NC STATE UNIVERSITY

Manipulating Data: Logical Statements & `dplyr`

Justin Post

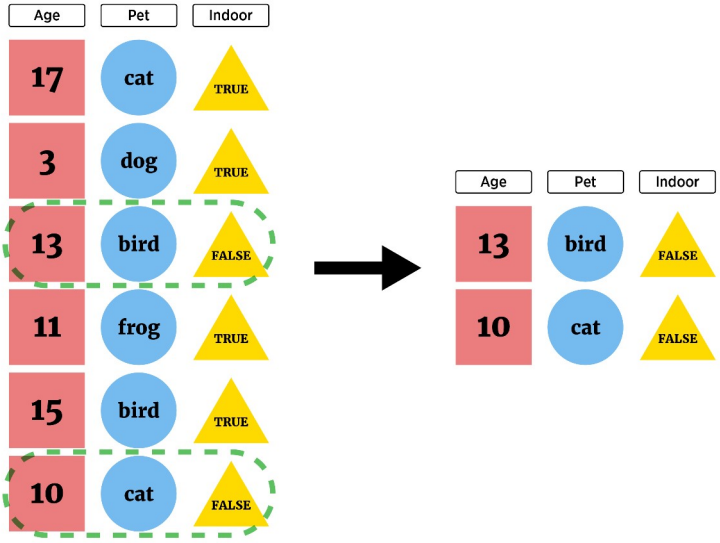
Where are we at?

- Data manipulation idea
- Documenting with Markdown
- Logical statements
- `dplyr`
- Creating new variables
    - Conditional execution (if then)
    - For loops
    - Vectorized functions
- Reshaping Data

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

Goal: Subset rows or columns

- **logical statement** - comparison that resolves as `TRUE` or `FALSE`

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

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

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

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

```
4 >= 1
```

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

```
4 != 1
```

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

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

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

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

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

Logical statements

Goal: Subset rows or columns

- **logical statement** - comparison that 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

Goal: Subset rows or columns

- Consider the built-in `iris` dataframe

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          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          5.0         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
```

Logical statements

Compound logicals via **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

Compound logicals via **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

Compound logicals via **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
```

Logical statements

Goal: Subset rows or columns

- Only pull out large petal setosa flowers

```
(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
```

Logical statements

Goal: Subset rows or columns

- Only pull out large petal setosa flowers

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

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 6           5.4         3.9          1.7         0.4  setosa
## 24          5.1         3.3          1.7         0.5  setosa
## 27          5.0         3.4          1.6         0.4  setosa
## 44          5.0         3.5          1.6         0.6  setosa
## 45          5.1         3.8          1.9         0.4  setosa
```

Logical statements

Goal: Subset rows or columns

- Only pull out large petal setosa flowers
- Easier with `subset` or `filter`!

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

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.4         3.9          1.7         0.4  setosa
## 2          5.1         3.3          1.7         0.5  setosa
## 3          5.0         3.4          1.6         0.4  setosa
## 4          5.0         3.5          1.6         0.6  setosa
## 5          5.1         3.8          1.9         0.4  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: 107,429 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 107,424 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 1.059e+05 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 1.059e+05 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

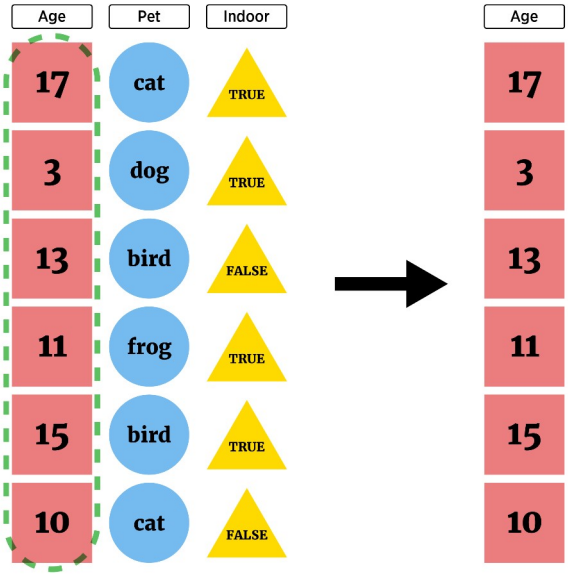
```
#descending instead
arrange(Batting, teamID, desc(G))
```

```
## # 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 1.059e+05 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 1.059e+05 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 1.059e+05 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 1.059e+05 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 1.059e+05 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 1.059e+05 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 1.059e+05 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**
- Many `joins` to combine tibbles too! (Similar to SQL)

Recap/Next Up!

- Data manipulation idea
- Documenting with Markdown
- Logical statements
- `dplyr`
- Creating new variables
    - Conditional execution (if then)
    - For loops
    - Vectorized functions
- Reshaping Data