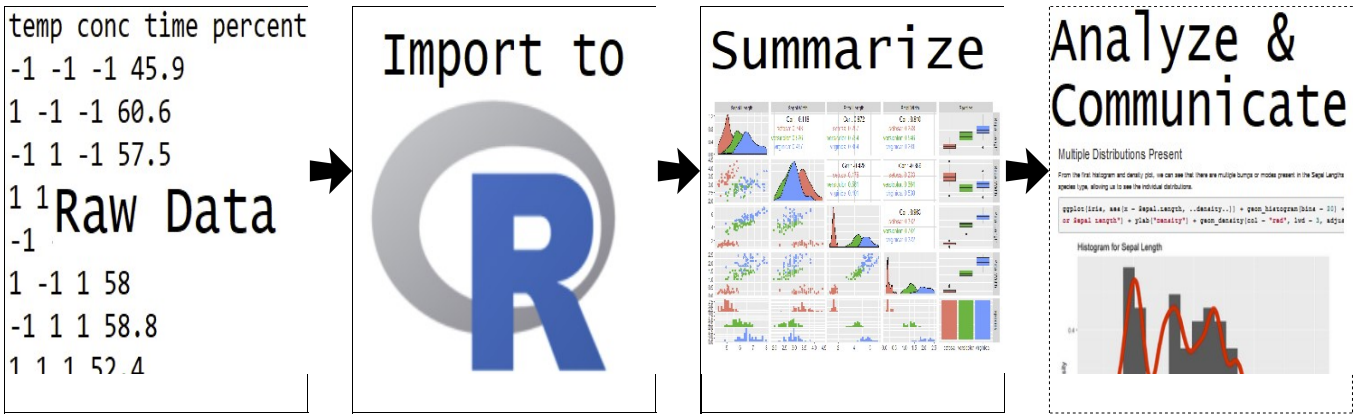
NC STATE UNIVERSITY

Introduction to R for Data Science Part III

Justin Post

What is this course about?

Basic use of R for reading, manipulating, and plotting data!



Schedule

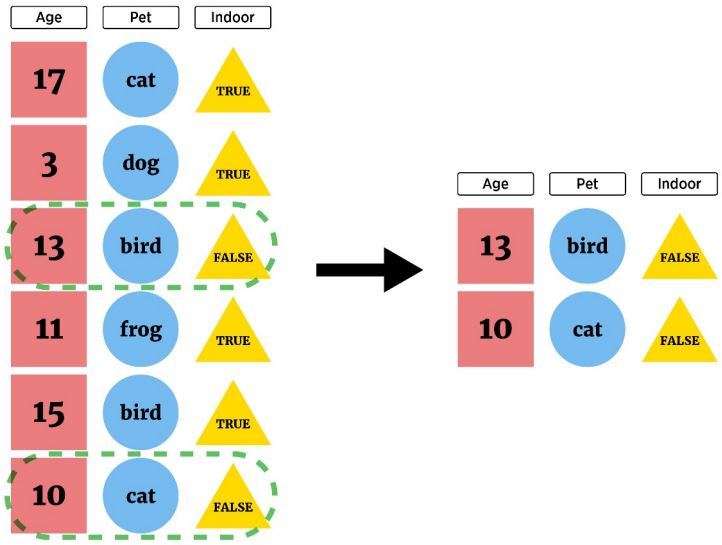
Day 2

- **Logical Statements and Subsetting/Manipulating Data**
- Numerical and Graphical Summaries
- Basic Analyses

Data manipulation idea

We may want to subset our full data set or create new data

- Grab only certain types of observations (**filter rows**)



Logical Statements

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

```r
"hi" == " hi" #== is comparison
```

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

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

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

```r
4 >= 1
```

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

```r
4 != 1
```

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

```r
sqrt(3)^2  == 3
```

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

```r
dplyr::near(sqrt(3)^2, 3)
```

```
## [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
```

```
is.matrix(c("hello", "world"))
```

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

Logical Statements/Subsetting Data

- Useful for indexing a vector

```
iris <- as_tibble(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

Goal: Subset rows or columns

- Consider the built-in `iris` dataframe

```
head(iris)
```

```
## # A tibble: 6 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
## 6          5.4         3.9          1.7         0.4 setosa
```

Logical statements

Goal: Subset rows or columns

- **logical statement** - useful for indexing an R object
- Concept:
  - Feed index a vector of `TRUE`/`FALSE`
  - R returns elements where `TRUE`

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

Logical statements

- Concept:
  - Feed index a vector of `TRUE`/`FALSE`
  - R returns elements where `TRUE`

```
iris$Species == "setosa" #vector indicating setosa values
```

```
##   [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [25]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [37]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [49]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [73] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [85] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [97] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [109] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [121] 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
## [145] FALSE FALSE FALSE FALSE FALSE FALSE
```

Logical statements

Goal: Subset rows or columns

- **logical statement** - useful for indexing an R object

```
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

Goal: Subset rows or columns

- **logical statement** - useful for indexing an R object
- Similarly, can 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
```

dplyr

Goal: Subset rows or columns

- **logical statement** - useful for indexing an R object
- Similarly, can use `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
```

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
&	`A & B = TRUE`	`A & B = FALSE`	`A & B = FALSE`
|	`A | B = TRUE`	`A | B = TRUE`	`A | B = FALSE`

Logical Statements

Logical Operators

- `&` ‘and’
- `|` ‘or’

Operator	A,B true	A true, B false	A,B false
&	`A & B = TRUE`	`A & B = FALSE`	`A & B = FALSE`
|	`A | B = TRUE`	`A | B = TRUE`	`A | B = FALSE`

- `&&` 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 FALSE
##  [13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
##  [25] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE
##  [49] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [73] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [85] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [97] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [109] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [121] 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
## [145] 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"), ]
```

`tidyverse` for data manipulations

Overview of dplyr and tidyr packages

- `dplry` package made for most standard data manipulation tasks
- `tidyr` package reshapes data
- Both part of `tidyverse`
- Make sure `library(tidyverse)` has been run!

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(tibble, actions, ...)`

dplyr

- Basic commands
  - `as_tibble()` - convert data frame to one with better printing
  - `filter()` - subset **rows**
  - `arrange()` - reorder **rows**
  - `select()` - subset **columns**
  - `rename()` - rename **columns**
  - `mutate()` - add newly created **column**
  - `transmute()` - create new variable
  - `group_by()` - group **rows** by a variable
  - `summarise()` - apply basic function to data

dplyr

`as_tibble()` - convert data frame to one with better printing and no simplification

- Just 'wrap' data frame

```
#install.packages("Lahman")
library(Lahman)
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 SO
## 1 abercda01   1871     1    TRO   NA  1   4  0  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  0
## 3 allisar01   1871     1    CL1   NA 29 137 28 40   4   5  0  19  3  1  2  5
## 4 allisdo01   1871     1    WS3   NA 27 133 28 44  10   2  2  27  1  1  0  2
##   IBB HBP SH SF GIDP
## 1  NA  NA NA NA    0
## 2  NA  NA NA NA    0
## 3  NA  NA NA NA    1
## 4  NA  NA NA NA    0
```

dplyr

```
Batting <- as_tibble(Batting)
Batting
```

```
## # A tibble: 105,861 x 22
##   playerID yearID stint teamID lgID      G    AB     R     H   X2B   X3B    HR
##   <chr>     <int> <int> <fct>  <fct> <int> <int> <int> <int> <int> <int> <int>
## 1 abercda~   1871     1 TRO    NA        1     4     0     0     0     0     0
## 2 addybo01   1871     1 RC1    NA       25   118    30    32     6     0     0
## 3 allisar~   1871     1 CL1    NA       29   137    28    40     4     5     0
## 4 allisdo~   1871     1 WS3    NA       27   133    28    44    10     2     2
## 5 ansonca~   1871     1 RC1    NA       25   120    29    39    11     3     0
## # ... with 105,856 more rows, and 10 more variables: RBI <int>, SB <int>,
## #   CS <int>, BB <int>, SO <int>, IBB <int>, HBP <int>, SH <int>, SF <int>,
## #   GIDP <int>
```

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

dplyr

`filter()` - subset rows

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

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

```
## # A tibble: 4,817 x 22
##   playerID yearID stint teamID lgID      G    AB     R     H   X2B   X3B    HR
##   <chr>     <int> <int> <fct>  <fct> <int> <int> <int> <int> <int> <int> <int>
## 1 barklsa~   1887     1 PIT    NL       89   340    44    76    10     4     1
## 2 beeched~   1887     1 PIT    NL       41   169    15    41     8     0     2
## 3 bishobi~   1887     1 PIT    NL        3     9     0     0     0     0     0
## 4 brownto~   1887     1 PIT    NL       47   192    30    47     3     4     0
## 5 carrofr~   1887     1 PIT    NL      102   421    71   138    24    15     6
## # ... with 4,812 more rows, and 10 more variables: RBI <int>, SB <int>,
## #   CS <int>, BB <int>, SO <int>, IBB <int>, HBP <int>, SH <int>, SF <int>,
## #   GIDP <int>
```

dplyr

`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    HR
##   <chr>     <int> <int> <fct>  <fct> <int> <int> <int> <int> <int> <int> <int>
## 1 anderji~   2000     1 PIT    NL       27    50     5     7     1     0     0
## 2 arroybr~   2000     1 PIT    NL       21    21     2     3     2     0     0
## 3 avenbr01   2000     1 PIT    NL       72   148    18    37    11     0     5
## 4 benjami~   2000     1 PIT    NL       93   233    28    63    18     2     2
## 5 bensokr~   2000     1 PIT    NL       32    65     3     6     2     0     0
## # ... with 41 more rows, and 10 more variables: RBI <int>, SB <int>, CS <int>,
## #   BB <int>, SO <int>, IBB <int>, HBP <int>, SH <int>, SF <int>, GIDP <int>
```

dplyr

`arrange()` - reorder rows

```
#reorder by teamID
arrange(Batting, teamID)
```

```
## # A tibble: 105,861 x 22
##   playerID yearID stint teamID lgID      G    AB     R     H   X2B   X3B    HR
##   <chr>     <int> <int> <fct>  <fct> <int> <int> <int> <int> <int> <int> <int>
## 1 berrych~   1884     1 ALT    UA        7    25     2     6     0     0     0
## 2 brownji~   1884     1 ALT    UA       21    88    12    22     2     2     1
## 3 carropa~   1884     1 ALT    UA       11    49     4    13     1     0     0
## 4 connojo~   1884     1 ALT    UA        3    11     0     1     0     0     0
## 5 crosscl~   1884     1 ALT    UA        2     7     1     4     1     0     0
## # ... with 105,856 more rows, and 10 more variables: RBI <int>, SB <int>,
## #   CS <int>, BB <int>, SO <int>, IBB <int>, HBP <int>, SH <int>, SF <int>,
## #   GIDP <int>
```

dplyr

`arrange()` - reorder rows

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

```
## # A tibble: 105,861 x 22
##   playerID yearID stint teamID lgID      G    AB     R     H   X2B   X3B    HR
##   <chr>     <int> <int> <fct>  <fct> <int> <int> <int> <int> <int> <int> <int>
## 1 daisege~   1884     1 ALT    UA        1     4     0     0     0     0     0
## 2 crosscl~   1884     1 ALT    UA        2     7     1     4     1     0     0
## 3 manloch~   1884     1 ALT    UA        2     7     1     3     0     0     0
## 4 connojo~   1884     1 ALT    UA        3    11     0     1     0     0     0
## 5 shafff01   1884     1 ALT    UA        6    19     1     3     0     0     0
## # ... with 105,856 more rows, and 10 more variables: RBI <int>, SB <int>,
## #   CS <int>, BB <int>, SO <int>, IBB <int>, HBP <int>, SH <int>, SF <int>,
## #   GIDP <int>
```

dplyr

`arrange()` - reorder rows

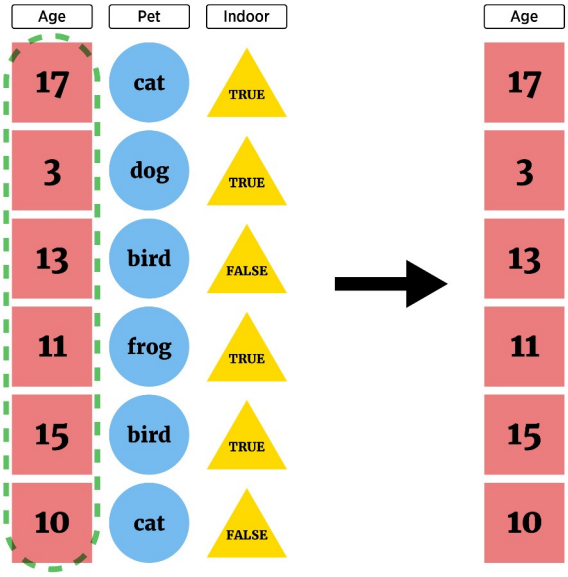
```
#descending instead
arrange(Batting, teamID, desc(G))
```

```
## # A tibble: 105,861 x 22
##   playerID yearID stint teamID lgID      G    AB     R     H   X2B   X3B    HR
##   <chr>     <int> <int> <fct>  <fct> <int> <int> <int> <int> <int> <int> <int>
## 1 smithge~   1884     1 ALT    UA       25   108     9    34     8     1     0
## 2 harrifr~   1884     1 ALT    UA       24    95    10    25     2     1     0
## 3 doughch~   1884     1 ALT    UA       23    85     6    22     5     0     0
## 4 murphjo~   1884     1 ALT    UA       23    94    10    14     1     0     0
## 5 brownji~   1884     1 ALT    UA       21    88    12    22     2     2     1
## # ... with 105,856 more rows, and 10 more variables: RBI <int>, SB <int>,
## #   CS <int>, BB <int>, SO <int>, IBB <int>, HBP <int>, SH <int>, SF <int>,
## #   GIDP <int>
```

Data manipulation idea

We may want to subset our full data set or create new data

- Look at only certain variables (**select columns**)



dplyr

`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
select(Batting, X2B)
```

```
## # A tibble: 105,861 x 1
##     X2B
##   <int>
## 1     0
## 2     6
## 3     4
## 4    10
## 5    11
## # ... with 105,856 more rows
```

dplyr

`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
select(Batting, playerID, X2B)
```

```
## # A tibble: 105,861 x 2
##   playerID    X2B
##   <chr>     <int>
## 1 abercda01     0
## 2 addybo01      6
## 3 allisar01     4
## 4 allisdo01    10
## 5 ansonca01    11
## # ... with 105,856 more rows
```

Aside: Piping or Chaining

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

```
arrange(select(filter(Batting, teamID == "PIT"), playerID, G, X2B), desc(X2B))
```

```
## # A tibble: 4,817 x 3
##   playerID      G   X2B
##   <chr>     <int> <int>
## 1 wanerpa01   154    62
## 2 wanerpa01   148    53
## 3 sanchfr01   157    53
## 4 wanerpa01   152    50
## 5 comorad01   152    47
## # ... with 4,812 more rows
```

Aside: Piping or Chaining

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

```
Batting %>% filter(teamID == "PIT") %>% select(playerID, G, X2B) %>% arrange(desc(X2B))
```

```
## # A tibble: 4,817 x 3
##   playerID      G   X2B
##   <chr>     <int> <int>
## 1 wanerpa01   154    62
## 2 wanerpa01   148    53
## 3 sanchfr01   157    53
## 4 wanerpa01   152    50
## 5 comorad01   152    47
## # ... with 4,812 more rows
```

Aside: Piping or Chaining

- Generically, pipe does the following

`x %>% f(y)` turns into `f(x,y)`

`x %>% f(y) %>% g(z)` turns into `g(f(x, y), z)`

- Can be used with functions outside the tidyverse if this structure works!

dplyr

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

- Many ways to select variables

```
#all columns between
Batting %>% select(X2B:HR)
```

```
## # A tibble: 105,861 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 105,856 more rows
```

dplyr

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

- Many ways to select variables

```
#all columns containing
Batting %>% select(contains("X"))
```

```
## # A tibble: 105,861 x 2
##     X2B   X3B
##   <int> <int>
## 1     0     0
## 2     6     0
## 3     4     5
## 4    10     2
## 5    11     3
## # ... with 105,856 more rows
```

dplyr

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

- Many ways to select variables

```
#all columns starting with
Batting %>% select(starts_with("X"))
```

```
## # A tibble: 105,861 x 2
##     X2B   X3B
##   <int> <int>
## 1     0     0
## 2     6     0
## 3     4     5
## 4    10     2
## 5    11     3
## # ... with 105,856 more rows
```

dplyr

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

- Many ways to select variables

```
#multiple selections
Batting %>% select(starts_with("X"), ends_with("ID"), G)
```

```
## # A tibble: 105,861 x 7
##     X2B   X3B playerID  yearID teamID lgID      G
##   <int> <int> <chr>      <int> <fct>  <fct> <int>
## 1     0     0 abercda01   1871 TRO    NA        1
## 2     6     0 addybo01    1871 RC1    NA       25
## 3     4     5 allisar01   1871 CL1    NA       29
## 4    10     2 allisdo01   1871 WS3    NA       27
## 5    11     3 ansonca01   1871 RC1    NA       25
## # ... with 105,856 more rows
```

dplyr

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

- May want to reorder variables

```
#reorder
Batting %>% select(playerID, HR, everything())
```

```
## # A tibble: 105,861 x 22
##   playerID    HR yearID stint teamID lgID      G    AB     R     H   X2B   X3B
##   <chr>    <int>  <int> <int> <fct>  <fct> <int> <int> <int> <int> <int> <int>
## 1 abercda~     0   1871     1 TRO    NA        1     4     0     0     0     0
## 2 addybo01     0   1871     1 RC1    NA       25   118    30    32     6     0
## 3 allisar~     0   1871     1 CL1    NA       29   137    28    40     4     5
## 4 allisdo~     2   1871     1 WS3    NA       27   133    28    44    10     2
## 5 ansonca~     0   1871     1 RC1    NA       25   120    29    39    11     3
## # ... with 105,856 more rows, and 10 more variables: RBI <int>, SB <int>,
## #   CS <int>, BB <int>, SO <int>, IBB <int>, HBP <int>, SH <int>, SF <int>,
## #   GIDP <int>
```

dplyr

`rename()` - rename variables

```
#rename our previous
Batting %>%
  select(starts_with("X"), ends_with("ID"), G) %>%
  rename("Doubles" = X2B, "Triples" = X3B)
```

```
## # A tibble: 105,861 x 7
##   Doubles Triples playerID  yearID teamID lgID      G
##     <int>   <int> <chr>      <int> <fct>  <fct> <int>
## 1       0       0 abercda01   1871 TRO    NA        1
## 2       6       0 addybo01    1871 RC1    NA       25
## 3       4       5 allisar01   1871 CL1    NA       29
## 4      10       2 allisdo01   1871 WS3    NA       27
## 5      11       3 ansonca01   1871 RC1    NA       25
## # ... with 105,856 more rows
```

`dplyr`

Cheat sheet

- Basic commands
  - `as_tibble()` - convert data frame to one with better printing
  - `filter()` - subset **rows**
  - `arrange()` - reorder **rows**
  - `select()` - subset **columns**
  - `rename()` - ranem **columns**
- Many `joins` to combine tibbles too! (Similar to SQL)

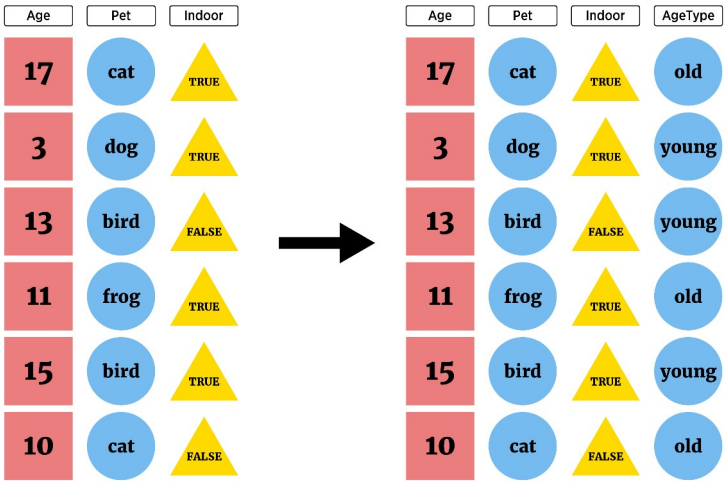
Activity

- **Manipulating Data Activity** instructions available on web
- Just the first part
- We'll send you to breakout rooms
- One TA or instructor in each room to help out
- Feel free to ask questions about anything you didn't understand as well!

Data manipulation idea

We may want to subset our full data set or create new data

- Create new variables



Creating New Variables

Given a data frame and an appropriate length vector (a new variable), you can use `cbind` (column bind) to add the variable to the dataframe

```
temp <- cbind(iris, extra = rep("a", 150))
str(temp)
```

```
## 'data.frame':    150 obs. of  6 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ extra       : Factor w/ 1 level "a": 1 1 1 1 1 1 1 1 1 1 ...
```

Creating New Variables

Or simply add as a named (list) element!

```
iris$extra <- rep("a", 150)
str(iris)
```

```
## tibble [150 x 6] (S3: tbl_df/tbl/data.frame)
##  $ Sepal.Length: num [1:150] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num [1:150] 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num [1:150] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num [1:150] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ extra       : chr [1:150] "a" "a" "a" "a" ...
```

Creating New Variables

Better method: use `dplyr`!

- `mutate()` - add newly created **column(s)** to current data frame
- `transmute()` - create new data frame with created variable(s)

Creating New Variables

Better method: use `dplyr`!

- `mutate()` - add newly created **column(s)** to current data frame
- `transmute()` - create new data frame with created variable(s)
- Syntax:

```
mutate(data, newVarName = functionOfData, newVarName2 =
functionOfData, ...)
```

Creating New Variables

- Consider a data set on movie ratings

```
library(fivethirtyeight)
```

```
## Some larger datasets need to be installed separately, like senators and
## house_district_forecast. To install these, we recommend you install the
## fivethirtyeightdata package by running:
## install.packages('fivethirtyeightdata', repos =
## 'https://fivethirtyeightdata.github.io/drat/', type = 'source')
```

```
fandango
```

```
## # A tibble: 146 x 23
##   film   year rottentomatoes rottentomatoes_~ metacritic metacritic_user  imdb
##   <chr> <dbl>          <int>            <int>      <int>           <dbl> <dbl>
## 1 Aven~  2015             74               86         66             7.1   7.8
## 2 Cind~  2015             85               80         67             7.5   7.1
## 3 Ant-~  2015             80               90         64             8.1   7.8
## 4 Do Y~  2015             18               84         22             4.7   5.4
## 5 Hot ~  2015             14               28         29             3.4   5.1
## # ... with 141 more rows, and 16 more variables: fandango_stars <dbl>,
## #   fandango_ratingvalue <dbl>, rt_norm <dbl>, rt_user_norm <dbl>,
## #   metacritic_norm <dbl>, metacritic_user_nom <dbl>, imdb_norm <dbl>,
## #   rt_norm_round <dbl>, rt_user_norm_round <dbl>, metacritic_norm_round <dbl>,
## #   metacritic_user_norm_round <dbl>, imdb_norm_round <dbl>,
## #   metacritic_user_vote_count <int>, imdb_user_vote_count <int>,
## #   fandango_votes <int>, fandango_difference <dbl>
```

Creating New Variables

- `mutate()` - add newly created **column(s)** to current data frame

```
##Create an average rottentomatoes score variable
fandango %>% mutate(avgRotten = (rottentomatoes + rottentomatoes_user)/2)
```

```
## # A tibble: 146 x 24
##   film   year rottentomatoes rottentomatoes_~ metacritic metacritic_user  imdb
##   <chr> <dbl>          <int>            <int>      <int>           <dbl> <dbl>
## 1 Aven~  2015             74               86         66             7.1   7.8
## 2 Cind~  2015             85               80         67             7.5   7.1
## 3 Ant-~  2015             80               90         64             8.1   7.8
## 4 Do Y~  2015             18               84         22             4.7   5.4
## 5 Hot ~  2015             14               28         29             3.4   5.1
## # ... with 141 more rows, and 17 more variables: fandango_stars <dbl>,
## #   fandango_ratingvalue <dbl>, rt_norm <dbl>, rt_user_norm <dbl>,
## #   metacritic_norm <dbl>, metacritic_user_nom <dbl>, imdb_norm <dbl>,
## #   rt_norm_round <dbl>, rt_user_norm_round <dbl>, metacritic_norm_round <dbl>,
## #   metacritic_user_norm_round <dbl>, imdb_norm_round <dbl>,
## #   metacritic_user_vote_count <int>, imdb_user_vote_count <int>,
## #   fandango_votes <int>, fandango_difference <dbl>, avgRotten <dbl>
```

Creating New Variables

- `mutate()` - add newly created **column(s)** to current data frame

```
#can't see it!
fandango %>% mutate(avgRotten = (rottentomatoes + rottentomatoes_user)/2) %>%
  select(film, year, avgRotten, everything())
```

```
## # A tibble: 146 x 24
##   film   year avgRotten rottentomatoes rottentomatoes_~ metacritic
##   <chr> <dbl>     <dbl>          <int>            <int>      <int>
## 1 Aven~  2015      80               74               86         66
## 2 Cind~  2015      82.5             85               80         67
## 3 Ant-~  2015      85               80               90         64
## 4 Do Y~  2015      51               18               84         22
## 5 Hot ~  2015      21               14               28         29
## # ... with 141 more rows, and 18 more variables: metacritic_user <dbl>,
## #   imdb <dbl>, fandango_stars <dbl>, fandango_ratingvalue <dbl>,
## #   rt_norm <dbl>, rt_user_norm <dbl>, metacritic_norm <dbl>,
## #   metacritic_user_nom <dbl>, imdb_norm <dbl>, rt_norm_round <dbl>,
## #   rt_user_norm_round <dbl>, metacritic_norm_round <dbl>,
## #   metacritic_user_norm_round <dbl>, imdb_norm_round <dbl>,
## #   metacritic_user_vote_count <int>, imdb_user_vote_count <int>,
## #   fandango_votes <int>, fandango_difference <dbl>
```

Creating New Variables

- `mutate()` - add newly created **column(s)** to current data frame
- Add more than one variable

```
fandango %>%
  mutate(avgRotten = (rottentomatoes + rottentomatoes_user)/2,
         avgMeta = (metacritic_norm + metacritic_user_nom)/2) %>%
  select(film, year, avgRotten, avgMeta, everything())
```

```
## # A tibble: 146 x 25
##   film   year avgRotten avgMeta rottentomatoes rottentomatoes_~ metacritic
##   <chr> <dbl>     <dbl>   <dbl>          <int>            <int>      <int>
## 1 Aven~  2015      80      3.42             74               86         66
## 2 Cind~  2015      82.5    3.55             85               80         67
## 3 Ant-~  2015      85      3.62             80               90         64
## 4 Do Y~  2015      51      1.72             18               84         22
## 5 Hot ~  2015      21      1.58             14               28         29
## # ... with 141 more rows, and 18 more variables: metacritic_user <dbl>,
## #   imdb <dbl>, fandango_stars <dbl>, fandango_ratingvalue <dbl>,
## #   rt_norm <dbl>, rt_user_norm <dbl>, metacritic_norm <dbl>,
## #   metacritic_user_nom <dbl>, imdb_norm <dbl>, rt_norm_round <dbl>,
## #   rt_user_norm_round <dbl>, metacritic_norm_round <dbl>,
## #   metacritic_user_norm_round <dbl>, imdb_norm_round <dbl>,
## #   metacritic_user_vote_count <int>, imdb_user_vote_count <int>,
## #   fandango_votes <int>, fandango_difference <dbl>
```

Creating New Variables

- `transmute()` - create new data frame with created variable(s)

```
#transmute will keep the new variable(s) only
fandango %>% transmute(avgRotten = (rottentomatoes + rottentomatoes_user)/2)
```

```
## # A tibble: 146 x 1
##   avgRotten
##       <dbl>
## 1      80
## 2      82.5
## 3      85
## 4      51
## 5      21
## # ... with 141 more rows
```

Creating New Variables

- `transmute()` - create new data frame with created variable(s)

```
#transmute will keep the new variable(s) only
fandango %>% transmute(avgRotten = (rottentomatoes + rottentomatoes_user)/2,
                       avgMeta = (metacritic_norm + metacritic_user_nom)/2)
```

```
## # A tibble: 146 x 2
##   avgRotten avgMeta
##       <dbl>   <dbl>
## 1      80      3.42
## 2      82.5    3.55
## 3      85      3.62
## 4      51      1.72
## 5      21      1.58
## # ... with 141 more rows
```

Creating New Variables

`mutate` and `transmute` can also use 'window' functions

- Functions that take a vector of values and return another vector of values (see Cheat sheet)

```
fandango %>% select(rottentomatoes) %>% mutate(cumulativeSum = cumsum(rottentomatoes))
```

```
## # A tibble: 146 x 2
##   rottentomatoes cumulativeSum
##            <int>         <int>
## 1             74            74
## 2             85           159
## 3             80           239
## 4             18           257
## 5             14           271
## # ... with 141 more rows
```

Creating New Variables

`mutate` and `transmute` can also use some statistical functions

```
fandango %>% select(rottentomatoes) %>%
  mutate(avg = mean(rottentomatoes), sd = sd(rottentomatoes))
```

```
## # A tibble: 146 x 3
##   rottentomatoes   avg    sd
##            <int> <dbl> <dbl>
## 1             74  60.8  30.2
## 2             85  60.8  30.2
## 3             80  60.8  30.2
## 4             18  60.8  30.2
## 5             14  60.8  30.2
## # ... with 141 more rows
```

Creating New Variables

`mutate` and `transmute` can also use some statistical functions

- `group_by` to create summaries for groups (more on this later)

```
fandango %>% select(year, rottentomatoes) %>%
  group_by(year) %>% mutate(avg = mean(rottentomatoes), sd = sd(rottentomatoes))
```

```
## # A tibble: 146 x 4
## # Groups:   year [2]
##    year rottentomatoes   avg    sd
##   <dbl>          <int> <dbl> <dbl>
## 1  2015             74  58.4  30.3
## 2  2015             85  58.4  30.3
## 3  2015             80  58.4  30.3
## 4  2015             18  58.4  30.3
## 5  2015             14  58.4  30.3
## # ... with 141 more rows
```

Creating New Variables

Conditional Execution with If then, If then else

- Often want to execute statements conditionally to create a variable
- `if then else` syntax

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

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

```
#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

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

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

```r
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"
```

Creating New Variables

Conditional Execution with If then, If then else

- Consider built-in data set `airquality`
  - daily air quality measurements in New York
  - from May (Day 1) to September (Day 153) in 1973

Creating New Variables

Conditional Execution with If then, If then else

- Consider built-in data set `airquality`

```
airquality <- as_tibble(airquality)
airquality
```

```
## # A tibble: 153 x 6
##   Ozone Solar.R  Wind  Temp Month   Day
##   <int>   <int> <dbl> <int> <int> <int>
## 1    41     190   7.4    67     5     1
## 2    36     118   8      72     5     2
## 3    12     149  12.6    74     5     3
## 4    18     313  11.5    62     5     4
## 5    NA      NA  14.3    56     5     5
## # ... with 148 more rows
```

Creating New Variables

Conditional Execution with If then, If then else

Want to code a wind category variable

- high wind days (wind $\geq$ 15mph)
- windy days (10mph $\leq$ wind < 15mph)
- lightwind days (6mph $\leq$ wind < 10mph)
- calm days (wind $\leq$ 6mph)

Creating New Variables

Conditional Execution with If then, If then else

Want to code a wind category variable

Issue: `if(condition)` can only take in a single comparison

```
if(airquality$Wind >= 15) {
  "High Wind"
  }
```

```
## Warning in if (airquality$Wind >= 15) {: the condition has length > 1 and only
## the first element will be used
```

Creating New Variables

Conditional Execution with If then, If then else

Want to code a wind category variable

- high wind days (15mph $\leq$ wind)
- windy days (10mph $\leq$ wind < 15mph)
- lightwind days (6mph $\leq$ wind < 10mph)
- calm days (wind $\leq$ 6mph)
- `ifelse()` is a vectorized version of `if then else`
- Syntax

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

Creating New Variables

Vectorized if else

```
ifelse(airquality$Wind >= 15, "HighWind",
          ifelse(airquality$Wind >= 10, "Windy",
                 ifelse(airquality$Wind >= 6, "LightWind", "Calm")))
```

```
##   [1] "LightWind" "LightWind" "Windy"     "Windy"     "Windy"     "Windy"
##   [7] "LightWind" "Windy"     "HighWind"  "LightWind" "LightWind" "LightWind"
##  [13] "LightWind" "Windy"     "Windy"     "Windy"     "Windy"     "HighWind"
##  [19] "Windy"     "LightWind" "LightWind" "HighWind"  "LightWind" "Windy"
##  [25] "HighWind"  "Windy"     "LightWind" "Windy"     "Windy"     "Calm"
##  [31] "LightWind" "LightWind" "LightWind" "HighWind"  "LightWind" "LightWind"
##  [37] "Windy"     "LightWind" "LightWind" "Windy"     "Windy"     "Windy"
##  [43] "LightWind" "LightWind" "Windy"     "Windy"     "Windy"     "HighWind"
##  [49] "LightWind" "Windy"     "Windy"     "LightWind" "Calm"      "Calm"
##  [55] "LightWind" "LightWind" "LightWind" "Windy"     "Windy"     "Windy"
##  [61] "LightWind" "Calm"      "LightWind" "LightWind" "Windy"     "Calm"
##  [67] "Windy"     "Calm"      "LightWind" "Calm"      "LightWind" "LightWind"
##  [73] "Windy"     "Windy"     "Windy"     "Windy"     "LightWind" "Windy"
##  [79] "LightWind" "Calm"      "Windy"     "LightWind" "LightWind" "Windy"
##  [85] "LightWind" "LightWind" "LightWind" "Windy"     "LightWind" "LightWind"
##  [91] "LightWind" "LightWind" "LightWind" "Windy"     "LightWind" "LightWind"
##  [97] "LightWind" "Calm"      "Calm"      "Windy"     "LightWind" "LightWind"
## [103] "Windy"     "Windy"     "Windy"     "LightWind" "Windy"     "Windy"
## [109] "LightWind" "LightWind" "Windy"     "Windy"     "HighWind"  "Windy"
## [115] "Windy"     "LightWind" "Calm"      "LightWind" "Calm"      "LightWind"
## [121] "Calm"      "LightWind" "LightWind" "LightWind" "Calm"      "Calm"
## [127] "Calm"      "LightWind" "HighWind"  "Windy"     "Windy"     "Windy"
## [133] "LightWind" "Windy"     "HighWind"  "LightWind" "Windy"     "Windy"
## [139] "LightWind" "Windy"     "Windy"     "Windy"     "LightWind" "Windy"
## [145] "LightWind" "Windy"     "Windy"     "HighWind"  "LightWind" "Windy"
## [151] "Windy"     "LightWind" "Windy"
```

Creating New Variables

Vectorized if else

- Can use with `transmute()` or `mutate()`

```
mutate(airquality, status = ifelse(airquality$Wind >= 15, "HighWind",
                                   ifelse(airquality$Wind >= 10, "Windy",
                                          ifelse(airquality$Wind >= 6, "LightWind", "Calm")))
)
```

```
## # A tibble: 153 x 7
##   Ozone Solar.R  Wind  Temp Month   Day status
##   <int>   <int> <dbl> <int> <int> <int> <chr>
## 1    41     190   7.4    67     5     1 LightWind
## 2    36     118   8      72     5     2 LightWind
## 3    12     149  12.6    74     5     3 Windy
## 4    18     313  11.5    62     5     4 Windy
## 5    NA      NA  14.3    56     5     5 Windy
## # ... with 148 more rows
```

`dplyr` package

- Basic commands
  - `as_tibble()` - 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

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
```

```
## # 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
```

```
inner_join(a, b)
```

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

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

```
## # 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
```

```
full_join(a, b)
```

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

```
## # A tibble: 4 x 3
##   color    num size
##   <chr>  <int> <chr>
## 1 green      1 S
## 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
```

```
## # 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
```

```
left_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 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
```

```
## # 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
```

```
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
```

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
```

```
## # 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
```

```
semi_join(a, b)
```

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

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

```
## # 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
```

```
anti_join(a, b)
```

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

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

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
```

```
## # 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
```

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

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

Overview of `dplyr` package cheatsheet

- Basic commands
  - `as_tibble()` - 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
- The second part
- We'll send you to breakout rooms
- One TA or instructor in each room to help out
- Feel free to ask questions about anything you didn't understand as well!