Stat 111 R Bootcamp

Spring 2019

Roadmap

- What is R?
- Language fundamentals
- Data exploration exercises and office hours

What is R?

- Free language for **statistical computing** and **graphics**
- Built for things statisticians care about:
  - Vectors and matrices
  - Linear/non-linear modeling
  - Statistical tests
- Highly extensible because of **packages** (and you can write your own, too!)

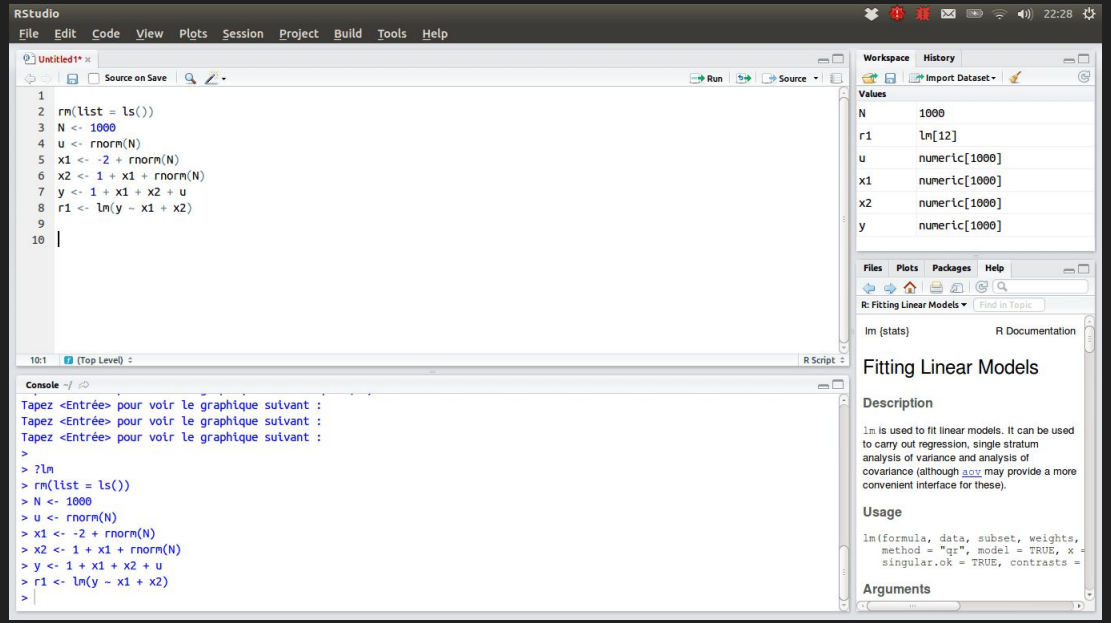
R and Stat 111

- Do not worry if you have no coding experience!
- Goal of this workshop is to build comfort looking things up and solving your own problems in R
- Homeworks may involve computation on real/synthetic data to demonstrate inference in practice
  - Other languages are also allowed (e.g. Python)
- At OH, Course staff will primarily provide conceptual support with computational problems, NOT debugging code

RStudio

An integrated development environment to organize your code, data, plots, files, and more.

You can even type your homeworks using RMarkdown!



Language Fundamentals

- Functions and documentation
- Variables and data types
- Vectors, matrices, data frames
- Loops, conditionals, vectorization
- Plotting

Not covered but worth reading:

- Factors, lists, pipes, apply() function

The Internet has many great R resources. Two that stand out include **DataCamp**, whose Chief Data Scientist was a Harvard stat concentrator, and the free online book **R for Data Science** by Grolemund and Wickham.

Functions and Documentation

Functions take inputs ("arguments") and return an output

sum(110, 111, 211) → 432

Sometimes, arguments are optional or carry a default value.

Exercise: Look up rnorm() and generate 10 i.i.d. N(5, 1) data points. What do pnorm(), dnorm() and qnorm() do?

(Hint: Use ?rnorm to read documentation)

Writing Your Own Functions

You can define your own functions to put repetitive tasks into reusable pieces of code.

```
say_hello <- function(name = "world") {
  # This is a comment, it doesn't affect your code
  # Not sure what cat does? Look it up with ?cat
  # Function will "return" or output the value in its last line
  cat("hello", name)
}
say_hello()
say_hello("Joe and Susan")
```

Variables

Often, we store values in "buckets" called variables. Variables let us represent values without having to retype them, and we can easily update a variable if the value changes.

```
# Assign/update values using the <- operator
mu <- 2.56823912
y <- rnorm(1, mean = mu)
pnorm(y, mean = mu)
mu <- mu + 1
pnorm(y, mean = mu)
print(y)
```

Basic Data Types

Values in R are generally one of the following types:

- Logical: TRUE, FALSE
- Integer: 110, 111
- Decimal: 3.14
- Complex: 3+2i
- Character: "joe", "susan"

Use class() to check a value's type.

Basic Data Types

Careful -- not all data types play well together!

```
# This won't work
"hello" + 1

# This is okay
1 + 3+2i + 2.72

# This is okay, too; logicals are coerced into 1's and 0's
4 + TRUE + FALSE
```

Useful Operators for Numerics

Basic operations: +, -, *, /
Exponents: ^
Modulo: %%
Comparisons: <, <=, >, >=, ==, !=
Other functions: exp(), log(), sin(), cos()

Exercise: Predict results for the following computations

- 110 < 111
- exp(log(-1))
- 4 + 12 / 6 %% 3

- TRUE
- NaN ← Short for "not a number"
- Inf ← Short for infinity

Vectors

Like in math, a 1d collection of elements of the same type.

```r
# Create a vector
my_vector <- vector("numeric", length = 3)
# Combine vectors
c(my_vector, -7, 32)
# Vector arithmetic
my_new_vector + 1
my_new_vector + c(4, 2, 5)
```

Vectors: Naming and Accessing Elements

```r
enrollment <- c(586, 328, 51)
# Give elements names
names(enrollment) <- c("Stat 110", "Stat 111", "Stat 211")
# Access Stat 111 enrollment in a few different ways
enrollment[2]; enrollment["Stat 111"];
enrollment[c(FALSE, TRUE, FALSE)]
```

Notice that R is 1 indexed!

How would you get the enrollment of Stat 110 AND Stat 111?

- enrollment[c(1, 2)]
- enrollment[c("Stat 110", "Stat 111")]
- enrollment[enrollment > 100] (and many other possible ways)

Vectors: Useful Functions

```r
# The colon operator or seq() function yield sequences.
my_vector <- 1:10
my_other_vector <- seq(1, 10, by = 2)

# Other useful functions
mean(my_vector)
var(my_vector)
summary(my_vector)
table(my_vector)
```

Matrices

Like in math, a 2d collection of elements of the same type.

```r
# Create a matrix
my_matrix <- matrix(1:6, nrow = 2, ncol = 3, byrow = TRUE)
# Matrix arithmetic
my_matrix * 2
# Matrix multiplication and transpose
my_matrix %*% t(my_matrix)
```

Matrices: Naming and Accessing Elements

```r
enrollment = matrix(c(586, 620, 328, 355, 51, 44), nrow = 2)
rownames(enrollment) = c("2018", "2019")
colnames(enrollment) = c("Stat 110", "Stat 111", "Stat 211")
```

Predict the following:

- enrollment[1, 2]
- enrollment["2020", "Stat 110"]
- enrollment[2,]

- 328 (first row, second column)
- Subscript out of bounds error
- The entire second row

Matrices: Useful Functions

```r
enrollment = matrix(c(586, 620, 328, 355, 51, 44), nrow = 2)
rownames(enrollment) = c("2018", "2019")
colnames(enrollment) = c("Stat 110", "Stat 111", "Stat 211")

# Merge matrices together; also see rbind()
enrollment = cbind(enrollment, "Stat 139" = c(152, 175))

# You can guess what these do
dim(); nrow(); ncol()
rowSums(); colSums()
```

Data Frames

Like matrices, but columns can be of different types.

```
# Create a data frame
my_df <- data.frame("Dept" = c("Stat", "CS", "Gov"),
 "Enrollment" = c(177, 244, 221))
# More commonly, open a file
my_df <- read.table("filename")
my_df <- read.csv("filename")

# Preview data
head(my_df); tail(my_df); View(my_df); str(my_df)
```

Data Frames

```
my_df <- data.frame("Dept" = c("Stat", "CS", "Gov"),
 "Enrollment" = c(177, 244, 221))
# Select data like matrices. Can also use $ to get a column.
my_df$Enrollment
```

Predict the following:

- subset(my_df, Enrollment > 200)
- my_df[nrow(my_df):1,]
- my_df[sample(nrow(my_df)),]

- Data frame without first row
- Reverse the rows of the df
- Randomly reorder the rows

Conditionals: If … Then … Else ...

```
x <- rnorm(1)
if (x > 0) {
        print("You got a positive number!")
} else {
        print("You got a negative number!")
}
```

Loops: Repeat Chunks of Code

For loop:

```
total <- 0
for (i in 1:10) {
        total <- total + i
}
print(total)
```

While loop:

```
total <- 0
i <- 1
while (i <= 10) {
        total <- total + i
        i <- i + 1
}
print(total)
```

Loops vs. Vectorization

```r
total <- 0
for (i in 1:100) {
    total <- total + 1/i
}
total
```

```r
sum(1/(1:100))
```

It’s often cleaner to avoid loops/conditionals in favor of vectorized functions that apply an operation to an entire vector and return a vector. See the apply() function for more!

Plotting with ggplot2

```
install.packages("ggplot2")
library(ggplot2)
```

- R has built-in plotting called base graphics, which follows a “pen-and-paper” model
- ggplot2 is more modular, layering elements individually
- A handy cheat sheet
- Highly recommended: Chapter 3 of **R for Data Science**

ggplot2: The Grammar of Graphics

```
graphic <- ggplot(data = <DATA>) +
          <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

Explore the mpg data frame with ?mpg and str(mpg).
Then try:

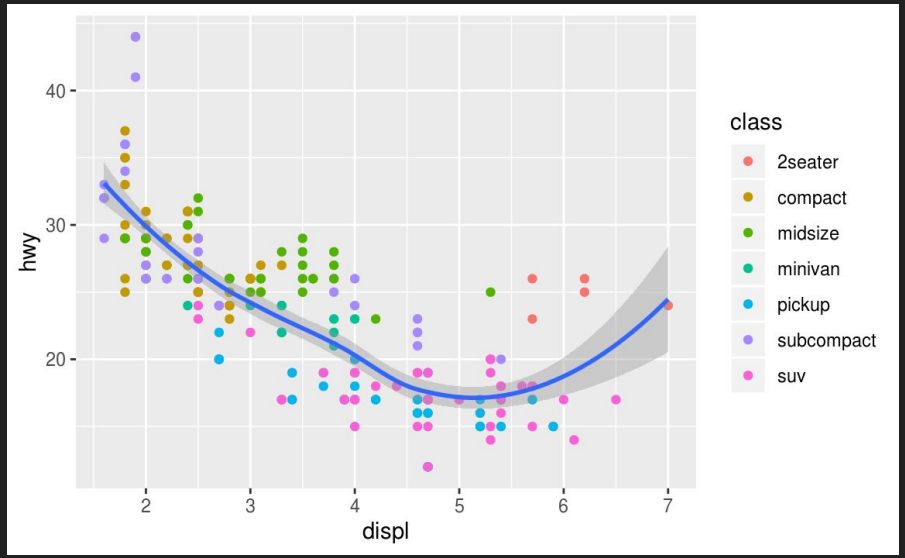
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
```

Then add a color aesthetic for class.
Other aesthetics you can try include shape, size, and alpha.

ggplot2: The Grammar of Graphics

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point(mapping = aes(color = class)) +
  geom_smooth()
```

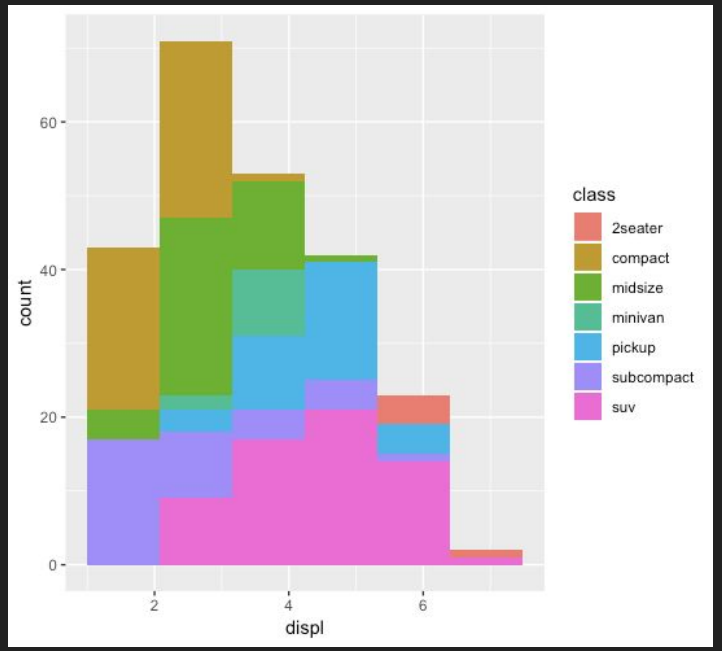
- Layer geoms together
- Mappings can be specified globally or locally



ggplot2: The Grammar of Graphics

```
ggplot(data = mpg, mapping = aes(x = displ, fill = class)) +
    geom_histogram()
```

Look up geom_histogram().
How can I get unit length bins?
How can I make sure they start at 1?



Data Exploration

- Logs of all Blue Bike trips in December 2018
- https://canvas.harvard.edu/files/7228708/
- Attempt the exercises on the handout and ask a TF if you need help!