Intro to R

Rose Hartman

Office of the Vice President for Research and Innovation
Center for Assessment Statistics & Evaluation

May 15th, 2016

Introduction

Hi, I'm Rose.

General R resources

- swirl
  `https://github.com/swirldev/swirl`
- R for cats
  `http://rforcats.net/`
- UCLA R materials
  `http://www.ats.ucla.edu/stat/r/sk/`
- Jenny Bryan's R class
  `http://stat545.com/`
- Advanced R by Hadley Wickham
  `http://adv-r.had.co.nz/`
- R Club!
  `http:blogs.uoregon.edu/rclub`

General R resources

- swirl
  `https://github.com/swirldev/swirl`
- R for cats
  `http://rforcats.net/`
- UCLA R materials
  `http://www.ats.ucla.edu/stat/r/sk/`
- Jenny Bryan's R class
  `http://stat545.com/`
- Advanced R by Hadley Wickham
  `http://adv-r.had.co.nz/`
- R Club!
  `http:blogs.uoregon.edu/rclub`

General R resources

- swirl
  `https://github.com/swirldev/swirl`
- R for cats
  `http://rforcats.net/`
- UCLA R materials
  `http://www.ats.ucla.edu/stat/r/sk/`
- Jenny Bryan's R class
  `http://stat545.com/`
- Advanced R by Hadley Wickham
  `http://adv-r.had.co.nz/`
- R Club!
  `http:blogs.uoregon.edu/rclub`

General R resources

- swirl
  `https://github.com/swirldev/swirl`
- R for cats
  `http://rforcats.net/`
- UCLA R materials
  `http://www.ats.ucla.edu/stat/r/sk/`
- Jenny Bryan's R class
  `http://stat545.com/`
- Advanced R by Hadley Wickham
  `http://adv-r.had.co.nz/`
- R Club!
  `http:blogs.uoregon.edu/rclub`

General R resources

- swirl
  `https://github.com/swirldev/swirl`
- R for cats
  `http://rforcats.net/`
- UCLA R materials
  `http://www.ats.ucla.edu/stat/r/sk/`
- Jenny Bryan's R class
  `http://stat545.com/`
- Advanced R by Hadley Wickham
  `http://adv-r.had.co.nz/`
- R Club!
  `http:blogs.uoregon.edu/rclub`

General R resources

- swirl
  `https://github.com/swirldev/swirl`
- R for cats
  `http://rforcats.net/`
- UCLA R materials
  `http://www.ats.ucla.edu/stat/r/sk/`
- Jenny Bryan's R class
  `http://stat545.com/`
- Advanced R by Hadley Wickham
  `http://adv-r.had.co.nz/`
- R Club!
  `http:blogs.uoregon.edu/rclub`

Welcome to the Wonderful World of R

1. R Basics
   - Overview of R
   - R as a calculator
   - Making and using vectors
   - Upgrading from vectors to data frames!
   - installing a package, reading in a data set
2. Data wrangling
   - dplyr and tidyr
3. Plotting
   - ggplot2

How does R compare to other ways of doing data analysis?

- 100% scripts (100% reproducible): data cleaning, transforming, analyzing, presenting, even writing and interpretation all in one place
- free (open)
- not a spreadsheet environment — this changes the way you think about your data
- handles pretty much anything you might want to do to your data (so no need to switch between environments), and is growing every day
- modular (you can't download the whole thing once and be done)

How does R compare to other ways of doing data analysis?

- 100% scripts (100% reproducible): data cleaning, transforming, analyzing, presenting, even writing and interpretation all in one place
- free (open)
- not a spreadsheet environment — this changes the way you think about your data
- handles pretty much anything you might want to do to your data (so no need to switch between environments), and is growing every day
- modular (you can't download the whole thing once and be done)

How does R compare to other ways of doing data analysis?

- 100% scripts (100% reproducible): data cleaning, transforming, analyzing, presenting, even writing and interpretation all in one place
- free (open)
- not a spreadsheet environment — this changes the way you think about your data
- handles pretty much anything you might want to do to your data (so no need to switch between environments), and is growing every day
- modular (you can't download the whole thing once and be done)

How does R compare to other ways of doing data analysis?

- 100% scripts (100% reproducible): data cleaning, transforming, analyzing, presenting, even writing and interpretation all in one place
- free (open)
- not a spreadsheet environment — this changes the way you think about your data
- handles pretty much anything you might want to do to your data (so no need to switch between environments), and is growing every day
- modular (you can't download the whole thing once and be done)

How does R compare to other ways of doing data analysis?

- 100% scripts (100% reproducible): data cleaning, transforming, analyzing, presenting, even writing and interpretation all in one place
- free (open)
- not a spreadsheet environment — this changes the way you think about your data
- handles pretty much anything you might want to do to your data (so no need to switch between environments), and is growing every day
- modular (you can't download the whole thing once and be done)

How do you learn R?

- It's a language. You learn by using it!
- My recommendation:
  Figure out something you want to do in R (a real, live data problem you have) and work on that, rather than reading about it in the abstract. You won't learn R by having someone explain it to you, only by doing it yourself. Best case scenario: Find a bunch of problems to work on in R, and a group to work on them with, so you get experience with a broad range of applications.

How do you learn R?

- It's a language. You learn by using it!
- My recommendation:
  Figure out something you want to do in R (a real, live data problem you have) and work on that, rather than reading about it in the abstract. You won't learn R by having someone explain it to you, only by doing it yourself. Best case scenario: Find a bunch of problems to work on in R, and a group to work on them with, so you get experience with a broad range of applications.

How do you learn R?

- It's a language. You learn by using it!
- My recommendation:
  Figure out something you want to do in R (a real, live data problem you have) and work on that, rather than reading about it in the abstract. You won't learn R by having someone explain it to you, only by doing it yourself. Best case scenario: Find a bunch of problems to work on in R, and a group to work on them with, so you get experience with a broad range of applications.

This workshop

- Focus on common stumbling blocks for people who try to learn R on their own. I want to take down barriers, so you'll be well placed to continue practicing R back in your natural habitat.
- Lots of practice, and we'll return to key concepts several times.
- For those of you who are learning/psych geeks...
  - interleaved training and progressive alignment
  - realistic practice problems from the two perspectives you're most likely to encounter in real life
  - consistent color-coding of different types of information

This workshop

- Focus on common stumbling blocks for people who try to learn R on their own. I want to take down barriers, so you'll be well placed to continue practicing R back in your natural habitat.
- Lots of practice, and we'll return to key concepts several times.
- For those of you who are learning/psych geeks...
  - interleaved training and progressive alignment
  - realistic practice problems from the two perspectives you're most likely to encounter in real life
  - consistent color-coding of different types of information

This workshop

- Focus on common stumbling blocks for people who try to learn R on their own. I want to take down barriers, so you'll be well placed to continue practicing R back in your natural habitat.
- Lots of practice, and we'll return to key concepts several times.
- For those of you who are learning/psych geeks...
  - interleaved training and progressive alignment
  - realistic practice problems from the two perspectives you're most likely to encounter in real life
  - consistent color-coding of different types of information

This workshop

- Focus on common stumbling blocks for people who try to learn R on their own. I want to take down barriers, so you'll be well placed to continue practicing R back in your natural habitat.
- Lots of practice, and we'll return to key concepts several times.
- For those of you who are learning/psych geeks...
  - interleaved training and progressive alignment
  - realistic practice problems from the two perspectives you're most likely to encounter in real life
  - consistent color-coding of different types of information

This workshop

- Focus on common stumbling blocks for people who try to learn R on their own. I want to take down barriers, so you'll be well placed to continue practicing R back in your natural habitat.
- Lots of practice, and we'll return to key concepts several times.
- For those of you who are learning/psych geeks...
  - interleaved training and progressive alignment
  - realistic practice problems from the two perspectives you're most likely to encounter in real life
  - consistent color-coding of different types of information

This workshop

- Focus on common stumbling blocks for people who try to learn R on their own. I want to take down barriers, so you'll be well placed to continue practicing R back in your natural habitat.
- Lots of practice, and we'll return to key concepts several times.
- For those of you who are learning/psych geeks...
  - interleaved training and progressive alignment
  - realistic practice problems from the two perspectives you're most likely to encounter in real life
  - consistent color-coding of different types of information

Keep an eye out for...

How would you do this?

What will this do?

Keep an eye out for...

How would you do this?

What will this do?

Keep an eye out for...

How would you do this?

What will this do?

Keep an eye out for...

How would you do this?

What will this do?

Learn more: resources to check out

Keep an eye out for...

How would you do this?

What will this do?

Learn more: resources to check out

Key idea: the big ideas you need to hold on to

Keep an eye out for...

How would you do this?

What will this do?

Learn more: resources to check out

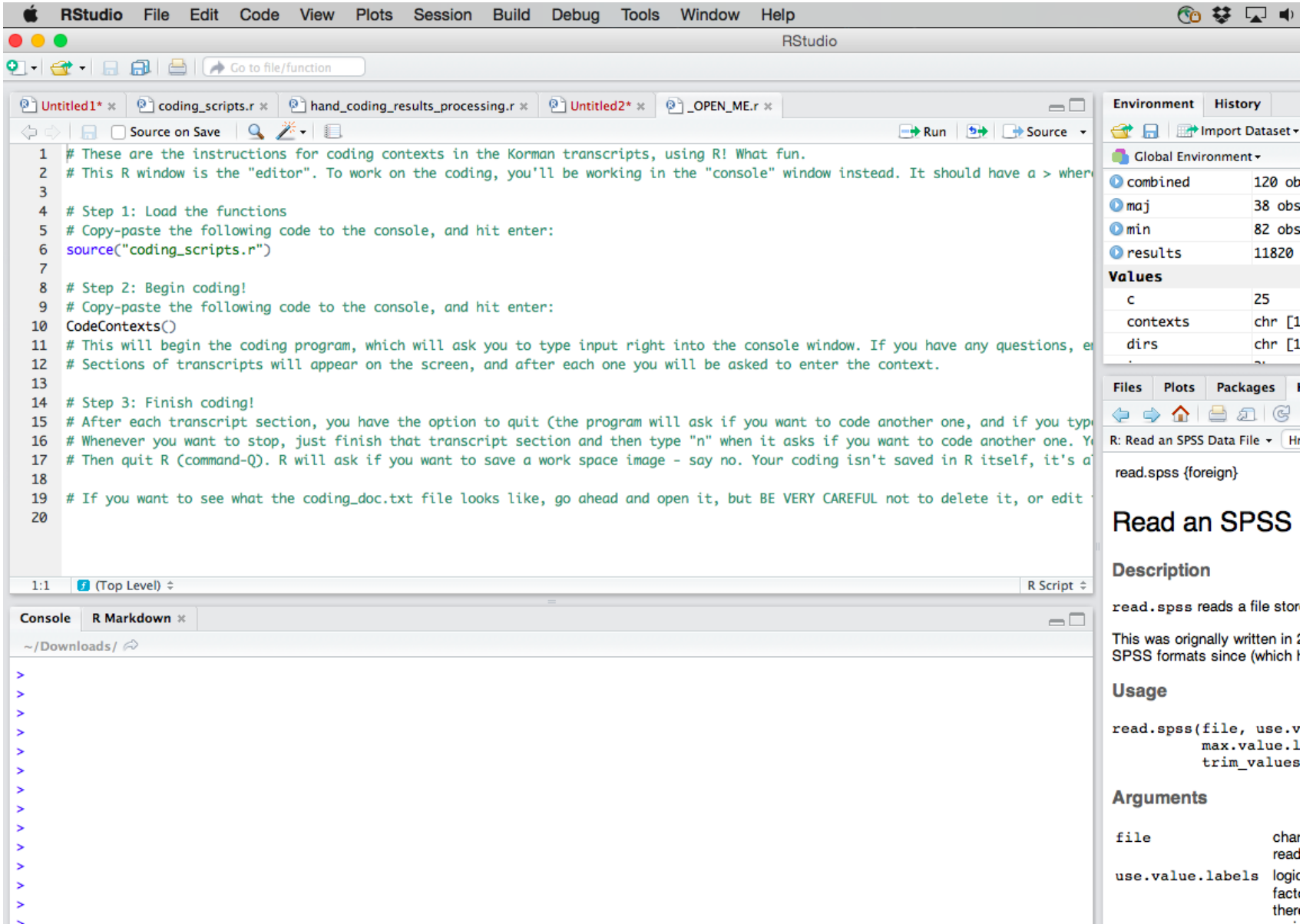
Key idea: the big ideas you need to hold on to

See also: other functions or packages that do a similar thing

Overview

1. R Basics
   - Overview of R
   - R as a calculator
   - Making and using vectors
   - Upgrading from vectors to data frames!
   - installing a package, reading in a data set
2. Data wrangling
   - dplyr and tidyr
3. Plotting
   - ggplot2

Let's get started!



Let's get started!

```
3 + 4
112/2
sqrt(5)
```

Let's get started!

```
3 + 4
112/2
sqrt(5)
```

Key idea: Functions look like do.something(to.this)

Functions in R

```
sum(3, 4)
log(1/2)
sin(0)
sin(1)
sin(pi)
?log
```

Functions in R

```
sum(3, 4)
log(1/2)
sin(0)
sin(1)
sin(pi)
?log
```

Key idea: Pull up the help documentation for a function by ?its.name

Creating objects in R

```
x <- 3
x
```

Creating objects in R

```
x <- 3
x
```

Key idea: If you save something as an object, you can get the value by just calling the object name.

Creating objects in R

```
x <- 3
x
```

Key idea: If you save something as an object, you can get the value by just calling the object name.

```
x + 4
y <- 4
```

Check out your environment in R Studio to see what objects you have currently. You can delete an object with rm(). For example:

```
rm(x)
```

Creating objects in R

```
x <- 3
x
```

Key idea: If you save something as an object, you can get the value by just calling the object name.

```
x + 4
y <- 4
```

Check out your environment in R Studio to see what objects you have currently. You can delete an object with rm(). For example:

```
rm(x)
```

Let's make it again, so we can keep working with it.

```
x <- 3
```

Functions

To understand computations in R, two slogans are helpful:
Everything that exists is an object.
Everything that happens is a function call.
- John Chambers

From `http://adv-r.had.co.nz/Functions.html`

Creating objects in R

What will this do?

```
x + y
```

Let's assign a value to x again.

```
x <- 1
```

What is the value of x?

Creating objects in R

What will this do?

```
x + y
```

Let's assign a value to x again.

```
x <- 1
```

What is the value of x?

Creating objects in R

What will this do?

```
x + y
```

Let's assign a value to x again.

```
x <- 1
```

What is the value of x?

Key idea: When you reuse an object name, it overwrites the old object (with no warning!)

Creating objects in R

```
variablex<- 3
variablex<-3
variablex <-
  3
variable x <- 3
```

Creating objects in R

```
variablex<- 3
variablex<-3
variablex <-
  3
variable x <- 3
```

Key idea: White space usually doesn't matter, except in the middle of a name

Creating objects in R

You can name an object pretty much anything you want (as long as there's no white space).

```
sqrt_5 <- sqrt(5)
pi <- sqrt(5)
```

Creating objects in R

You can name an object pretty much anything you want (as long as there's no white space).

```
sqrt_5 <- sqrt(5)
pi <- sqrt(5)
```

Key idea: Actually, there are some names you can't use (they're reserved).

```
favorite_phrase <- "woo hooo!"
```

Creating objects in R

You can name an object pretty much anything you want (as long as there's no white space).

```
sqrt_5 <- sqrt(5)
pi <- sqrt(5)
```

Key idea: Actually, there are some names you can't use (they're reserved).

```
favorite_phrase <- "woo hooo!"
```

Key idea: Strings need to be surrounded by quotes. If there are no quotes around something, it's treated as a name.

A note about what R needs from you

Implicit contract with the computer / scripting language: Computer will do tedious computation for you. In return, you will be completely precise in your instructions. Typos matter. Case matters. Get better at typing.
- Jenny Bryan

From `http://stat545-ubc.github.io/block002_hello-r-workspace-wd-project.html`

Key ideas from this section: Functions

Key ideas from this section: Functions

Key idea: Functions look like do.something(to.this)

Key ideas from this section: Functions

Key idea: Functions look like do.something(to.this)

Key idea: Pull up the help documentation for a function by ?its.name

Key ideas from this section: Saving objects

Key ideas from this section: Saving objects

Key idea: If you save something as an object, you can get the value by just calling the object name.

Key ideas from this section: Saving objects

Key idea: If you save something as an object, you can get the value by just calling the object name.

Key idea: When you reuse an object name, it overwrites the old object (with no warning!)

Key ideas from this section: Saving objects

Key idea: If you save something as an object, you can get the value by just calling the object name.

Key idea: When you reuse an object name, it overwrites the old object (with no warning!)

Key idea: White space usually doesn't matter, except in the middle of a name

Key ideas from this section: Saving objects

Key idea: If you save something as an object, you can get the value by just calling the object name.

Key idea: When you reuse an object name, it overwrites the old object (with no warning!)

Key idea: White space usually doesn't matter, except in the middle of a name

Key idea: There are some names you can't use (they're reserved).

Key ideas from this section: Saving objects

Key idea: If you save something as an object, you can get the value by just calling the object name.

Key idea: When you reuse an object name, it overwrites the old object (with no warning!)

Key idea: White space usually doesn't matter, except in the middle of a name

Key idea: There are some names you can't use (they're reserved).

Key idea: Strings need to be surrounded by quotes. If there are no quotes around something, it's treated as a name.

Overview

vectors

```
x <- 1:10
```

What will this do?

```
x + 4
```

vectors

```
x <- 1:10
```

What will this do?

```
x + 4
```

Key idea: When you do something to a vector, usually R does it to each element of the vector.

vectors

```
x <- c(1,2,10)
y <- c(3,5,7)
```

What will this do?

```
x + y
```

vectors

```
seq(from=1, to=10, by=1)
```

What will this do?

```
seq(from=1, to=10, by=2)
```

vectors

Try this:

```
seq(1, 10, by=1)
seq(1, 10)
```

What will this do?

```
seq(10, 1)
seq(5)
```

vectors

Try this:

```
seq(1, 10, by=1)
seq(1, 10)
```

What will this do?

```
seq(10, 1)
seq(5)
```

Key idea: Some arguments in functions have defaults

vectors

Try this:

```
seq(1, 10, by=1)
seq(1, 10)
```

What will this do?

```
seq(10, 1)
seq(5)
```

Key idea: Some arguments in functions have defaults

Key idea: You can (but don't have to) specify the names of arguments. I recommend doing it.

vectors

How would you do this? Find out the defaults for seq()

```
?seq
```

vectors

How would you do this? Find out the defaults for seq()

```
?seq
```

vectors

How would you do this? Find out the defaults for seq()

```
?seq
```

Learn more: Use ? when you know the exact name of the function you want, you just want to pull up the help documentation for it.

vectors

How would you do this? Find out the defaults for seq()

```
?seq
```

Learn more: Use ? when you know the exact name of the function you want, you just want to pull up the help documentation for it.

Learn more: Use ?? when you think you've got a function for what you need, you just can't remember its exact name.

vectors

How would you do this?

Generate this vector: 10, 20, 30, 40, 50

```
seq(from=10, to=50, by=10)
```

vectors

How would you do this?
Generate this vector: 10, 20, 30, 40, 50

```
seq(from=10, to=50, by=10)
```

vectors

Try this:

```
rep(3, times=1)
```

What will this do?

```
rep(favorite_phrase, 3)
c(favorite_phrase, 1:5)
```

```
x <- c(favorite_phrase, 1:5)
mode(x)
```

vectors

Try this:

```
rep(3, times=1)
```

What will this do?

```
rep(favorite_phrase, 3)
c(favorite_phrase, 1:5)
```

```
x <- c(favorite_phrase, 1:5)
mode(x)
```

vectors

Try this:

```
rep(3, times=1)
```

What will this do?

```
rep(favorite_phrase, 3)
c(favorite_phrase, 1:5)
```

```
x <- c(favorite_phrase, 1:5)
mode(x)
```

Key idea: You can't combine different types of items in the same vector. If you try, R will coerce them to be the same.

vectors

How would you do this?

Generate this vector: 10, 10, 20, 20, 30, 30

Generate this vector: 10, 20, 30, 10, 20, 30

```
c(10, 10, 20, 20, 30, 30)
c(10, 20, 30, 10, 20, 30)
rep(seq(10, 30, by=10), times=2)
rep(seq(10, 30, by=10), each=2)
sort(rep(seq(10, 30, by=10), 2))
```

vectors

How would you do this?

Generate this vector: 10, 10, 20, 20, 30, 30

Generate this vector: 10, 20, 30, 10, 20, 30

```
c(10, 10, 20, 20, 30, 30)
c(10, 20, 30, 10, 20, 30)
rep(seq(10, 30, by=10), times=2)
rep(seq(10, 30, by=10), each=2)
sort(rep(seq(10, 30, by=10), 2))
```

vectors

How would you do this?
Generate this vector: 10, 10, 20, 20, 30, 30
Generate this vector: 10, 20, 30, 10, 20, 30

```
c(10, 10, 20, 20, 30, 30)
c(10, 20, 30, 10, 20, 30)
rep(seq(10, 30, by=10), times=2)
rep(seq(10, 30, by=10), each=2)
sort(rep(seq(10, 30, by=10), 2))
```

Key idea: Applying functions sequentially. Send the output of one function as the input to the next.

vectors

How would you do this?
Generate this vector: 10, 10, 20, 20, 30, 30
Generate this vector: 10, 20, 30, 10, 20, 30

```
c(10, 10, 20, 20, 30, 30)
c(10, 20, 30, 10, 20, 30)
rep(seq(10, 30, by=10), times=2)
rep(seq(10, 30, by=10), each=2)
sort(rep(seq(10, 30, by=10), 2))
```

Key idea: Applying functions sequentially. Send the output of one function as the input to the next.

Key idea: There's more than one correct solution! That's usually the case.

Key ideas from this section

Key ideas from this section

Key idea: When you do something to a vector, usually R does it to each element of the vector.

Key ideas from this section

Key idea: When you do something to a vector, usually R does it to each element of the vector.

Key idea: Some arguments in functions have defaults

Key ideas from this section

Key idea: When you do something to a vector, usually R does it to each element of the vector.

Key idea: Some arguments in functions have defaults

Key idea: You can (but don't have to) specify the names of arguments. I recommend doing it.

Key ideas from this section

Key idea: When you do something to a vector, usually R does it to each element of the vector.

Key idea: Some arguments in functions have defaults

Key idea: You can (but don't have to) specify the names of arguments. I recommend doing it.

Key idea: Applying functions sequentially. Send the output of one function as the input to the next.

Key ideas from this section

Key idea: When you do something to a vector, usually R does it to each element of the vector.

Key idea: Some arguments in functions have defaults

Key idea: You can (but don't have to) specify the names of arguments. I recommend doing it.

Key idea: Applying functions sequentially. Send the output of one function as the input to the next.

Key idea: There's more than one correct solution! That's usually the case.

Playing with vectors

What will this do?

```
log(seq(from=0, to=10, length.out=100))
```

Playing with vectors

Try this:

```
cor(x,y)
plot(y ~ x)
```

How would you do this? Find out the defaults for plot()

What will this do?

```
plot(x ~ y)
```

Playing with vectors

Try this:

```
rnorm(10)
```

How would you do this? Find out the defaults for rnorm()

How would you do this?
Generate one example of a random IQ score (mean of 100, standard deviation of 15).

```
rnorm(1, mean=100, sd=15)
```

Playing with vectors

Try this:

```
rnorm(10)
```

How would you do this? Find out the defaults for rnorm()

How would you do this?
Generate one example of a random IQ score (mean of 100, standard deviation of 15).

```
rnorm(1, mean=100, sd=15)
```

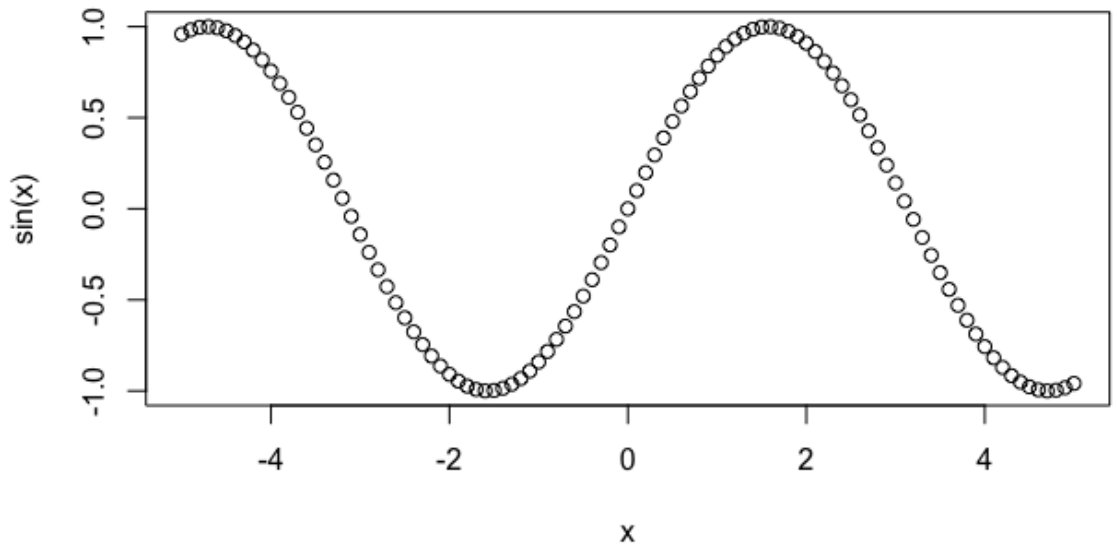
Playing with vectors

What will this do?

```
hist(rnorm(100, mean=50, sd=5))
```

Test your knowledge

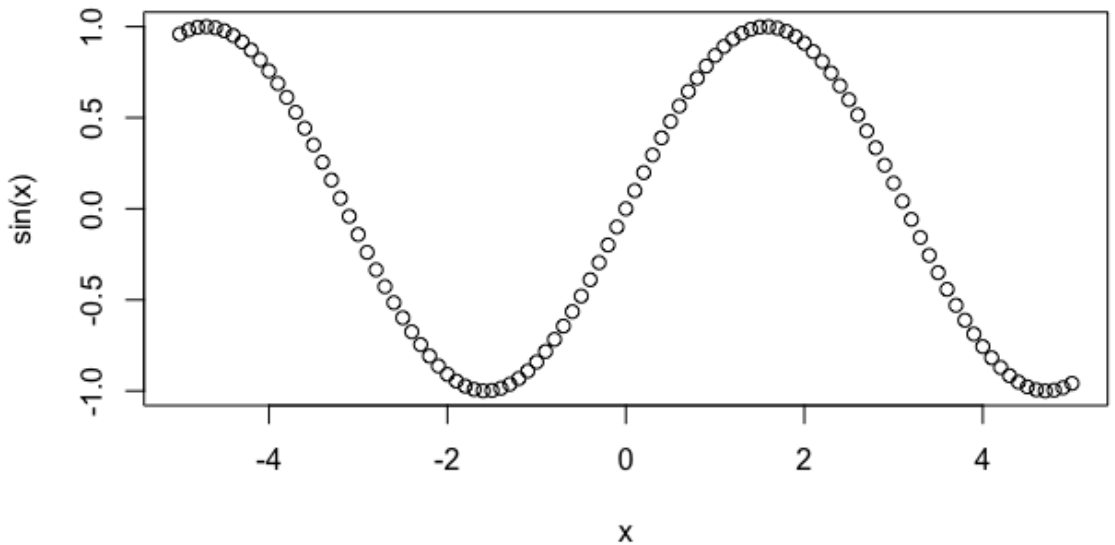
How would you do this?
Generate this plot (showing the value of sin from -5 to 5).




```
x <- seq(from=-5, to=5, by=.1)
plot(sin(x) ~ x)
```

Test your knowledge

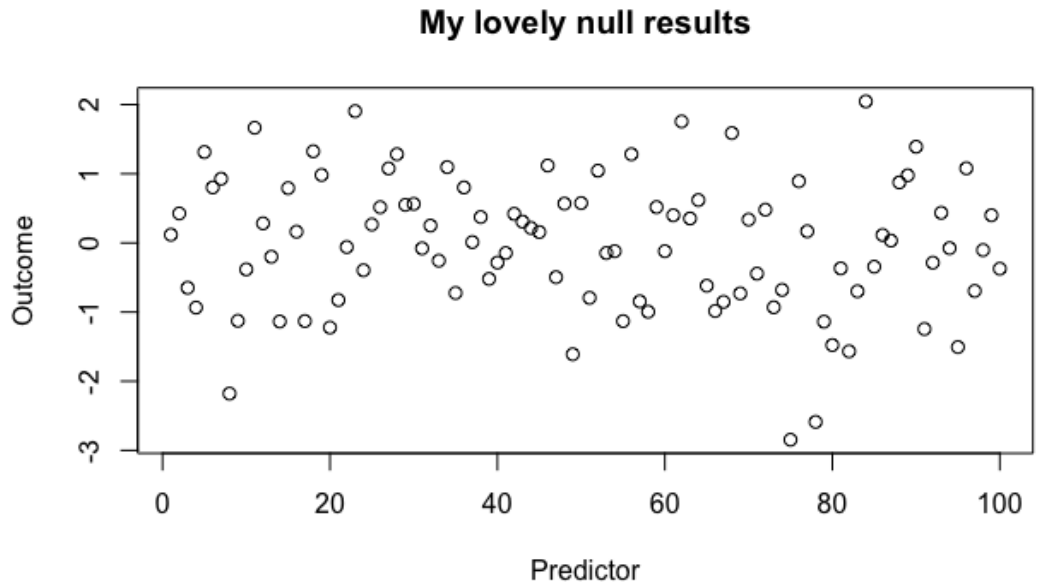
How would you do this?
Generate this plot (showing the value of sin from -5 to 5).




```
x <- seq(from=-5, to=5, by=.1)
plot(sin(x) ~ x)
```

Test your knowledge

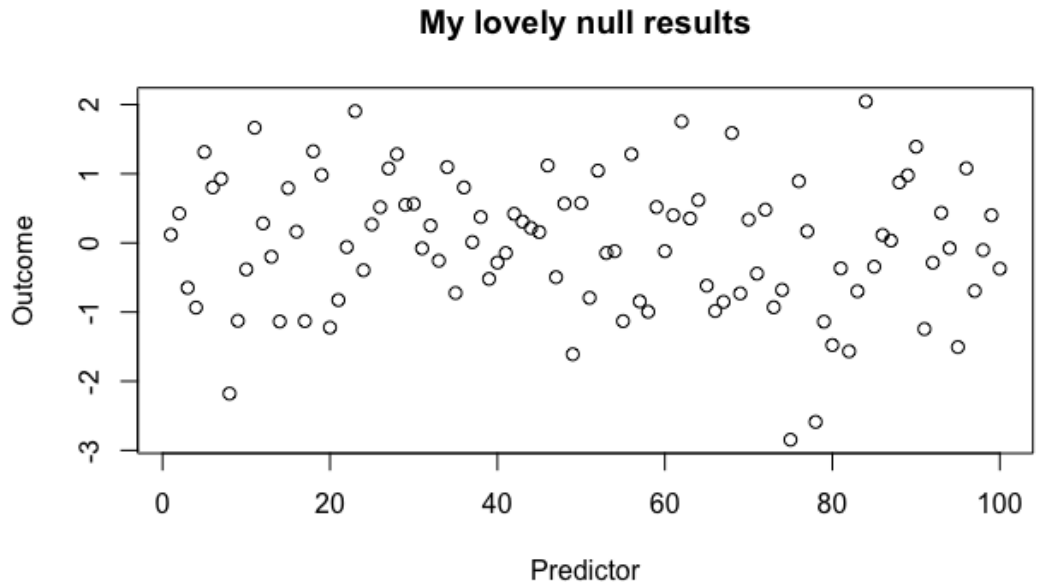
How would you do this? Generate this plot (showing random scatter plot, random x and random y)




```
plot(rnorm(100) ~ rnorm(100), xlab="Predictor", ylab="Outcome",
     main="My lovely null results")
```

Test your knowledge

How would you do this? Generate this plot (showing random scatter plot, random x and random y)




```
plot(rnorm(100) ~ rnorm(100), xlab="Predictor", ylab="Outcome",
     main="My lovely null results")
```

Overview

1. R Basics
   - Overview of R
   - R as a calculator
   - Making and using vectors
   - Upgrading from vectors to data frames!
   - installing a package, reading in a data set
2. Data wrangling
   - dplyr and tidyr
3. Plotting
   - ggplot2

Data frames

Try this:

```
x <- data.frame(id=1:10, scores = rnorm(10))
x
str(x)
```

Data frames

Try this:

```
x <- data.frame(id=1:10, scores = rnorm(10))
x
str(x)
```

Key idea: Use str(something) to learn about its structure.

Data frames

Try this:

```
x <- data.frame(id=1:10, scores = rnorm(10))
x
str(x)
```

Key idea: Use str(something) to learn about its structure.

```
head(x)
tail(x)
View(x)
```

Data frames

Try this:

```
x <- data.frame(id=1:10, scores = rnorm(10))
x
str(x)
```

Key idea: Use str(something) to learn about its structure.

```
head(x)
tail(x)
View(x)
```

Key idea: Use head(), tail() and View() to peak at a data frame.

Data frames

```
conditions <- rep(c("condition 1", "condition 2", "condition 3"),
                  each=5)
```

Data frames

```
conditions <- rep(c("condition 1", "condition 2", "condition 3"),
                  each=5)
```

See also: gl() for generating levels of a categorical variable

```
scores <- runif(15, min=1, max=10)
```

How would you do this?
Learn about a function you don't recognize: runif()

Data frames

```
my_data <- data.frame(conditions, scores)
str(my_data)
```

Data frames

```
my_data <- data.frame(conditions, scores)
str(my_data)
```

Key idea: You can combine different types of variables in the same data frame.

Data frames

```
my_data <- data.frame(conditions, scores)
str(my_data)
```

Key idea: You can combine different types of variables in the same data frame.

Try this:

```
my_data$scores
```

Data frames

```
my_data <- data.frame(conditions, scores)
str(my_data)
```

Key idea: You can combine different types of variables in the same data frame.

Try this:

```
my_data$scores
```

Key idea: Each column in a data frame is a vector.

Data frames

```
my_data <- data.frame(conditions, scores)
str(my_data)
```

Key idea: You can combine different types of variables in the same data frame.

Try this:

```
my_data$scores
```

Key idea: Each column in a data frame is a vector.

Key idea: You can refer to one column within a data frame with $, like dataframe$columnname.

Data frames

Some important things to know about dataframes in R:

- Common types of variables: numeric, factor, character, and logical
- Dataframes must be rectangular (all columns must have the same number of elements).
- Columns have names (i.e. variable names)
- Dataframes, like other R objects, can have additional attributes. You can use this to store additional metadata about variables, etc.

Data frames

Some important things to know about dataframes in R:

- Common types of variables: numeric, factor, character, and logical
- Dataframes must be rectangular (all columns must have the same number of elements).
- Columns have names (i.e. variable names)
- Dataframes, like other R objects, can have additional attributes. You can use this to store additional metadata about variables, etc.

Data frames

Some important things to know about dataframes in R:

- Common types of variables: numeric, factor, character, and logical
- Dataframes must be rectangular (all columns must have the same number of elements).
- Columns have names (i.e. variable names)
- Dataframes, like other R objects, can have additional attributes. You can use this to store additional metadata about variables, etc.

Data frames

Some important things to know about dataframes in R:

- Common types of variables: numeric, factor, character, and logical
- Dataframes must be rectangular (all columns must have the same number of elements).
- Columns have names (i.e. variable names)
- Dataframes, like other R objects, can have additional attributes. You can use this to store additional metadata about variables, etc.

Data frames

Some important things to know about dataframes in R:

- Common types of variables: numeric, factor, character, and logical
- Dataframes must be rectangular (all columns must have the same number of elements).
- Columns have names (i.e. variable names)
- Dataframes, like other R objects, can have additional attributes. You can use this to store additional metadata about variables, etc.

Data frames

Some important things to know about dataframes in R:

- Common types of variables: numeric, factor, character, and logical
- Dataframes must be rectangular (all columns must have the same number of elements).
- Columns have names (i.e. variable names)
- Dataframes, like other R objects, can have additional attributes. You can use this to store additional metadata about variables, etc.

Data frames

Some important things to know about dataframes in R:

- Common types of variables: numeric, factor, character, and logical
- Dataframes must be rectangular (all columns must have the same number of elements).
- Columns have names (i.e. variable names)
- Dataframes, like other R objects, can have additional attributes. You can use this to store additional metadata about variables, etc.

Data frames

Some important things to know about dataframes in R:

- Common types of variables: numeric, factor, character, and logical
- Dataframes must be rectangular (all columns must have the same number of elements).
- Columns have names (i.e. variable names)
- Dataframes, like other R objects, can have additional attributes. You can use this to store additional metadata about variables, etc.

Data frames

There are a bunch of data sets built-in to R:

```
head(iris)
str(iris)
data()
colnames(iris)
summary(iris)
```

Key ideas from this section

Key ideas from this section

Key idea: Use str(something) to learn about its structure.

Key ideas from this section

Key idea: Use str(something) to learn about its structure.

Key idea: Use head(), tail() and View() to peak at a data frame.

Key ideas from this section

Key idea: Use str(something) to learn about its structure.

Key idea: Use head(), tail() and View() to peak at a data frame.

Key idea: You can combine different types of variables in the same data frame.

Key ideas from this section

Key idea: Use str(something) to learn about its structure.

Key idea: Use head(), tail() and View() to peak at a data frame.

Key idea: You can combine different types of variables in the same data frame.

Key idea: Each column in a data frame is a vector.

Key ideas from this section

Key idea: Use str(something) to learn about its structure.

Key idea: Use head(), tail() and View() to peak at a data frame.

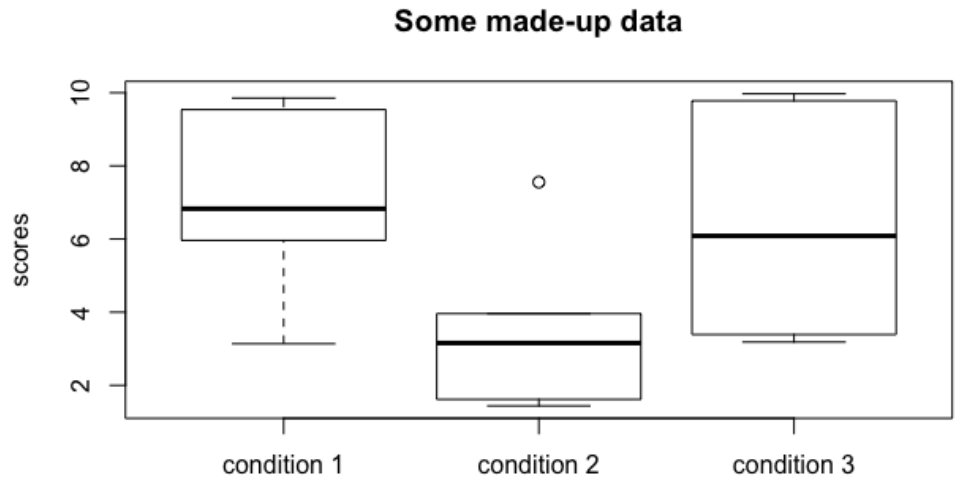
Key idea: You can combine different types of variables in the same data frame.

Key idea: Each column in a data frame is a vector.

Key idea: You can refer to one column within a data frame with $, like dataframe$columnname.

Test your knowledge

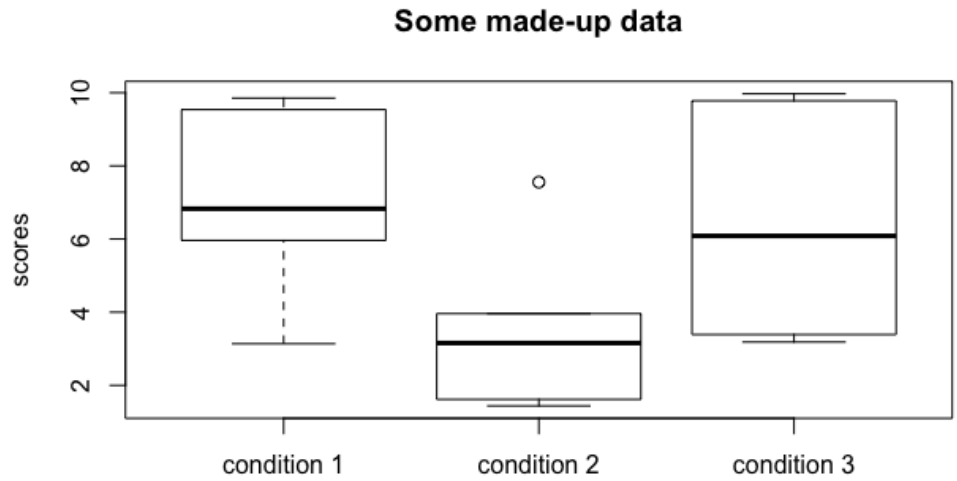
How would you do this?
Generate this plot (plot showing condition on the x-axis and score on the y-axis from my_data)




```
plot(scores ~ conditions, data=my_data, xlab=NULL,
     main="Some made-up data")
```

Test your knowledge

How would you do this?
Generate this plot (plot showing condition on the x-axis and score on the y-axis from my_data)




```
plot(scores ~ conditions, data=my_data, xlab=NULL,
     main="Some made-up data")
```

Data frames

Let's take another look at that code:

```
plot(scores ~ conditions, data=my_data, xlab=NULL,
     main="Some made-up data")
```

Data frames

Let's take another look at that code:

```
plot(scores ~ conditions, data=my_data, xlab=NULL,
     main="Some made-up data")
```

Key idea: Some functions, like plot(), allow you to specify a dataframe, and then you can use the bare column names within the function.

Data frames

Let's take another look at that code:

```
plot(scores ~ conditions, data=my_data, xlab=NULL,
     main="Some made-up data")
```

Key idea: Some functions, like plot(), allow you to specify a dataframe, and then you can use the bare column names within the function.

Key idea: Some functions, like plot(), are clever and will change their behavior depending on the object you give them.

Overview

1. R Basics
   - Overview of R
   - R as a calculator
   - Making and using vectors
   - Upgrading from vectors to data frames!
   - installing a package, reading in a data set
2. Data wrangling
   - dplyr and tidyr
3. Plotting
   - ggplot2

Getting your data into R

```
install.packages("haven")
```

Getting your data into R

```
install.packages("haven")
```

See also: the foreign package

```
library("haven")
```

Getting your data into R

```
install.packages("haven")
```

See also: the foreign package

```
library("haven")
```

See also: require() for loading an installed package

Learning about packages

```
help(package="haven")
```

Learning about packages

```
help(package="haven")
```

Key idea: To pull up all of the documentation for a specific package, you can use the help() command. It's like ? but more flexible.

Functions in packages

```
?read_sav
```

Functions in packages

```
?read_sav
```

See also: foreign::read.spss()

Functions in packages

```
?read_sav
```

See also: foreign::read.spss()

Key idea: To refer to a specific function in a package, use package.name::function.name()

Functions in packages

```
?read_sav
```

See also: foreign::read.spss()

Key idea: To refer to a specific function in a package, use package.name::function.name()

A note about packages

- When you run install.packages(), R contacts CRAN to get the package you want, so you need an internet connection for it to work.
- You only need to install each package once (and again whenever you want updates for it), and it will be saved on your computer so you can use it offline.
- You need to load each package you want to use again in each R session. Why?

A note about packages

- When you run install.packages(), R contacts CRAN to get the package you want, so you need an internet connection for it to work.
- You only need to install each package once (and again whenever you want updates for it), and it will be saved on your computer so you can use it offline.
- You need to load each package you want to use again in each R session. Why?

A note about packages

- When you run install.packages(), R contacts CRAN to get the package you want, so you need an internet connection for it to work.
- You only need to install each package once (and again whenever you want updates for it), and it will be saved on your computer so you can use it offline.
- You need to load each package you want to use again in each R session. Why?

A note about packages

- Note that different packages can have functions with the same name (e.g. select() is a function in several packages)
- packages get updated, and some packages may not continue to work on new versions of R if the package is no longer being supported (this is rare).
- You need to load each package you want to use again in each R session. Why?

A note about packages

- Note that different packages can have functions with the same name (e.g. select() is a function in several packages)
- packages get updated, and some packages may not continue to work on new versions of R if the package is no longer being supported (this is rare).
- You need to load each package you want to use again in each R session. Why?

See also: To manage package versions, check out packrat.

A note about packages

- Note that different packages can have functions with the same name (e.g. select() is a function in several packages)
- packages get updated, and some packages may not continue to work on new versions of R if the package is no longer being supported (this is rare).
- You need to load each package you want to use again in each R session. Why?

See also: To manage package versions, check out packrat.

Key ideas from this section: packages

Key ideas from this section: packages

Key idea: To pull up all of the documentation for a specific package, you can use the help() command. It's like ? but more flexible.

Key ideas from this section: packages

Key idea: To pull up all of the documentation for a specific package, you can use the help() command. It's like ? but more flexible.

Key idea: To refer to a specific function in a package, use package.name::function.name()

Getting your data into R

Does this work for you?

```
atlas <- read_sav("ATLAS.sav")
```

Getting your data into R

When R looks for files on your computer, it doesn't search you're whole computer. It only looks in whatever folder (directory) it's currently in.
R's current folder is called the "working directory."

```
getwd()
```

If you want R to find something on your computer, you have three options:

1. Put the file in R's working directory
2. Move R's working directory to where ever the file is saved
3. Specify the file path when you tell R to look for the file

Getting your data into R

When R looks for files on your computer, it doesn't search you're whole computer. It only looks in whatever folder (directory) it's currently in.
R's current folder is called the "working directory."

```
getwd()
```

If you want R to find something on your computer, you have three options:

1. Put the file in R's working directory
2. Move R's working directory to where ever the file is saved
3. Specify the file path when you tell R to look for the file

Getting your data into R

When R looks for files on your computer, it doesn't search you're whole computer. It only looks in whatever folder (directory) it's currently in.
R's current folder is called the "working directory."

```
getwd()
```

If you want R to find something on your computer, you have three options:

1. Put the file in R's working directory
2. Move R's working directory to where ever the file is saved
3. Specify the file path when you tell R to look for the file

Getting your data into R

Option 1: Put the file in R's working directory.

Option 2: Move R's working directory to where ever the file is saved.
For example, if I want to get a file that's saved on my desktop, I can use setwd() to move R's gaze there:

```
setwd("/Users/TARDIS/Desktop")
```

Option 3: Specify the file path when you tell R to look for the file.

```
atlas <- read_sav("/Users/TARDIS/Desktop/ATLAS.sav")
```

This doesn't change R's working directory.

Getting your data into R

Option 1: Put the file in R's working directory.

Option 2: Move R's working directory to where ever the file is saved.
For example, if I want to get a file that's saved on my desktop, I can use setwd() to move R's gaze there:

```
setwd("/Users/TARDIS/Desktop")
```

Option 3: Specify the file path when you tell R to look for the file.

```
atlas <- read_sav("/Users/TARDIS/Desktop/ATLAS.sav")
```

This doesn't change R's working directory.

Getting your data into R

Option 1: Put the file in R's working directory.

Option 2: Move R's working directory to where ever the file is saved.
For example, if I want to get a file that's saved on my desktop, I can use setwd() to move R's gaze there:

```
setwd("/Users/TARDIS/Desktop")
```

Option 3: Specify the file path when you tell R to look for the file.

```
atlas <- read_sav("/Users/TARDIS/Desktop/ATLAS.sav")
```

This doesn't change R's working directory.

Getting your data into R

Option 1: Put the file in R's working directory.

Option 2: Move R's working directory to where ever the file is saved.
For example, if I want to get a file that's saved on my desktop, I can use setwd() to move R's gaze there:

```
setwd("/Users/TARDIS/Desktop")
```

Option 3: Specify the file path when you tell R to look for the file.

```
atlas <- read_sav("/Users/TARDIS/Desktop/ATLAS.sav")
```

This doesn't change R's working directory.

Key idea: To read or save files, you need to know R's working directory and where the files are located on your computer (or where you want them saved).

Getting your data into R

This is a data set about the efficacy of a strength training self-efficacy intervention on high school student athletes.
I have it stored in my working directory, in a folder called "data".

```
atlas <- read_sav("data/ATLAS.sav")
head(atlas)
str(atlas)
summary(atlas)
```

Getting your data into R

This is a data set about publicly available data, from OSF: https://osf.io/srgjb/

```
osf <- read.csv("data/OSF_data.csv")
head(osf)
str(osf)
summary(osf)
```

Getting your data into R

This is a data set about publicly available data, from OSF: `https://osf.io/srgjb/`

```
osf <- read.csv("data/OSF_data.csv")
head(osf)
str(osf)
summary(osf)
```

Key idea: Always check your work after reading in a data frame. Use str() and View() or head().

Getting your data into R

This is a data set about publicly available data, from OSF: `https://osf.io/srgjb/`

```
osf <- read.csv("data/OSF_data.csv")
head(osf)
str(osf)
summary(osf)
```

Key idea: Always check your work after reading in a data frame. Use str() and View() or head().

```
osf$X <- NULL
str(osf)
```

Getting your data into R

This is a file is from the corpus of infant-directed speech I'm currently analyzing.

```
corpus <- read.table("data/corpus.txt")
```

Getting your data into R

This is a file is from the corpus of infant-directed speech I'm currently analyzing.

```
corpus <- read.table("data/corpus.txt")
```

Key idea: If you specify a file path with no initial /, it's a relative path, meaning R will look for a folder called that *within* its current working directory.

Getting your data into R

This is a file is from the corpus of infant-directed speech I'm currently analyzing.

```
corpus <- read.table("data/corpus.txt")
```

Key idea: If you specify a file path with no initial /, it's a relative path, meaning R will look for a folder called that *within* its current working directory.

```
corpus <- read.table("data/corpus.txt",
                     header=TRUE,
                     sep="\t",
                     quote = NULL,
                     row.names = NULL,
                     stringsAsFactors = FALSE)
```

Getting your data into R

This is a file is from the corpus of infant-directed speech I'm currently analyzing.

```
corpus <- read.table("data/corpus.txt")
```

Key idea: If you specify a file path with no initial /, it's a relative path, meaning R will look for a folder called that *within* its current working directory.

```
corpus <- read.table("data/corpus.txt",
                     header=TRUE,
                     sep="\t",
                     quote = NULL,
                     row.names = NULL,
                     stringsAsFactors = FALSE)
```

Key idea: It may take several tries to read in a file correctly, depending on how it's formatted. Keep checking!

Getting your data into R

This is a file is from the corpus of infant-directed speech I'm currently analyzing.

```
corpus <- read.table("data/corpus.txt")
```

Key idea: If you specify a file path with no initial /, it's a relative path, meaning R will look for a folder called that *within* its current working directory.

```
corpus <- read.table("data/corpus.txt",
                     header=TRUE,
                     sep="\t",
                     quote = NULL,
                     row.names = NULL,
                     stringsAsFactors = FALSE)
```

Key idea: It may take several tries to read in a file correctly, depending on how it's formatted. Keep checking!

Getting your data into R

Do you have your own data? Try reading it into R.

Key ideas: Getting your data into R

Key ideas: Getting your data into R

Key idea: To read or save files, you need to know R's working directory and where the files are located on your computer (or where you want them saved).

Key ideas: Getting your data into R

Key idea: To read or save files, you need to know R's working directory and where the files are located on your computer (or where you want them saved).

Key idea: If you specify a file path with no initial /, it's a relative path, meaning R will look for a folder called that *within* its current working directory.

Key ideas: Getting your data into R

Key idea: To read or save files, you need to know R's working directory and where the files are located on your computer (or where you want them saved).

Key idea: If you specify a file path with no initial /, it's a relative path, meaning R will look for a folder called that *within* its current working directory.

Key idea: Always check your work after reading in a data frame. Use str() and View() or head().

Key ideas: Getting your data into R

Key idea: To read or save files, you need to know R's working directory and where the files are located on your computer (or where you want them saved).

Key idea: If you specify a file path with no initial /, it's a relative path, meaning R will look for a folder called that *within* its current working directory.

Key idea: Always check your work after reading in a data frame. Use str() and View() or head().

Key idea: It may take several tries to read in a data file correctly, depending on how it's formatted. Keep checking to see if the read-in object looks correct!

Overview

1. R Basics
   - Overview of R
   - R as a calculator
   - Making and using vectors
   - Upgrading from vectors to data frames!
   - installing a package, reading in a data set
2. Data wrangling
   - dplyr and tidyr
3. Plotting
   - ggplot2

Working with data in R

R has some built-in data sets, and there are also often additional data sets included with packages. Let's install the gapminder package to get those data.

How would you do this?
Install a new package called "gapminder"

Working with data in R

R has some built-in data sets, and there are also often additional data sets included with packages. Let's install the gapminder package to get those data.

How would you do this?
Install a new package called "gapminder"

```
install.packages("gapminder")
library("gapminder")
help(package="gapminder")
```

From `http://www.gapminder.org/data/`

Working with data in R

Let's take a look at the gapminder data set:

```
head(gapminder)
str(gapminder)
summary(gapminder)
```

Working with data in R

Let's take a look at the gapminder data set:

```
head(gapminder)
str(gapminder)
summary(gapminder)
```

dplyr

As anyone who's conducted a full analysis knows, it's only actually a tiny portion of your analysis time that's spent doing stats. The rest is data cleaning / processing / reformatting / wrangling.
There are lots of great ways to manipulate data in R (watch for "See also" boxes). I'm only going to focus on teaching you one of them: dplyr, and its companion tidyr.
The reason is that dplyr is specifically designed for exactly what most researchers need: to handle all of the most common data wrangling tasks in a streamlined, intuitive way. It's fast and slick, and it integrates beautifullly with great plotting tools, so you can get to the exciting part of your analysis faster.

dplyr

The dplyr package is a little unusual in that it's not just a collection of functions that you're supposed to throw in with all of the rest of the functions you've got in R; it's designed to work as a cohesive set to meet pretty much all of your data manipulation needs. Rather than just having dplyr functions sprinkled throughout a script of yours, you're likely to have dplyr sections within the script, where you use several related functions in a row. It's mostly made up of functions that work like "verbs" for modifying your data. Also, you also have the option to use pipes if you want, to make your data cleaning steps easier to read (and write).

dplyr

The dplyr package is a little unusual in that it's not just a collection of functions that you're supposed to throw in with all of the rest of the functions you've got in R; it's designed to work as a cohesive set to meet pretty much all of your data manipulation needs. Rather than just having dplyr functions sprinkled throughout a script of yours, you're likely to have dplyr sections within the script, where you use several related functions in a row. It's mostly made up of functions that work like "verbs" for modifying your data. Also, you also have the option to use pipes if you want, to make your data cleaning steps easier to read (and write).

Learn more: Here's an introduction to dplyr (with lots of examples!) by Hadley Wickham himself `https://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html`

dplyr

```
install.packages("dplyr", "tidyr")
library("dplyr")
library("tidyr")
help(package="dplyr")
```

Tons of functions!

dplyr

Take some rows from the data frame, according to some rule.
For example, let's get all of the Asian countries from gapminder.

```
?filter
```

dplyr

Take some rows from the data frame, according to some rule.
For example, let's get all of the Asian countries from gapminder.

```
?filter
```

```
gapminder.asia <- filter(gapminder, continent=="Asia")
```

dplyr

Take some rows from the data frame, according to some rule.
For example, let's get all of the Asian countries from gapminder.

```
?filter
```

```
gapminder.asia <- filter(gapminder, continent=="Asia")
```

Key idea: To test for equality (are these two things the same?), you need to use ==

How would you do this? Just pull out the cases with less than average life expectancy

dplyr

How would you do this? Just pull out the cases with less than average life expectancy

```
filter(gapminder, lifeExp < mean(lifeExp))
```

dplyr

How would you do this? Just pull out the cases with less than average life expectancy

```
filter(gapminder, lifeExp < mean(lifeExp))
```

Key idea: You can use $>, <, >=, <=$ to express inequalities

dplyr

How would you do this? Just pull out the cases in the Americas with less than average life expectancy

Hint: Look at the help documentation for dplyr::filter

dplyr

How would you do this? Just pull out the cases in the Americas with less than average life expectancy

Hint: Look at the help documentation for dplyr::filter

```
filter(gapminder, lifeExp < mean(lifeExp) & continent == "Americas")
```

dplyr

How would you do this? Just pull out the cases in the Americas with less than average life expectancy

Hint: Look at the help documentation for dplyr::filter

```
filter(gapminder, lifeExp < mean(lifeExp) & continent == "Americas")
```

Key idea: You can use & to specify two (or more) logical tests that all need to be met.

dplyr

How would you do this? Just pull out the cases in the Americas with less than average life expectancy

Hint: Look at the help documentation for dplyr::filter

```
filter(gapminder, lifeExp < mean(lifeExp) & continent == "Americas")
```

Key idea: You can use & to specify two (or more) logical tests that all need to be met.

See also: base::subset()

How would you do this? Just pull out the cases with particularly low (less than 40) or particularly high (greater than 75) life expectancies.

dplyr

How would you do this? Just pull out the cases with particularly low (less than 40) or particularly high (greater than 75) life expectancies.

```
filter(gapminder, lifeExp < 40 | lifeExp > 75)
```

dplyr

How would you do this? Just pull out the cases with particularly low (less than 40) or particularly high (greater than 75) life expectancies.

```
filter(gapminder, lifeExp < 40 | lifeExp > 75)
```

Key idea: You can use | to specify "or" logical tests, where any of them can be met.

dplyr

Selecting some of the columns.

```
select(gapminder, country, continent, year, lifeExp)
```

tidyr

Converting between long and wide formats. Use spread() to go wide, and gather() to go long.
Let's say we want each row to just be one year and have each country be a column, so we can look at the life expectancy data in wide format.

```
gapminder.lifeExp <- select(gapminder,
                            country,
                            year,
                            lifeExp)
wide <- spread(gapminder.lifeExp, key=country, lifeExp)
wide
```

tidyr

Converting between long and wide formats. Use spread() to go wide, and gather() to go long.
Let's say we want each row to just be one year and have each country be a column, so we can look at the life expectancy data in wide format.

```
gapminder.lifeExp <- select(gapminder,
                            country,
                            year,
                            lifeExp)
wide <- spread(gapminder.lifeExp, key=country, lifeExp)
wide
```

See also: reshape2::melt and reshape2::cast

tidyr

Converting between long and wide formats. Use spread() to go wide, and gather() to go long.
Now let's bring it back to long format.

```
long <- gather(wide,
               key="country",
               value="lifeExp",
               Afghanistan:Zimbabwe)
long
```

tidyr

```
help(package="tidyr")
```

Tons of great stuff!

dplyr

But we can do this much more neatly (without creating a lot of temporary objects) by using pipes.

```
gapminder %>%
  select(country, year, lifeExp) %>%
  spread(key=country, lifeExp)
```

dplyr

But we can do this much more neatly (without creating a lot of temporary objects) by using pipes.

```
gapminder %>%
  select(country, year, lifeExp) %>%
  spread(key=country, lifeExp)
```

Key idea: A pipe sends the output of one function to be the first argument in the next function.

dplyr

But we can do this much more neatly (without creating a lot of temporary objects) by using pipes.

```
gapminder %>%
  select(country, year, lifeExp) %>%
  spread(key=country, lifeExp)
```

Key idea: A pipe sends the output of one function to be the first argument in the next function.

Key idea: When you use pipes, you omit the first argument of the function after the pipe.

dplyr

Doing things to each of the levels of a factor in a data frame. For example, maybe you want the mean population for each country, collapsing across years.

```
gapminder %>%
  group_by(country)
```

```
gapminder %>%
  group_by(country) %>%
  summarize(mean.pop=mean(pop))
```

dplyr

Doing things to each of the levels of a factor in a data frame. For example, maybe you want the mean population for each country, collapsing across years.

```
gapminder %>%
  group_by(country)
```

```
gapminder %>%
  group_by(country) %>%
  summarize(mean.pop=mean(pop))
```

dplyr

Doing things to each of the levels of a factor in a data frame. For example, maybe you want the mean population for each country, collapsing across years.

```
gapminder %>%
  group_by(country)
```

```
gapminder %>%
  group_by(country) %>%
  summarize(mean.pop=mean(pop))
```

See also: base::by() and psych::describeBy()

How would you do this?

Get the min and max GDP per capita for each continent

dplyr

How would you do this?
Get the min and max GDP per capita for each continent

```
gapminder %>%
  group_by(continent) %>%
  summarize(min.gpd=min(gdpPercap), max.gpd=max(gdpPercap))
```

dplyr

How would you do this?
Get the min and max GDP per capita for each continent

```
gapminder %>%
  group_by(continent) %>%
  summarize(min.gpd=min(gdpPercap), max.gpd=max(gdpPercap))
```

What will this do?
If you ran summarize() without grouping first

```
gapminder %>%
  summarize(min.gpd=min(gdpPercap), max.gpd=max(gdpPercap))
```

dplyr

Combining data frames: join operations.

```
?join
```

left_join keeps all of the rows in whichever dataframe you list first, and will supplement it with columns from the second dataframe.

dplyr

Combining data frames: join operations.

```
?join
```

left_join keeps all of the rows in whichever dataframe you list first, and will supplement it with columns from the second dataframe.

```
cont.means <- gapminder %>%
  group_by(continent, year) %>%
  summarize(cont.gdpPercap=mean(gdpPercap))
gapminder %>%
  left_join(cont.means, by=c("continent", "year"))
```

dplyr

Combining data frames: join operations.

```
?join
```

left_join keeps all of the rows in whichever dataframe you list first, and will supplement it with columns from the second dataframe.

```
cont.means <- gapminder %>%
  group_by(continent, year) %>%
  summarize(cont.gdpPercap=mean(gdpPercap))
gapminder %>%
  left_join(cont.means, by=c("continent", "year"))
```

See also: base::merge()

dplyr

Combining data frames: join operations.

```
?join
```

left_join keeps all of the rows in whichever dataframe you list first, and will supplement it with columns from the second dataframe.

```
cont.means <- gapminder %>%
  group_by(continent, year) %>%
  summarize(cont.gdpPercap=mean(gdpPercap))
gapminder %>%
  left_join(cont.means, by=c("continent", "year"))
```

See also: base::merge()

Learn more:
http://stat545.com/bit001_dplyr-cheatsheet.html

dplyr

Add new calculated columns.

```
gapminder %>%
  mutate(pop.diff=pop - mean(pop), pop.sd=sd(pop), pop.Z=pop.diff/pop.s
```

Key ideas from this section

Key ideas from this section

Key idea: To test for equality (are these two things the same?), you need to use ==

Key ideas from this section

Key idea: To test for equality (are these two things the same?), you need to use $==$

Key idea: You can use $>, <, >=, <=$ to express inequalities

Key ideas from this section

Key idea: To test for equality (are these two things the same?), you need to use $==$

Key idea: You can use $>, <, >=, <=$ to express inequalities

Key idea: You can use & to specify two (or more) logical tests that all need to be met.

Key ideas from this section

Key idea: To test for equality (are these two things the same?), you need to use $==$

Key idea: You can use $>, <, >=, <=$ to express inequalities

Key idea: You can use & to specify two (or more) logical tests that all need to be met.

Key idea: You can use | to specify "or" logical tests, where any of them can be met.

Key ideas from this section

Key idea: To test for equality (are these two things the same?), you need to use $==$

Key idea: You can use $>, <, >=, <=$ to express inequalities

Key idea: You can use & to specify two (or more) logical tests that all need to be met.

Key idea: You can use | to specify "or" logical tests, where any of them can be met.

Key idea: A pipe sends the output of one function to be the first argument in the next function.

Key ideas from this section

Key idea: To test for equality (are these two things the same?), you need to use $==$

Key idea: You can use $>, <, >=, <=$ to express inequalities

Key idea: You can use & to specify two (or more) logical tests that all need to be met.

Key idea: You can use | to specify "or" logical tests, where any of them can be met.

Key idea: A pipe sends the output of one function to be the first argument in the next function.

Key idea: When you use pipes, you omit the first argument of the function after the pipe.

Test your knowledge

How would you do this?

For each country at each year, calculate its difference in population from its continent's average population during that year.

Test your knowledge

How would you do this?
For each country at each year, calculate its difference in population from its continent's average population during that year.

```
cont.means <- gapminder %>%
  group_by(continent, year) %>%
  summarize(cont.pop=mean(pop))

gapminder %>%
  left_join(cont.means, by=c("continent", "year")) %>%
  mutate(diff.pop=pop-cont.pop)
```

Test your knowledge

How would you do this?

Get the average life expectancy and population over the last 10 years of the data (1997-2007) for each continent.

Test your knowledge

How would you do this?
Get the average life expectancy and population over the last 10 years of the data (1997-2007) for each continent.

```
gapminder %>%
  filter(year >= 1997) %>%
  group_by(continent) %>%
  summarize(mean(lifeExp), mean(pop))
```

Test your knowledge

What will this do?
Can you understand this code? Use ? to look up functions you don't recognize.

```
gapminder %>%
  filter(continent == "Asia") %>%
  select(year, country, lifeExp) %>%
  arrange(year) %>%
  group_by(year) %>%
  filter(min_rank(desc(lifeExp)) < 2 | min_rank(lifeExp) < 2)
```

Test your knowledge

What will this do?
Can you understand this code? Use ? to look up functions you don't recognize.

```
gapminder %>%
  group_by(continent, country) %>%
  select(country, year, continent, lifeExp) %>%
  mutate(le_delta = lifeExp - lag(lifeExp)) %>%
  summarize(worst_le_delta = min(le_delta, na.rm = TRUE)) %>%
  filter(min_rank(worst_le_delta) < 2) %>%
  arrange(worst_le_delta)
```

Overview

1. R Basics
   - Overview of R
   - R as a calculator
   - Making and using vectors
   - Upgrading from vectors to data frames!
   - installing a package, reading in a data set
2. Data wrangling
   - dplyr and tidyr
3. Plotting
   - ggplot2

Plotting in R

Lots of options for plotting in R. We've already seen some examples of the base R plotting functions, and they work great. The ggplot2 package is different because it's specifically designed with data exploration in mind — not just what plots you'll need to draw, but what questions you'll need to answer about your data.

When might you want to use base plotting, and when might ggplot2 be better?

Plotting in R

Lots of options for plotting in R. We've already seen some examples of the base R plotting functions, and they work great. The ggplot2 package is different because it's specifically designed with data exploration in mind — not just what plots you'll need to draw, but what questions you'll need to answer about your data.

When might you want to use base plotting, and when might ggplot2 be better?

Learn more:
`https://github.com/jennybc/ggplot2-tutorial`

ggplot2

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

ggplot2

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

Learn more: Lots and lots of example plots with code
http://shinyapps.org/apps/RGraphCompendium/index.php
and http://shiny.stat.ubc.ca/r-graph-catalog/

ggplot2

ggolot2 works in layers. The first thing you do is set up the variables you want to plot in ggplot(), then you add layers for the plots you want to draw.

```
ggplot(gapminder, aes(x=year, y=pop))
```

ggplot2

ggolot2 works in layers. The first thing you do is set up the variables you want to plot in ggplot(), then you add layers for the plots you want to draw.

```
ggplot(gapminder, aes(x=year, y=pop))
```

Key idea: ggplot() and aes() set up what data will get plotted. Then you tell it how to plot with geoms.

ggplot2

Scatterplot

```
ggplot(gapminder, aes(x=year, y=pop)) +
  geom_point()
```

You can make the points semi-transparent to ameliorate issues from overplotting (alpha=0 is invisible, alpha=1 is full strength):

```
ggplot(gapminder, aes(x=year, y=pop)) +
  geom_point(alpha=.3)
```

ggplot2

Scatterplot

```
ggplot(gapminder, aes(x=year, y=pop)) +
  geom_point()
```

You can make the points semi-transparent to ameliorate issues from overplotting (alpha=0 is invisible, alpha=1 is full strength):

```
ggplot(gapminder, aes(x=year, y=pop)) +
  geom_point(alpha=.3)
```

ggplot2

Boxplot

```
ggplot(gapminder, aes(x=continent, y=lifeExp)) +
  geom_boxplot()
```

ggplot2

Histogram

```
ggplot(gapminder, aes(x=lifeExp)) +
  geom_histogram()
```

ggplot2

Barplot

```
ggplot(gapminder, aes(x=continent, y=lifeExp)) +
  geom_bar()
```

ggplot2

Barplot

```
ggplot(gapminder, aes(x=continent, y=lifeExp)) +
  geom_bar()
```

geom_barplot is really built for counting up numbers of cases, rather than displaying means (geom_boxplot is designed for that). But you can get a barplot, too, you just need to do some data wrangling first.

ggplot2

Barplot

```
ggplot(gapminder, aes(x=continent, y=lifeExp)) +
  geom_bar()
```

geom_barplot is really built for counting up numbers of cases, rather than displaying means (geom_boxplot is designed for that). But you can get a barplot, too, you just need to do some data wrangling first.

Key idea: You should have your data manipulation (e.g. getting summary stats) done before you try to get ggplot to plot it.

ggplot2

Let's plot the mean lifeExp for each continent.

```
plot.data <- gapminder %>%
  group_by(continent) %>%
  summarize(mean.lifeExp=mean(lifeExp),
            sd=sd(lifeExp),
            n=n(),
            se=sd/sqrt(n))

ggplot(plot.data, aes(x=continent, y=mean.lifeExp)) +
  geom_bar(stat="identity")
```

ggplot2

Let's plot the mean lifeExp for each continent.

```
plot.data <- gapminder %>%
  group_by(continent) %>%
  summarize(mean.lifeExp=mean(lifeExp),
            sd=sd(lifeExp),
            n=n(),
            se=sd/sqrt(n))

ggplot(plot.data, aes(x=continent, y=mean.lifeExp)) +
  geom_bar(stat="identity")
```

Let's add error bars!

```
ggplot(plot.data, aes(x=continent, y=mean.lifeExp)) +
  geom_bar(stat="identity") +
  geom_errorbar(aes(ymax=mean.lifeExp + 2*se,
                    ymin=mean.lifeExp - 2*se), width = 0.3)
```

ggplot2

Adding layers to our scatterplot.
geom_smooth() adds a line of best fit:

```
ggplot(gapminder, aes(x=year, y=pop)) +
  geom_point(alpha=.3) +
  geom_smooth()
```

ggplot2

Adding layers to our scatterplot.
geom_smooth() adds a line of best fit:

```
ggplot(gapminder, aes(x=year, y=pop)) +
  geom_point(alpha=.3) +
  geom_smooth()
```

Key idea: ggplot works with layers, and you can add as many layers as you like.

ggplot2

Line plot

```
ggplot(gapminder, aes(x=year, y=pop, group=country)) +
  geom_line(alpha=.3)
```

ggplot2

The fun part! Exploring relationships bewteen several variables at once.
Use color= for lines and points and fill= for other shapes.

```
ggplot(gapminder, aes(x=year, y=gdpPercap, color=continent)) +
  geom_point(alpha=.6)

ggplot(gapminder, aes(x=year, y=gdpPercap,
                      color=continent, fill=continent)) +
  geom_point(alpha=.6) +
  geom_smooth()
```

```
ggplot(gapminder, aes(x=lifeExp, fill=continent)) +
  geom_histogram()
```

ggplot2: axes

You can control a lot about how your plots look with theme().

```
?theme
```

There are a few sensible theme bundlles available as well:

```
ggplot(gapminder, aes(x=year, y=gdpPercap,
                      color=continent, fill=continent)) +
  geom_point(alpha=.6) +
  geom_smooth() +
  theme_bw()
```

ggplot2: axes

You can control a lot about how your plots look with theme().

```
?theme
```

There are a few sensible theme bundlles available as well:

```
ggplot(gapminder, aes(x=year, y=gdpPercap,
                      color=continent, fill=continent)) +
  geom_point(alpha=.6) +
  geom_smooth() +
  theme_bw()
```

ggplot2: axes

```
ggplot(gapminder, aes(x=year, y=gdpPercap,
                      color=continent, fill=continent)) +
  geom_point(alpha=.6, show.legend = FALSE) +
  geom_smooth(show.legend = FALSE) +
  theme(axis.title.x=element_blank(),
        axis.text.x =element_text(angle=90)) +
  xlim(c(1970, 2007)) +
  labs(title="GDP per capita over time")
```

ggplot2: axes

```
ggplot(gapminder, aes(x=year, y=gdpPercap,
                      color=continent, fill=continent)) +
  geom_point(alpha=.6, show.legend = FALSE) +
  geom_smooth(show.legend = FALSE) +
  theme(axis.title.x=element_blank(),
        axis.text.x =element_text(angle=90)) +
  xlim(c(1970, 2007)) +
  labs(title="GDP per capita over time")
```

Key idea: There are LOTS of different ways to control the look of your plots. This feels overwhelming at first, but hopefully you'll appreciate the flexibility later.

ggplot2: axes

log transform an axis scale

```
ggplot(gapminder, aes(x=year, y=gdpPercap,
                      color=continent, fill=continent)) +
  geom_point(alpha=.6) +
  geom_smooth() +
  scale_y_log10()
```

ggplot2: axes

Changing the order a factor displays in.
Option 1: organize the levels of a factor based on some stat in the dataset

```
ggplot(plot.data, aes(y=continent, x=mean.lifeExp)) +
  geom_point() +
  geom_errorbarh(aes(xmax=mean.lifeExp + 2*se, xmin=mean.lifeExp - 2*se
```

```
ggplot(plot.data, aes(y=reorder(continent, mean.lifeExp), x=mean.lifeEx
  geom_point() +
  geom_errorbarh(aes(xmax=mean.lifeExp + 2*se, xmin=mean.lifeExp - 2*se
```

ggplot2: axes

Changing the order a factor displays in.
Option 1: organize the levels of a factor based on some stat in the dataset

```
ggplot(plot.data, aes(y=continent, x=mean.lifeExp)) +
  geom_point() +
  geom_errorbarh(aes(xmax=mean.lifeExp + 2*se, xmin=mean.lifeExp - 2*se
```

```
ggplot(plot.data, aes(y=reorder(continent, mean.lifeExp), x=mean.lifeEx
  geom_point() +
  geom_errorbarh(aes(xmax=mean.lifeExp + 2*se, xmin=mean.lifeExp - 2*se
```

ggplot2: axes

```
ggplot(plot.data, aes(x=reorder(continent, mean.lifeExp), y=mean.lifeEx
  geom_point() +
  geom_errorbar(aes(xmax=mean.lifeExp + 2*se, xmin=mean.lifeExp - 2*se)
  coord_flip()
```

ggplot2: axes

Changing the order a factor displays in.
Option 2: Change the order of the factor levels right in the data frame itself

```
levels(plot.data$continent)
plot.data$continent <- factor(plot.data$continent,
                              levels=c("Americas", "Europe", "Africa",
```

```
ggplot(plot.data, aes(y=continent, x=mean.lifeExp)) +
  geom_point() +
  geom_errorbarh(aes(xmax=mean.lifeExp + 2*se, xmin=mean.lifeExp - 2*se
```

ggplot2: axes

Changing the order a factor displays in.
Option 2: Change the order of the factor levels right in the data frame itself

```
levels(plot.data$continent)
plot.data$continent <- factor(plot.data$continent,
                              levels=c("Americas", "Europe", "Africa",
```

```
ggplot(plot.data, aes(y=continent, x=mean.lifeExp)) +
  geom_point() +
  geom_errorbarh(aes(xmax=mean.lifeExp + 2*se, xmin=mean.lifeExp - 2*se
```

Test your knowledge: ggplot2

Use this data frame (one of the ones that comes with ggplot2):

```
head(diamonds)
str(diamonds)
```

How would you do this?
Make a plot showing the relationship between log(carat) and log(price) of diamonds. Add a line that shows the linear best fit (i.e. a regression line)

```
ggplot(diamonds, aes(x=carat, y=price)) +
  geom_point(alpha=.2) +
  scale_x_log10() +
  scale_y_log10() +
  geom_smooth(method="lm") +
  ggtitle("Diamonds")
```

Test your knowledge: ggplot2

Use this data frame (one of the ones that comes with ggplot2):

```
head(diamonds)
str(diamonds)
```

How would you do this?
Make a plot showing the relationship between log(carat) and log(price) of diamonds. Add a line that shows the linear best fit (i.e. a regression line)

```
ggplot(diamonds, aes(x=carat, y=price)) +
  geom_point(alpha=.2) +
  scale_x_log10() +
  scale_y_log10() +
  geom_smooth(method="lm") +
  ggtitle("Diamonds")
```

Test your knowledge: ggplot2

What will this do? What will this plot look like? Try drawing it (on paper) if you can.

```
ggplot(diamonds, aes(x=price, fill=cut, color=cut)) +
  geom_density(alpha=.4) +
  labs(y=NULL)
```

Test your knowledge: ggplot2

What will this do? What will this plot look like? Try drawing it (on paper) if you can.

```
ggplot(diamonds, aes(x=price, fill=cut, color=cut)) +
  geom_density(alpha=.4) +
  labs(y=NULL) +
  facet_wrap(~ color)
```

Key ideas from this section

Key ideas from this section

Key idea: ggplot() and aes() set up what data will get plotted. Then you tell it how to plot with geoms.

Key ideas from this section

Key idea: ggplot() and aes() set up what data will get plotted. Then you tell it how to plot with geoms.

Key idea: You should have your data manipulation (e.g. getting summary stats) done before you try to get ggplot to plot it.

Key ideas from this section

Key idea: ggplot() and aes() set up what data will get plotted. Then you tell it how to plot with geoms.

Key idea: You should have your data manipulation (e.g. getting summary stats) done before you try to get ggplot to plot it.

Key idea: ggplot works with layers, and you can add as many layers as you like.