STAT111 R Bootcamp

Roodmap

- What is R?
- Basics and fundmaentals
- Data exploration exercises and OHs

What is R?

- Open-source language for statistical computing and graphics
- Ranked 12th TIOBE index.
- Built for things statisticians care about:
  - Vectors and matrices
  - Linear/non-linear modeling
  - Statistical tests
  - Visualization and more!
- 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 (IDE) to organize your code, data, plots, files, and more.
- You can even type your homeworks using RMarkdown!

R Resources

- DataCamp, whose Chief Data Scientist was a Harvard stat concentrator
- R for Data Science , the free online book by Grolemund and Wickham.
- Google and Stackoverflow

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

Functions and Documentation

- Functions take inputs (“arguments”) and return an output

```r
sum(110, 111, 211)
```

```
## [1] 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)

Functions and Documentation

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

```
sum(110, 111, 211)
```

```
## [1] 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)

```
?rnorm
rnorm(n = 10, mean = 5, sd = 1)
```

```
##  [1] 3.501596 6.604848 2.839507 5.854525 4.315099 5.285335 6.159325
##  [8] 4.143381 4.706517 3.972850
```

```
pnorm(0.5)
```

```
## [1] 0.6914625
```

```
dnorm(0.5)
```

Writing Your Own Functions

```r
SayHello <- function(name = "world"){
  # This is a comment
  # Use ?cat to see what cat do
  # function will "return" or output the value of the last line
  # also can use "return()"
  cat("hello", name)
}
SayHello()
```

```
## hello world
```

```r
SayHello("Joe and Susan")
```

```
## hello Joe and Susan
```

Variables

- Often store values in "buckets" called variables.
  - Represent values without having to retype them
  - Update a variable if the value changes

```r
# Assign or update using "<-"
# "=" also works
mu <- 2.56
y <- rnorm(1, mean = mu)
pnorm(y, mean = mu)
```

```
## [1] 0.1762332
```

```r
# update
mu <- mu + 1
pnorm(y, mean = mu)
```

```
## [1] 0.02681482
```

```r
print(y)
```

```
## [1] 1.630184
```

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

- 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.
- Careful – not all data types play well together!

```
# This won't work
# "hello" + 1
# This is okay
1 + 3+2i + 2.72
```

```
## [1] 6.72+2i
```

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

```
## [1] 5
```

Useful Operators for Numerics

- Operators
  - Basic operations: +, -, *, /
  - Exponents: ^
  - Modulo: %%
  - Comparisons: <, <=, >, >=, ==, !=
  - Other functions: exp(), log(), sin(), cos()
- **Exercise**: Predict results for the following computations:

Useful Operators for Numerics

- Operators
  - 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
```

Useful Operators for Numerics

- Operators
  - Basic operations: +, -, *, /
  - Exponents: ^
  - Modulo: %%; Quotient: %/%
  - Comparisons: <, <=, >, >=, ==, !=
  - Other functions: exp(), log(), sin(), cos()
- **Exercise**: Predict results for the following computations:

```r
110 < 111
```

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

```r
exp(log(-1))
```

```
## Warning in log(-1): NaNs produced
```

```
## [1] NaN
```

```r
4 + 12 / 6 %% 3
```

```
## [1] Inf
```

Vectors

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

```
# Create a vector
my_vector <- vector("numeric", length = 3)
my_vector
```

```
## [1] 0 0 0
```

```
# my_vector <- numeric(length = 3)
# Combine vectors
c(my_vector, -7, 32)
```

```
## [1]  0  0  0 -7 32
```

```
# Vector arithmetic
my_vector + 1
```

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

```
my_vector + c(4, 2, 5)
```

```
## [1] 4 2 5
```

Vectors: Naming and Accessing Elements

```
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"];
```

```
## Stat 111
##      328
```

```
## Stat 111
##      328
```

```
enrollment[c(FALSE, TRUE, FALSE)]
```

```
## Stat 111
##      328
```

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

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"];
```

```
## Stat 111
##      328
```

```
## Stat 111
##      328
```

```r
enrollment[c(FALSE, TRUE, FALSE)]
```

```
## Stat 111
##      328
```

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

```r
enrollment[c(1, 2)]
```

```
## Stat 110 Stat 111
##      586      328
```

```r
enrollment[c("Stat 110", "Stat 111")]
```

Vectors: Useful Functions

```
# 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)
```

```
## [1] 5.5
```

```
var(my_vector)
```

```
## [1] 9.166667
```

```
summary(my_vector)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    3.25    5.50    5.50    7.75   10.00
```

```
table(my_vector)
```

```
## my_vector
##  1  2  3  4  5  6  7  8  9 10
##  1  1  1  1  1  1  1  1  1  1
```

Matrices

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

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

```
##      [,1] [,2] [,3]
## [1,]    2    4    6
## [2,]    8   10   12
```

```
# Matrix multiplication and transpose
my_matrix %*% t(my_matrix)
```

```
##      [,1] [,2]
## [1,]   14   32
## [2,]   32   77
```

Matrices

Exercise: Try `my_matrix * my_matrix` ?

Matrices

Exercise: Try `my_matrix * my_matrix` ?

```
# Create a matrix
my_matrix * my_matrix
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    9
## [2,]   16   25   36
```

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")
enrollment
```

```
##      Stat 110 Stat 111 Stat 211
## 2018      586      328       51
## 2019      620      355       44
```

* Predict the following

```r
enrollment[1, 2]
enrollment["2020", "Stat 110"]
enrollment[2, ]
```

Matrices: Useful Functions

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

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

Matrices: Useful Functions

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

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

Data Frames

- Like matrices, but columns can be of different types.

```r
# Create a data frame
my_df <- data.frame("Dept" = c("Stat", "CS", "Gov"),
 "Enrollment" = c(177, 244, 221))
```

```r
# More commonly, open a file
my_df <- read.table("filename")
my_df <- read.csv("filename")
```

```r
# Preview data
head(my_df); tail(my_df);
```

```
##   Dept Enrollment
## 1 Stat        177
## 2   CS        244
## 3  Gov        221
```

```
##   Dept Enrollment
## 1 Stat        177
## 2   CS        244
## 3  Gov        221
```

```r
# View(my_df); str(mu_df)
```

Data Frames

```
my_df
```

```
##   Dept Enrollment
## 1 Stat        177
## 2   CS        244
## 3  Gov        221
```

- Predict the following:

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

Data Frames

```
my_df
```

```
##   Dept Enrollment
## 1 Stat        177
## 2   CS        244
## 3  Gov        221
```

- Predict the following:

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

```
##   Dept Enrollment
## 2   CS        244
## 3  Gov        221
```

```
##   Dept Enrollment
## 3  Gov        221
## 2   CS        244
## 1 Stat        177
```

Conditionals: If . . . Then . . . Else . . .

```r
x <- rnorm(1)
x
```

```
## [1] -0.1625652
```

```r
if (x > 0) {
    print("You got a positive number!")
} else {
    print("You got a negative number!")
}
```

```
## [1] "You got a negative number!"
```

Loops: Repeat Chunks of Code

- For loops vs while loops

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

```
## [1] 55
```

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

```
## [1] 55
```

Loops vs. Vectorization

- Vectoriztion:
  - Avoid loops/conditionals
  - Vectorized functions that apply an operation to an entire vector and return a vector.
  - Cleaner and computationally faster!
  - See the apply() function for more!

```r
sum(1:10)
```

```
## [1] 55
```

```r
enrollment
```

```
##      Stat 110 Stat 111 Stat 211 Stat 139
## 2018      586      328       51      152
## 2019      620      355       44      175
```

```r
apply(enrollment, 2, mean)
```

```
## Stat 110 Stat 111 Stat 211 Stat 139
##    603.0    341.5     47.5    163.5
```

Visualization: Plotting with ggplot2

- R has built-in plotting called base graphics, which follows a “pen-and-paper” model
- ggplot2 is more modular, layering elements individually
- Google: “ggglot2 cheatsheet”"
- Highly recommended: Chapter 3 of R for Data Science
- Install ggplot2 package

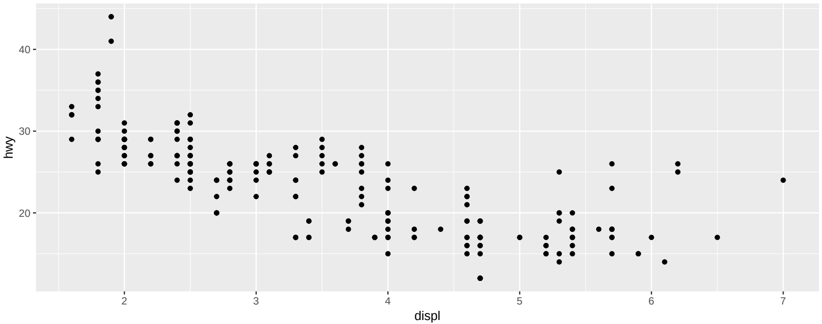
```r
install.packages("ggplot2")
library(ggplot2)
```

ggplot2: The Grammar of Graphics

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

```
# Try explore "mgp" data first
?mgp
str(mpg)
```

```
# scatterplot: engine displacement vs. highway miles per gallon
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
```

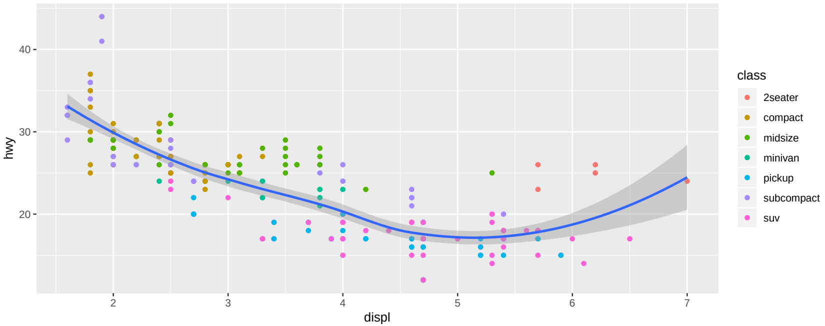


ggplot2: The Grammar of Graphics

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

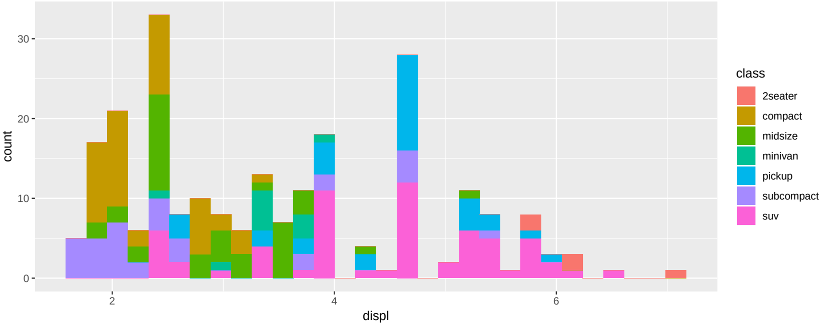
```
# Then add a color aesthetic for class.
# Other aesthetics you can try include shape, size, and alpha.
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point(mapping = aes(color = class)) +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



ggplot2: The Grammar of Graphics

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




```
# look up geom_histogram, adjust binwidth, boundary...
```

Exercises and Data Exploration

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