.big` vector to select large shoes
big.shoes <- shoe.sizes[shoe.is.big]  # returns 7, 11, 8

# Even better, do it all at once!
shoe.sizes[shoe.sizes > 6.5]  # returns 7, 11, 8
```

module-7

Module-7 exercise-1

Recycling

Operations in R are vectorized (i.e., applied across the entire vector)

```
# Create a `colors` vector
colors <- c('yellow', 'blue', 'orange')

upper.case <- toupper(letters)# returns a vector YELLOW, BLUE, ORANGE
```

Values get recycled when there are an unequal number of elemnts

```
# Create vectors to combine
v1 <- c(1, 2, 3)
v2 <- c(1, 2)

# Add vectors
v3 <- v1 + v2  # returns (2, 4, 4)
```

Module-7 exercise-2

Upcoming...

By Tuesday: You should feel comfortable with **modules 5 - 7**

Due Tuesday, 10/11 (***before class***): a2-foundational-skills