

NC STATE UNIVERSITY

Introduction to R for Data Science

Part III

Justin Post
August 12-13, 2019

What do we want to be able to do?

- Read in data
- **Manipulate data**
- Plot data
- Summarize data
- Analyze data

Schedule

Day 1

- Install R/R studio
- R Studio Interface
- Classes and Objects
- Attributes and Basic Data Object Manipulation
- Reading in Data/Writing Out Data
- **Logical Statements and Subsetting/Manipulating Data**

Logical Statements

- Logical statement - comparison of two quantities
- resolves as TRUE or FALSE

#Use of ==, !=, >=, <=, >, <

```
"hi" == " hi"
```

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

```
"hi" == "hi"
```

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

```
4 == 1
```

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

```
4 >= 3
```

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

```
4 != 1
```

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

```
"hi" != "hello"
```

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

Logical Statements

- Logical statement - comparison of two quantities
- resolves as TRUE or FALSE

```
#use of is. functions
```

```
is.numeric("Word")
```

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

```
is.numeric(10)
```

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

```
is.character("10")
```

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

```
is.na(c(1:2, NA, 3))
```

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

Logical Statements/Subsetting Data

- Useful for indexing a vector

```
iris <- tbl_df(iris)
iris
```

```
## # A tibble: 150 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##         <dbl>         <dbl>         <dbl>         <dbl> <fct>
## 1         5.1         3.5         1.4         0.2 setosa
## 2         4.9         3         1.4         0.2 setosa
## 3         4.7         3.2         1.3         0.2 setosa
## 4         4.6         3.1         1.5         0.2 setosa
## 5         5         3.6         1.4         0.2 setosa
## # ... with 145 more rows
```

Logical Statements/Subsetting Data

- Useful for indexing a vector
- Standard way for subsetting data (or use `subset` function)

```
iris[iris$Species == "setosa", ]
```

- Concept:
 - Feed index a vector of `TRUE/FALSE` or `0/1` values
 - R returns elements where `TRUE` or `1` occurred

Logical Statements/Subsetting Data

- Useful for indexing a vector

```
## Obtain a vector that indicates which rows are "setosa" species  
iris$Species == "setosa"
```

```
##      [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  
##      [12]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  
##      [23]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  
##      [34]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  
##      [45]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE  
##      [56] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
##      [67] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
##      [78] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
##      [89] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
##     [100] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
##     [111] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
##     [122] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
##     [133] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
##     [144] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```


Logical Statements/Subsetting Data

- Useful for indexing a vector
- Standard way for subsetting data (or use `subset` function)

```
iris[iris$Species == "setosa", ]
```

```
## # A tibble: 50 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##         <dbl>         <dbl>         <dbl>         <dbl> <fct>
## 1         5.1         3.5         1.4         0.2 setosa
## 2         4.9         3         1.4         0.2 setosa
## 3         4.7         3.2         1.3         0.2 setosa
## 4         4.6         3.1         1.5         0.2 setosa
## 5         5         3.6         1.4         0.2 setosa
## # ... with 45 more rows
```

Logical Statements/Subsetting Data

- Useful for indexing a vector
- Standard way for subsetting data (or use `subset` function)

```
subset(iris, Species == "setosa")
```

```
## # A tibble: 50 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##         <dbl>         <dbl>         <dbl>         <dbl> <fct>
## 1         5.1         3.5         1.4         0.2 setosa
## 2         4.9         3         1.4         0.2 setosa
## 3         4.7         3.2         1.3         0.2 setosa
## 4         4.6         3.1         1.5         0.2 setosa
## 5         5         3.6         1.4         0.2 setosa
## # ... with 45 more rows
```

Logical Statements/Subsetting Data

- Useful for indexing a vector
- Optional way: `filter()` from `dplyr` (installed with `tidyverse`)

```
filter(iris, Species == "setosa")
```

```
## # A tibble: 50 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##         <dbl>         <dbl>         <dbl>         <dbl> <fct>
## 1         5.1         3.5         1.4         0.2 setosa
## 2         4.9         3         1.4         0.2 setosa
## 3         4.7         3.2         1.3         0.2 setosa
## 4         4.6         3.1         1.5         0.2 setosa
## 5         5         3.6         1.4         0.2 setosa
## # ... with 45 more rows
```

Logical Statements/Subsetting Data

```
filter(iris, Species != "setosa")
```

```
## # A tibble: 100 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##   <dbl>         <dbl>         <dbl>         <dbl> <fct>
## 1           7         3.2         4.7         1.4 versicolor
## 2          6.4         3.2         4.5         1.5 versicolor
## 3          6.9         3.1         4.9         1.5 versicolor
## 4          5.5         2.3          4         1.3 versicolor
## 5          6.5         2.8         4.6         1.5 versicolor
## # ... with 95 more rows
```

- We'll spend a good bit of time with dplyr in a bit!

Implicit Data Change

Aside: Coercion

- R attempts to coerce data into usable form when necessary
- Ex: Atomic vector - all elements must be the same type

```
#coerce numeric to string  
c("hi", 10)
```

```
## [1] "hi" "10"
```

```
#coerce TRUE/FALSE to numeric  
c(TRUE, FALSE) + 0
```

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

Implicit Data Change

Aside: Coercion

- R attempts to coerce data into usable form when necessary
- Coerce from less flexible to more flexible
 - Data types from least to most flexible:
 - logical
 - integer
 - double
 - character.

```
#logical to character  
c(TRUE, "hi")
```

```
## [1] "TRUE" "hi"
```

Implicit Data Change

Aside: Coercion

- R attempts to coerce data into usable form when necessary
- Explicit coercion with `as.` functions

```
as.numeric(c(TRUE, FALSE, TRUE))
```

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

```
mean(c(TRUE, FALSE, TRUE))
```

```
## [1] 0.6666667
```

```
as.character(c(1, 2, 3.5, TRUE))
```

```
## [1] "1" "2" "3.5" "1"
```

- Why does TRUE return "1"?

Logical Statements

Logical Operators

- `&` 'and'
- `|` 'or'

Operator	A,B true	A true, B false	A,B false
<code>&</code>	<code>A & B = TRUE</code>	<code>A & B = FALSE</code>	<code>A & B = FALSE</code>
<code> </code>	<code>A B = TRUE</code>	<code>A B = TRUE</code>	<code>A B = FALSE</code>

Logical Statements

Logical Operators

- `&` 'and'
- `|` 'or'

Operator	A,B true	A true, B false	A,B false
<code>&</code>	<code>A & B = TRUE</code>	<code>A & B = FALSE</code>	<code>A & B = FALSE</code>
<code> </code>	<code>A B = TRUE</code>	<code>A B = TRUE</code>	<code>A B = FALSE</code>

- `&&` and `||` are alternatives
- Looks at only first comparison if given a vector of comparisons

Logical Statements

Logical Operators

```
set.seed(3)
x <- runif(n = 10, min = 0, max = 1); x

## [1] 0.1680415 0.8075164 0.3849424 0.3277343 0.6021007 0.6043941 0.1246334
## [8] 0.2946009 0.5776099 0.6309793

(x < 0.25) | (x > 0.75)

## [1] TRUE TRUE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE

(x < 0.25) || (x > 0.75)

## [1] TRUE
```

Subsetting Data

- Only pull out large petal setosa flowers

```
filter(iris, (Petal.Length > 1.5) & (Petal.Width > 0.3) &
       (Species == "setosa"))
```

```
## # A tibble: 5 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##   <dbl>         <dbl>         <dbl>         <dbl> <fct>
## 1         5.4         3.9         1.7         0.4 setosa
## 2         5.1         3.3         1.7         0.5 setosa
## 3         5         3.4         1.6         0.4 setosa
## 4         5         3.5         1.6         0.6 setosa
## 5         5.1         3.8         1.9         0.4 setosa
```

Subsetting Data

What's the idea for the filter function?

- Condition evaluates a vector of `TRUE/FALSE`
- Returns values where `TRUE` is present

Subsetting Data

What's the idea for the filter function?

- Condition evaluates a vector of TRUE/FALSE
- Returns values where TRUE is present

```
(iris$Petal.Length > 1.5) & (iris$Petal.Width > 0.3) &  
  (iris$Species == "setosa")
```

```
## [1] FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE  
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [23] FALSE TRUE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE  
## [34] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE  
## [45] TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [56] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [67] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [78] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [89] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [100] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [111] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [122] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [133] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [144] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

Subsetting Data

- Only pull out large petal setosa flowers

```
filter(iris, (Petal.Length > 1.5) & (Petal.Width > 0.3) &
        (Species == "setosa"))
```

```
## # A tibble: 5 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##         <dbl>         <dbl>         <dbl>         <dbl> <fct>
## 1         5.4         3.9         1.7         0.4 setosa
## 2         5.1         3.3         1.7         0.5 setosa
## 3         5         3.4         1.6         0.4 setosa
## 4         5         3.5         1.6         0.6 setosa
## 5         5.1         3.8         1.9         0.4 setosa
```

- In `[,]` notation this is more work!

```
iris[(iris$Petal.Length > 1.5) & (iris$Petal.Width > 0.3) &
      (iris$Species == "setosa"), ]
```

Conditional Execution

If then, If then else

- Often want to execute statements conditionally (say to create a new variable)
- `if then else` concept

```
if (condition) {  
  then execute code  
}
```

```
#if then else  
if (condition) {  
  execute this code  
} else {  
  execute this code  
}
```

Conditional Execution

If then, If then else

- Often want to execute statements conditionally (say to create a new variable)
- `if then else` concept

```
#Or more if statements
if (condition) {
  execute this code
} else if (condition2) {
  execute this code
} else if (condition3) {
  execute this code
} else {
  #if no conditions met
  execute this code
}
```


Conditional Execution

If then, If then else

```
#silly example
a <- 5
if (a < 10){
  print("hi")
}

## [1] "hi"
```

```
if (a < 10){
  print("hi")
} else if (a < 40){
  print("goodbye")
} else {
  print("aloha")
}

## [1] "hi"
```

Conditional Execution

If then, If then else

```
#silly example
a <- 20
if (a < 10){
  print("hi")
}

if (a < 10){
  print("hi")
} else if (a < 40){
  print("goodbye")
} else {
  print("aloha")
}

## [1] "goodbye"
```

Conditional Execution

If then, If then else

```
#silly example
a <- "string"
if (a < 10){
  print("hi")
}

if (a < 10){
  print("hi")
} else if (a < 40){
  print("goodbye")
} else {
  print("aloha")
}

## [1] "aloha"
```

Manipulating Data

- Logical statments great for data filtering
- Quite useful for creating new variables too
- Issue: `if` condition can only take in a single comparison
- Create new variable for **Large Setosa** flowers

Manipulating Data

- Logical statments great for data filtering
- Quite useful for creating new variables too
- Issue: `if` condition can only take in a single comparison
- Create new variable for **Large Setosa** flowers

```
if ((iris$Petal.Length > 1.5) & (iris$Petal.Width > 0.3) &
    (iris$Species == "setosa")) {
  "Large Setosa"
}
```

```
## Warning in if ((iris$Petal.Length > 1.5) & (iris$Petal.Width > 0.3) & (iris
## $Species == : the condition has length > 1 and only the first element will
## be used
```

Manipulating Data

- Logical statements great for data filtering
- Quite useful for creating new variables too
- `ifelse()` is **vectorized** if statement (see help)
- Returns a vector

#syntax

```
ifelse(vector_condition, if_true_do_this, if_false_do_this)
```

Manipulating Data

- Create new variable for Large **Setosa** flowers

```
ifelse((iris$Petal.Length > 1.5) & (iris$Petal.Width > 0.3) &
      (iris$Species == "setosa"), "L-S", "NotL-S")
```

```
## [1] "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "L-S" "NotL-S"
## [8] "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S"
## [15] "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S"
## [22] "NotL-S" "NotL-S" "L-S" "NotL-S" "NotL-S" "L-S" "NotL-S"
## [29] "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S"
## [36] "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S"
## [43] "NotL-S" "L-S" "L-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S"
## [50] "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S"
## [57] "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S"
## [64] "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S"
## [71] "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S"
## [78] "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S"
## [85] "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S"
## [92] "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S"
## [99] "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S"
## [106] "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S"
## [113] "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S"
## [120] "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S"
## [127] "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S"
## [134] "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S"
## [141] "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S" "NotL-S"
## [148] "NotL-S" "NotL-S" "NotL-S"
```

Manipulating Data

- `dplyr` has a nice function called `transmute()`

```
transmute(iris, Size = ifelse(
  (Petal.Length > 1.5) & (Petal.Width > 0.3) & (Species == "setosa"), "LS", "NotLS")
)
```

```
## # A tibble: 150 x 1
##   Size
##   <chr>
## 1 NotLS
## 2 NotLS
## 3 NotLS
## 4 NotLS
## 5 NotLS
## # ... with 145 more rows
```


Manipulating Data

- `dplyr` function `mutate()` does the same but *adds variable to existing data frame*

```
mutate(iris, Size = ifelse(
  (Petal.Length > 1.5) & (Petal.Width > 0.3) & (Species == "setosa"), "LS", "NotLS")
)
```

```
## # A tibble: 150 x 6
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species Size
##   <dbl>         <dbl>         <dbl>         <dbl> <fct>   <chr>
## 1         5.1         3.5         1.4         0.2 setosa NotLS
## 2         4.9         3         1.4         0.2 setosa NotLS
## 3         4.7         3.2         1.3         0.2 setosa NotLS
## 4         4.6         3.1         1.5         0.2 setosa NotLS
## 5          5         3.6         1.4         0.2 setosa NotLS
## # ... with 145 more rows
```

Recap!

Logical Operators

- `&` 'and'
- `|` 'or'
- `if (condition) { ... }`
- `if (condition) { ... } else if (condition) { ... }`
- `ifelse(vector_condition, result_if_true, result_if_false)`

Subsetting/Manipulating Data

Overview of dplyr package

- `dplyr` package made for most standard data manipulation tasks
- Part of `tidyverse`
- Make sure `library(tidyverse)` has been run

dplyr package

- Basic commands
 - `tbl_df()` - convert data frame to one with better printing
 - `filter()` - subset **rows**
 - `arrange()` - reorder **rows**
 - `select()` - subset **columns**
 - `mutate()` - add newly created **column**
 - `transmute()` - create new variable
 - `group_by()` - group **rows** by a variable
 - `summarise()` - apply basic function to data
 - `left_join()`, `right_join()`, `inner_join()`, `full_join()` - commands to combine multiple data frames

Tidyverse Syntax

- Reason to prefer `dplyr` and packages from the `tidyverse`
- Fast!
- Good defaults
- All packages have similar syntax! All work on `tibbles` (data frames)
- Syntax: `function(data.frame, actions, ...)`

Subsetting/Manipulating Data

`tbl_df()` - convert data frame to one with better printing

- If data read in with `haven`, `readxl`, or `readr` probably in this format!
- Just 'wrap' data frame

```
#install.packages("Lahman")  
library(Lahman)  
head(Batting, n = 4) #look at just first 4 observations
```

Subsetting/Manipulating Data

`tbl_df()` - convert data frame to one with better printing

```
head(Batting, n = 4) #look at just first 4 observations
```

```
##      playerID yearID stint teamID lgID  G  AB  R  H X2B X3B HR  RBI  SB  CS  BB
## 1 abercda01   1871     1    TRO   NA   1   4   0   0   0   0   0   0   0   0   0
## 2 addybo01    1871     1    RC1   NA  25 118 30 32   6   0   0  13   8   1   4
## 3 allisar01    1871     1    CL1   NA  29 137 28 40   4   5   0  19   3   1   2
## 4 allisdo01    1871     1    WS3   NA  27 133 28 44  10   2   2  27   1   1   0
##      SO  IBB  HBP  SH  SF  GIDP
## 1   0   NA   NA  NA  NA   NA
## 2   0   NA   NA  NA  NA   NA
## 3   5   NA   NA  NA  NA   NA
## 4   2   NA   NA  NA  NA   NA
```

Subsetting/Manipulating Data

```
Batting <- tbl_df(Batting)
Batting
```

```
## # A tibble: 102,816 x 22
##   playerID yearID stint teamID lgID      G    AB    R    H   X2B   X3B
##   <chr>      <int> <int> <fct>  <fct> <int> <int> <int> <int> <int> <int>
## 1 abercda~   1871     1 TRO    NA      1     4     0     0     0     0
## 2 addybo01   1871     1 RC1    NA     25    118    30    32     6     0
## 3 allisar~   1871     1 CL1    NA     29    137    28    40     4     5
## 4 allisdo~   1871     1 WS3    NA     27    133    28    44    10     2
## 5 ansonca~   1871     1 RC1    NA     25    120    29    39    11     3
## # ... with 1.028e+05 more rows, and 11 more variables: HR <int>,
## #   RBI <int>, SB <int>, CS <int>, BB <int>, SO <int>, IBB <int>,
## #   HBP <int>, SH <int>, SF <int>, GIDP <int>
```


Subsetting/Manipulating Data

`filter()` - subset rows

- Use `filter()` to obtain only PIT data

```
filter(Batting, teamID == "PIT")
```

```
## # A tibble: 4,722 x 22
##   playerID yearID stint teamID lgID      G    AB    R    H   X2B   X3B
##   <chr>      <int> <int> <fct>  <fct> <int> <int> <int> <int> <int> <int>
## 1 barklsa~  1887     1 PIT    NL      89   340   44    76    10     4
## 2 beeched~  1887     1 PIT    NL      41   169   15    41     8     0
## 3 bishobi~  1887     1 PIT    NL       3     9    0     0     0     0
## 4 brownto~  1887     1 PIT    NL      47   192   30    47     3     4
## 5 carrofr~  1887     1 PIT    NL     102   421   71   138    24    15
## # ... with 4,717 more rows, and 11 more variables: HR <int>, RBI <int>,
## #   SB <int>, CS <int>, BB <int>, SO <int>, IBB <int>, HBP <int>,
## #   SH <int>, SF <int>, GIDP <int>
```

Subsetting/Manipulating Data

`filter()` - subset rows

- Multiple filters

```
filter(Batting, teamID == "PIT" & yearID == 2000)
```

```
## # A tibble: 46 x 22
##   playerID yearID stint teamID lgID      G    AB    R    H   X2B   X3B
##   <chr>      <int> <int> <fct> <fct> <int> <int> <int> <int> <int> <int>
## 1 anderji~   2000     1 PIT    NL     27    50     5     7     1     0
## 2 arroybr~   2000     1 PIT    NL     21    21     2     3     2     0
## 3 avenbr01   2000     1 PIT    NL     72   148    18    37    11     0
## 4 benjami~   2000     1 PIT    NL     93   233    28    63    18     2
## 5 bensokr~   2000     1 PIT    NL     32    65     3     6     2     0
## # ... with 41 more rows, and 11 more variables: HR <int>, RBI <int>,
## #   SB <int>, CS <int>, BB <int>, SO <int>, IBB <int>, HBP <int>,
## #   SH <int>, SF <int>, GIDP <int>
```

Subsetting/Manipulating Data

`arrange()` - reorder rows

#reorder by teamID

`arrange(Batting, teamID)`

A tibble: 102,816 x 22

	playerID	yearID	stint	teamID	lgID	G	AB	R	H	X2B	X3B
	<chr>	<int>	<int>	<fct>	<fct>	<int>	<int>	<int>	<int>	<int>	<int>
## 1	berrych~	1884	1	ALT	UA	7	25	2	6	0	0
## 2	brownji~	1884	1	ALT	UA	21	88	12	22	2	2
## 3	carropa~	1884	1	ALT	UA	11	49	4	13	1	0
## 4	connojo~	1884	1	ALT	UA	3	11	0	1	0	0
## 5	crosscl~	1884	1	ALT	UA	2	7	1	4	1	0

... with 1.028e+05 more rows, and 11 more variables: HR <int>,
 ## # RBI <int>, SB <int>, CS <int>, BB <int>, SO <int>, IBB <int>,
 ## # HBP <int>, SH <int>, SF <int>, GIDP <int>

Subsetting/Manipulating Data

`arrange()` - reorder rows

```
#get secondary arrangement as well
arrange(Batting, teamID, G)
```

```
## # A tibble: 102,816 x 22
##   playerID yearID stint teamID lgID      G    AB    R    H   X2B   X3B
##   <chr>      <int> <int> <fct> <fct> <int> <int> <int> <int> <int> <int>
## 1 daisege~  1884     1 ALT   UA      1     4     0     0     0     0
## 2 crosscl~  1884     1 ALT   UA      2     7     1     4     1     0
## 3 manloch~  1884     1 ALT   UA      2     7     1     3     0     0
## 4 connojo~  1884     1 ALT   UA      3    11     0     1     0     0
## 5 berrych~  1884     1 ALT   UA      7    25     2     6     0     0
## # ... with 1.028e+05 more rows, and 11 more variables: HR <int>,
## #   RBI <int>, SB <int>, CS <int>, BB <int>, SO <int>, IBB <int>,
## #   HBP <int>, SH <int>, SF <int>, GIDP <int>
```

Subsetting/Manipulating Data

`arrange()` - reorder rows

#descending instead

```
arrange(Batting, teamID, desc(G))
```

```
## # A tibble: 102,816 x 22
```

```
##   playerID yearID stint teamID lgID      G    AB    R    H   X2B   X3B
##   <chr>      <int> <int> <fct> <fct> <int> <int> <int> <int> <int> <int>
## 1 smithge~  1884     1 ALT   UA     25  108    9   34     8     1
## 2 harrifr~  1884     1 ALT   UA     24   95   10   25     2     1
## 3 doughch~  1884     1 ALT   UA     23   85    6   22     5     0
## 4 murphjo~  1884     1 ALT   UA     23   94   10   14     1     0
## 5 brownji~  1884     1 ALT   UA     21   88   12   22     2     2
## # ... with 1.028e+05 more rows, and 11 more variables: HR <int>,
## #   RBI <int>, SB <int>, CS <int>, BB <int>, SO <int>, IBB <int>,
## #   HBP <int>, SH <int>, SF <int>, GIDP <int>
```

Subsetting/Manipulating Data

Piping or Chaining

- Applying multiple functions: nesting hard to parse!
- Piping or Chaining with `%>%` operator helps

```
arrange(filter(Batting, teamID == "PIT"), desc(G))
```

```
## # A tibble: 4,722 x 22
##   playerID yearID stint teamID lgID      G    AB    R    H   X2B   X3B
##   <chr>      <int> <int> <fct> <fct> <int> <int> <int> <int> <int> <int>
## 1 mazerbi~   1967     1 PIT    NL     163   639   62   167    25     3
## 2 bonilbo~   1989     1 PIT    NL     163   616   96   173    37    10
## 3 mazerbi~   1964     1 PIT    NL     162   601   66   161    22     8
## 4 clenddo~   1965     1 PIT    NL     162   612   89   184    32    14
## 5 mazerbi~   1966     1 PIT    NL     162   621   56   163    22     7
## # ... with 4,717 more rows, and 11 more variables: HR <int>, RBI <int>,
## #   SB <int>, CS <int>, BB <int>, SO <int>, IBB <int>, HBP <int>,
## #   SH <int>, SF <int>, GIDP <int>
```

Subsetting/Manipulating Data

Piping or Chaining

- Applying multiple functions: nesting hard to parse!
- Piping or Chaining with `%>%` operator helps

```
Batting %>% filter(teamID == "PIT") %>% arrange(desc(G))
```

```
## # A tibble: 4,722 x 22
##   playerID yearID stint teamID lgID      G    AB    R    H   X2B   X3B
##   <chr>      <int> <int> <fct> <fct> <int> <int> <int> <int> <int> <int>
## 1 mazerbi~   1967     1 PIT    NL     163  639   62  167   25    3
## 2 bonilbo~   1989     1 PIT    NL     163  616   96  173   37   10
## 3 mazerbi~   1964     1 PIT    NL     162  601   66  161   22    8
## 4 clenddo~   1965     1 PIT    NL     162  612   89  184   32   14
## 5 mazerbi~   1966     1 PIT    NL     162  621   56  163   22    7
## # ... with 4,717 more rows, and 11 more variables: HR <int>, RBI <int>,
## #   SB <int>, CS <int>, BB <int>, SO <int>, IBB <int>, HBP <int>,
## #   SH <int>, SF <int>, GIDP <int>
```

Subsetting/Manipulating Data

Piping or Chaining

- Applying multiple functions: nesting hard to parse!
- Piping or Chaining with `%>%` operator helps
- If `dplyr` or `magrittr` package loaded, can use with other functions

```
a<-runif(n = 10)
```

```
a
```

```
## [1] 0.5120159 0.5050239 0.5340354 0.5572494 0.8679195 0.8297087 0.1114492
```

```
## [8] 0.7036884 0.8974883 0.2797326
```


Subsetting/Manipulating Data

Piping or Chaining

```
#silly example
a %>% quantile()

##           0%          25%          50%          75%          100%
## 0.1114492 0.5067719 0.5456424 0.7982036 0.8974883

a %>% quantile() %>% range()

## [1] 0.1114492 0.8974883
```

Subsetting/Manipulating Data

`select()` - subset **columns**

- Often only want select variables (saw `$` and `[,]`)
- `select()` function has same syntax as other `dplyr` functions!

#Choose a single column by name

```
Batting %>% select(X2B)
```

```
## # A tibble: 102,816 x 1
```

```
##       X2B
```

```
##   <int>
```

```
## 1      0
```

```
## 2      6
```

```
## 3      4
```

```
## 4     10
```

```
## 5     11
```

```
## # ... with 1.028e+05 more rows
```

Subsetting/Manipulating Data

`select()` - subset **columns**

- Many ways to select variables

#all columns between

```
Batting %>% select(X2B:HR)
```

```
## # A tibble: 102,816 x 3
```

```
##       X2B     X3B     HR
```

```
##   <int> <int> <int>
```

```
## 1      0      0      0
```

```
## 2      6      0      0
```

```
## 3      4      5      0
```

```
## 4     10      2      2
```

```
## 5     11      3      0
```

```
## # ... with 1.028e+05 more rows
```

Subsetting/Manipulating Data

`select()` - subset columns

- Many ways to select variables

#all columns containing

```
Batting %>% select(contains("X"))
```

```
## # A tibble: 102,816 x 2
```

```
##       X2B    X3B
```

```
##   <int> <int>
```

```
## 1      0      0
```

```
## 2      6      0
```

```
## 3      4      5
```

```
## 4     10      2
```

```
## 5     11      3
```

```
## # ... with 1.028e+05 more rows
```

Subsetting/Manipulating Data

`select()` - subset columns

- Many ways to select variables

#all columns starting with

```
Batting %>% select(starts_with("X"))
```

```
## # A tibble: 102,816 x 2
```

```
##       X2B    X3B
```

```
##   <int> <int>
```

```
## 1      0      0
```

```
## 2      6      0
```

```
## 3      4      5
```

```
## 4     10      2
```

```
## 5     11      3
```

```
## # ... with 1.028e+05 more rows
```

Subsetting/Manipulating Data

`select()` - subset columns

- Many ways to select variables

#all columns ending with

```
Batting %>% select(ends_with("ID"))
```

```
## # A tibble: 102,816 x 4
```

```
##   playerID  yearID teamID lgID
```

```
##   <chr>      <int> <fct>  <fct>
```

```
## 1 abercda01   1871 TRO     NA
```

```
## 2 addybo01   1871 RC1     NA
```

```
## 3 allisar01  1871 CL1     NA
```

```
## 4 allisdo01  1871 WS3     NA
```

```
## 5 ansonca01  1871 RC1     NA
```

```
## # ... with 1.028e+05 more rows
```

Subsetting/Manipulating Data

`mutate()` - add newly created column

`transmute()` - create new variable

##Create an Extra Base Hits variable

```
Batting %>% mutate(ExtraBaseHits = X2B + X3B + HR)
```

```
## # A tibble: 102,816 x 23
```

```
##   playerID yearID stint teamID lgID      G    AB    R    H   X2B   X3B
##   <chr>      <int> <int> <fct>  <fct> <int> <int> <int> <int> <int> <int>
## 1 abercda~   1871     1 TRO    NA      1     4     0     0     0     0
## 2 addybo01   1871     1 RC1    NA     25    118    30    32     6     0
## 3 allisar~   1871     1 CL1    NA     29    137    28    40     4     5
## 4 allisdo~   1871     1 WS3    NA     27    133    28    44    10     2
## 5 ansonca~   1871     1 RC1    NA     25    120    29    39    11     3
## # ... with 1.028e+05 more rows, and 12 more variables: HR <int>,
## #   RBI <int>, SB <int>, CS <int>, BB <int>, SO <int>, IBB <int>,
## #   HBP <int>, SH <int>, SF <int>, GIDP <int>, ExtraBaseHits <int>
```

Subsetting/Manipulating Data

`mutate()` - add newly created column

`transmute()` - create new variable

#can't see it!

```
Batting %>% mutate(ExtraBaseHits = X2B + X3B + HR) %>% select(ExtraBaseHits)
```

```
## # A tibble: 102,816 x 1
```

```
##   ExtraBaseHits
```

```
##           <int>
```

```
## 1             0
```

```
## 2             6
```

```
## 3             9
```

```
## 4            14
```

```
## 5            14
```

```
## # ... with 1.028e+05 more rows
```


Subsetting/Manipulating Data

`mutate()` - add newly created column

`transmute()` - create new variable

#transmute will keep the new variable only

```
Batting %>% transmute(ExtraBaseHits = X2B + X3B + HR)
```

```
## # A tibble: 102,816 x 1
```

```
##   ExtraBaseHits
```

```
##           <int>
```

```
## 1             0
```

```
## 2             6
```

```
## 3             9
```

```
## 4            14
```

```
## 5            14
```

```
## # ... with 1.028e+05 more rows
```

Subsetting/Manipulating Data

- Basic data summarizations often done by groups
- Average score by Age group
- Median income by Education level
- Number of participants for each Race

Subsetting/Manipulating Data

`group_by()` - group **rows** by a variable

`summarise()` - apply basic function to data

- Summarization - find avg # of doubles (X2B)
- Remove NA's
- NA = Not Available (R's missing data indicator)

#average # of doubles for all players in data set

```
Batting %>% summarise(AvgX2B = mean(X2B, na.rm = TRUE))
```

```
## # A tibble: 1 x 1
```

```
##   AvgX2B
```

```
##   <dbl>
```

```
## 1    6.29
```

Subsetting/Manipulating Data

`group_by()` - group **rows** by a variable

`summarise()` - apply basic function to data

- Summarization - find avg # of doubles (X2B) *by team* (explore this idea more later)

```
Batting %>% group_by(teamID) %>% summarise(AvgX2B = mean(X2B, na.rm = TRUE))
```

```
## # A tibble: 149 x 2
##   teamID AvgX2B
##   <fct>   <dbl>
## 1 ALT      1.76
## 2 ANA      6.84
## 3 ARI      6.30
## 4 ATL      5.86
## 5 BAL      6.00
## # ... with 144 more rows
```

Subsetting/Manipulating Data

- May want to combine two data sets: `left_join()`, `right_join()`, `inner_join()`, `full_join()`

(Cite: <http://rpubs.com/justmarkham/dplyr-tutorial-part-2>)

```
# create two simple data frames
```

```
a <- data_frame(color = c("green", "yellow", "red"), num = 1:3)
```

```
b <- data_frame(color = c("green", "yellow", "pink"), size = c("S", "M", "L"))
```

a

```
## # A tibble: 3 x 2
##   color    num
##   <chr> <int>
## 1 green     1
## 2 yellow    2
## 3 red       3
```

b

```
## # A tibble: 3 x 2
##   color  size
##   <chr> <chr>
## 1 green  S
## 2 yellow M
## 3 pink   L
```

Subsetting/Manipulating Data

`left_join()`, `right_join()`, `inner_join()`, `full_join()` - combine multiple DFs

- Only include observations found in both "a" and "b" (automatically joins on variables that appear in both tables)

```
a                                b                                inner_join(a, b)

## # A tibble: 3 x 2             ## # A tibble: 3 x 2             ## Joining, by = "color"
##   color    num                ##   color size                ## # A tibble: 2 x 3
##   <chr>  <int>                ##   <chr> <chr>                ##   color    num size
## 1 green     1                ## 1 green  S                    ##   <chr>  <int> <chr>
## 2 yellow    2                ## 2 yellow M                    ## 1 green     1 S
## 3 red       3                ## 3 pink   L                    ## 2 yellow    2 M
```

Subsetting/Manipulating Data

`left_join()`, `right_join()`, `inner_join()`, `full_join()` - combine multiple DFs

- include observations found in either "a" or "b"

```
a                                b                                full_join(a, b)

## # A tibble: 3 x 2             ## # A tibble: 3 x 2             ## Joining, by = "color"
##   color    num                ##   color  size                ## # A tibble: 4 x 3
##   <chr>  <int>                ##   <chr> <chr>                ##   color    num size
## 1 green     1                ## 1 green  S                    ##   <chr>  <int> <chr>
## 2 yellow    2                ## 2 yellow M                    ## 1 green     1  S
## 3 red       3                ## 3 pink   L                    ## 2 yellow    2  M
##                                     ##                                     ## 3 red       3 <NA>
##                                     ##                                     ## 4 pink      NA  L
```

Subsetting/Manipulating Data

`left_join()`, `right_join()`, `inner_join()`, `full_join()` - combine multiple DFs

- include all observations found in "a", match with b

```
a                                b                                left_join(a, b)

## # A tibble: 3 x 2              ## # A tibble: 3 x 2              ## Joining, by = "color"
##   color    num                ##   color  size                ## # A tibble: 3 x 3
##   <chr>  <int>                ##   <chr> <chr>                ##   color    num size
## 1 green     1                ## 1 green  S                    ##   <chr>  <int> <chr>
## 2 yellow     2                ## 2 yellow M                    ## 1 green     1  S
## 3 red       3                ## 3 pink   L                    ## 2 yellow     2  M
##                                     ## 3 red     3 <NA>
```


Subsetting/Manipulating Data

`left_join()`, `right_join()`, `inner_join()`, `full_join()` - combine multiple DFs

- include all observations found in "b", match with a

```
a                                b                                right_join(a, b)

## # A tibble: 3 x 2             ## # A tibble: 3 x 2             ## Joining, by = "color"
##   color    num                ##   color  size                ## # A tibble: 3 x 3
##   <chr>  <int>                ##   <chr> <chr>                ##   color    num size
## 1 green     1                ## 1 green  S                    ##   <chr>  <int> <chr>
## 2 yellow    2                ## 2 yellow M                    ## 1 green     1  S
## 3 red       3                ## 3 pink   L                    ## 2 yellow    2  M
##                                     ## 3 pink   NA  L
```

Subsetting/Manipulating Data

`left_join()`, `right_join()`, `inner_join()`, `full_join()` - combine multiple DFs

- `right_join(a, b)` is identical to `left_join(b, a)` except for column ordering

```
right_join(a,b)
```

```
## Joining, by = "color"
```

```
## # A tibble: 3 x 3
##   color    num size
##   <chr>  <int> <chr>
## 1 green      1 S
## 2 yellow     2 M
## 3 pink      NA L
```

```
left_join(b, a)
```

```
## Joining, by = "color"
```

```
## # A tibble: 3 x 3
##   color  size    num
##   <chr> <chr> <int>
## 1 green  S         1
## 2 yellow M         2
## 3 pink   L        NA
```

Subsetting/Manipulating Data

`left_join()`, `right_join()`, `inner_join()`, `full_join()` - combine multiple DFs

- filter "a" to only show observations that match "b"

```
a                                b                                semi_join(a, b)

## # A tibble: 3 x 2             ## # A tibble: 3 x 2             ## Joining, by = "color"
##   color    num                ##   color  size                ## # A tibble: 2 x 2
##   <chr>  <int>                ##   <chr> <chr>                ##   color    num
## 1 green     1                ## 1 green  S                    ##   <chr>  <int>
## 2 yellow    2                ## 2 yellow M                    ## 1 green     1
## 3 red       3                ## 3 pink   L                    ## 2 yellow     2
```

Subsetting/Manipulating Data

`left_join()`, `right_join()`, `inner_join()`, `full_join()` - combine multiple DFs

- filter "a" to only show observations that don't match "b"

```
a                                b                                anti_join(a, b)

## # A tibble: 3 x 2             ## # A tibble: 3 x 2             ## Joining, by = "color"
##   color    num                ##   color  size                ## # A tibble: 1 x 2
##   <chr>  <int>                ##   <chr> <chr>                ##   color    num
## 1 green     1                ## 1 green  S                        ##   <chr> <int>
## 2 yellow    2                ## 2 yellow M                        ## 1 red      3
## 3 red       3                ## 3 pink   L
```

Subsetting/Manipulating Data

`left_join()`, `right_join()`, `inner_join()`, `full_join()` - combine multiple DFs

- sometimes matching variables don't have identical names

```
b <- b %>% rename(col = color)
a
```

```
## # A tibble: 3 x 2
##   color    num
##   <chr>  <int>
## 1 green     1
## 2 yellow    2
## 3 red       3
```

```
b
```

```
## # A tibble: 3 x 2
##   col      size
##   <chr>  <chr>
## 1 green   S
## 2 yellow M
## 3 pink    L
```

Subsetting/Manipulating Data

`left_join()`, `right_join()`, `inner_join()`, `full_join()` - combine multiple DFs

- specify that the join should occur by matching "color" in "a" with "col" in "b"

```
a                                     b                                     inner_join(a, b,
                                     by = c("color" = "col"))

## # A tibble: 3 x 2                ## # A tibble: 3 x 2                ## # A tibble: 2 x 3
##   color    num                ##   col    size                ##   color    num size
##   <chr>  <int>                ##   <chr> <chr>                ##   <chr>  <int> <chr>
## 1 green     1                ## 1 green  S                    ## 1 green     1 S
## 2 yellow     2                ## 2 yellow M                    ## 2 yellow     2 M
## 3 red        3                ## 3 pink   L
```

Overview of dplyr package [cheatsheet](#)

- Basic commands
 - `tbl_df()` - convert data frame to one with better printing
 - `filter()` - subset **rows**
 - `arrange()` - reorder **rows**
 - `select()` - subset **columns**
 - `mutate()` - add newly created **column**
 - `transmute()` - create new variable
 - `group_by()` - group **rows** by a variable
 - `summarise()` - apply basic function to data
 - `left_join()`, `right_join()`, `inner_join()`, `full_join()` - commands to combine multiple data frames

Activity

- [Manipulating Data Activity instructions](#) available on web
- Work in small groups
- Ask questions! TAs and I will float about the room
- Feel free to ask questions about anything you didn't understand as well!