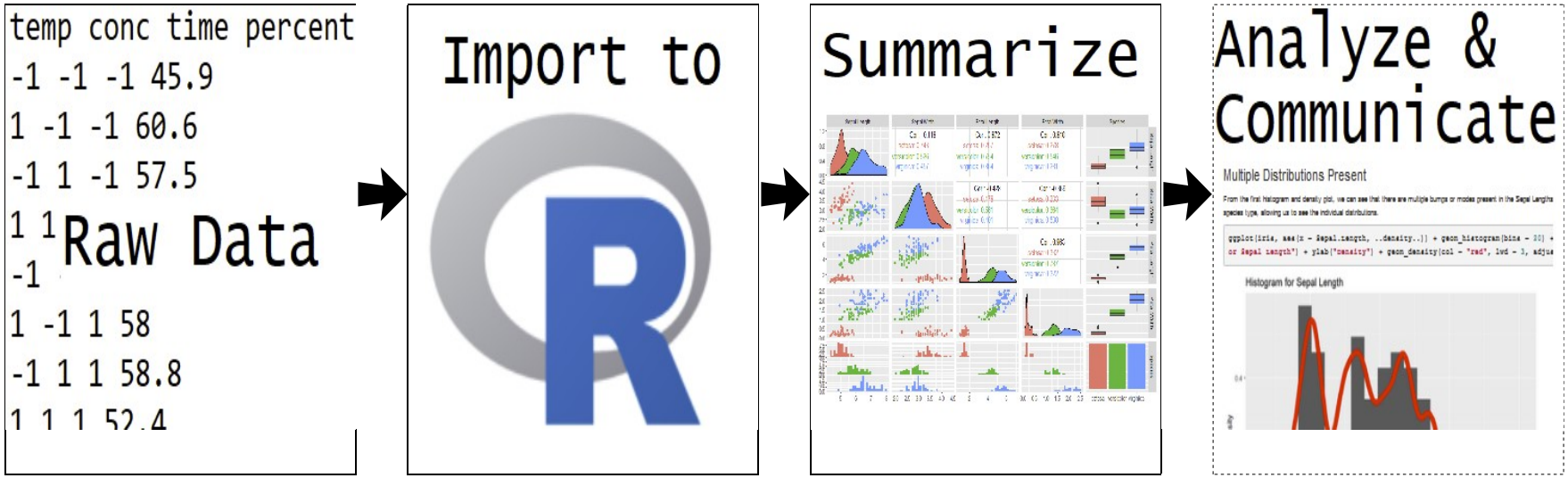
NC STATE UNIVERSITY

Manipulating Data and Documenting with Markdown

Justin Post

What is this course about?

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



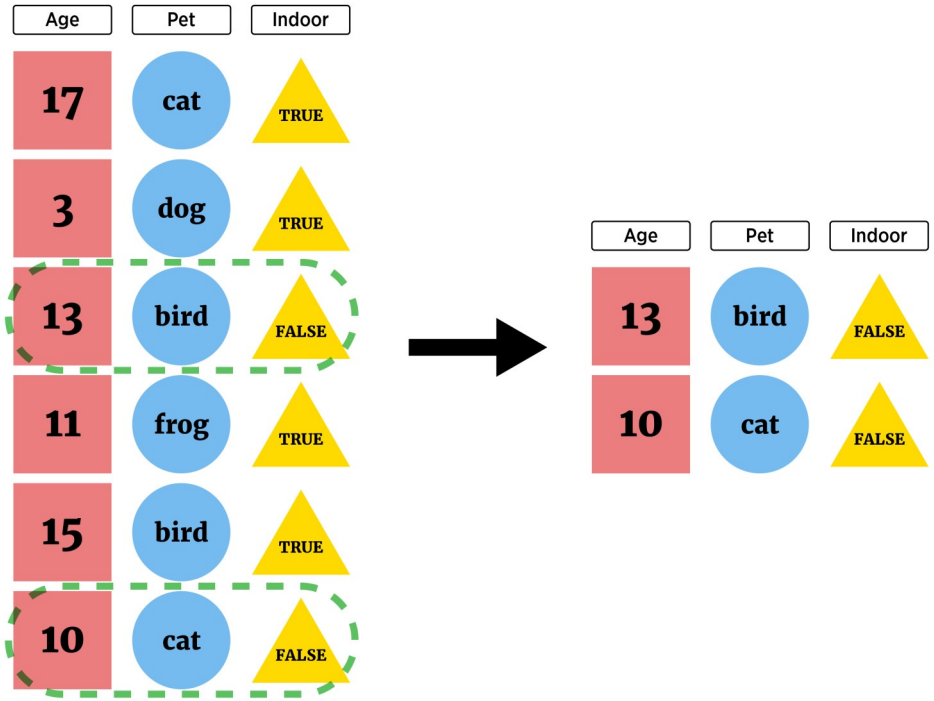
Where do we start?

- Data manipulation idea
- Documenting with Markdown
- Logical statements
- `dplyr`, `tidyr`, and creating new variables

Data manipulation idea

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

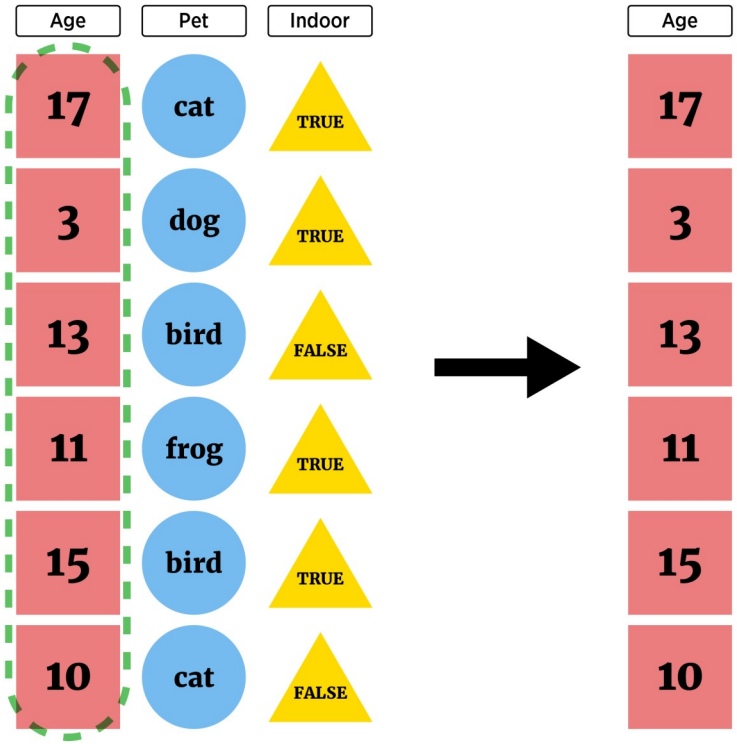
- Grab only certain types of observations (**filter rows**)



Data manipulation idea

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

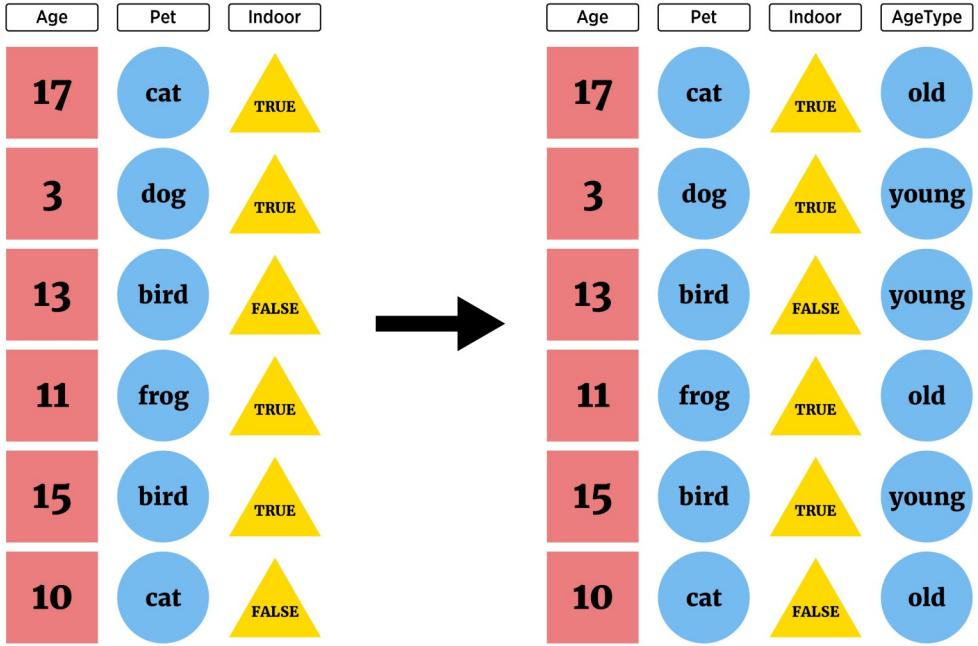
- Look at only certain variables (**select columns**)



Data manipulation idea

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

- Create new variables



Data manipulation idea

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

- Vital to make your work reproducible!
- Traditional documentation through comments (`#` in R) in script
- May have heard of JUPYTER notebooks
- R Markdown - built in notebook for R studio

Documenting with Markdown

- R Markdown = Digital “Notebook”: Program that weaves word processing and code.
- Designed to be used in three ways (R for Data Science)

Documenting with Markdown

- R Markdown = Digital “Notebook”: Program that weaves word processing and code.
- Designed to be used in three ways (R for Data Science)
  - Communicating to decision makers (focus on conclusions not code)
  - Collaborating with other data scientists (including future you!)
  - As environment to do data science (documents what you did and what you were thinking)

Markdown Verbage

- May have heard of HTML (HyperText Mark-up Language)
  - Write plain text that the browser interprets and renders

Markdown Verbage

- May have heard of HTML (HyperText Mark-up Language)
  - Write plain text that the browser interprets and renders
- Markdown is a specific markup language
  - Easier syntax
  - Not as powerful
- Any plain text file can be used (.Rmd extension associates it with R Studio)

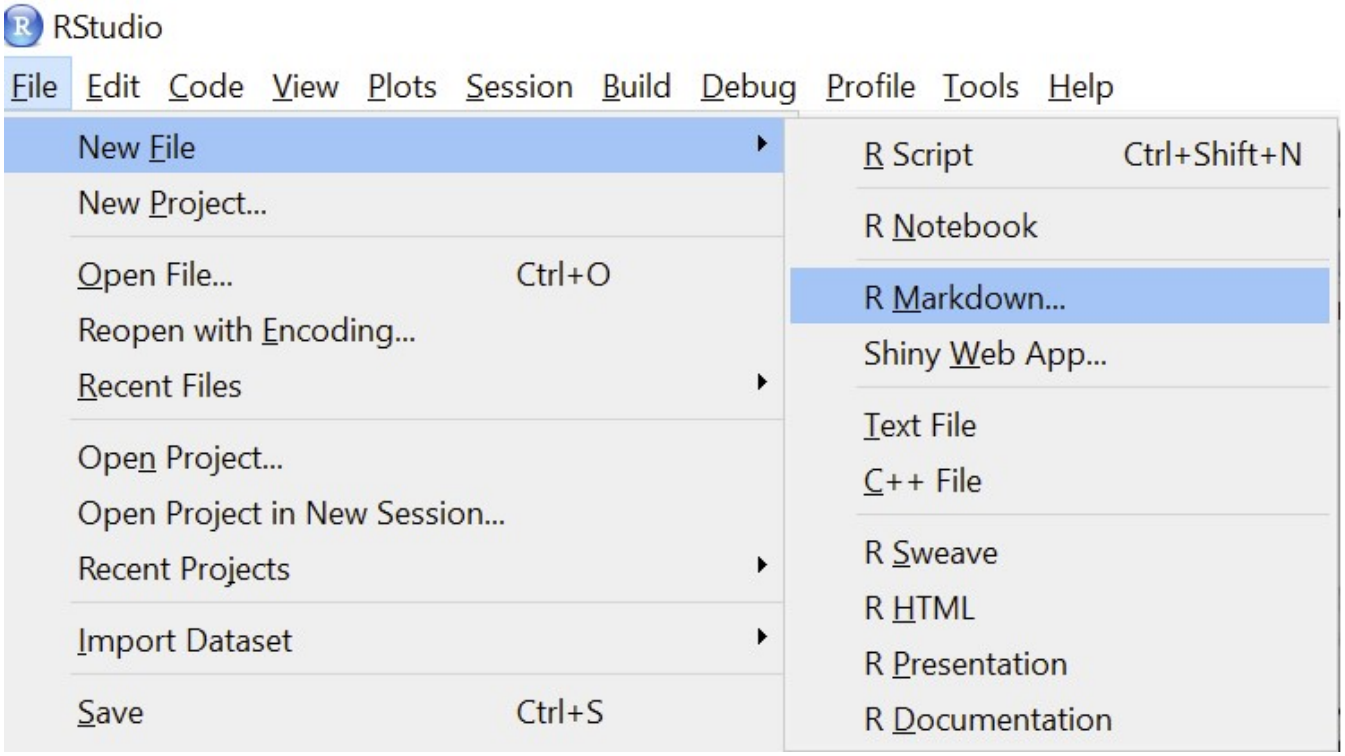
R Markdown

R Markdown file contains three important types of content:

1. (Optional) YAML header surrounded by ---s
2. Chunks of R code
3. Text mixed with simple text formatting instructions

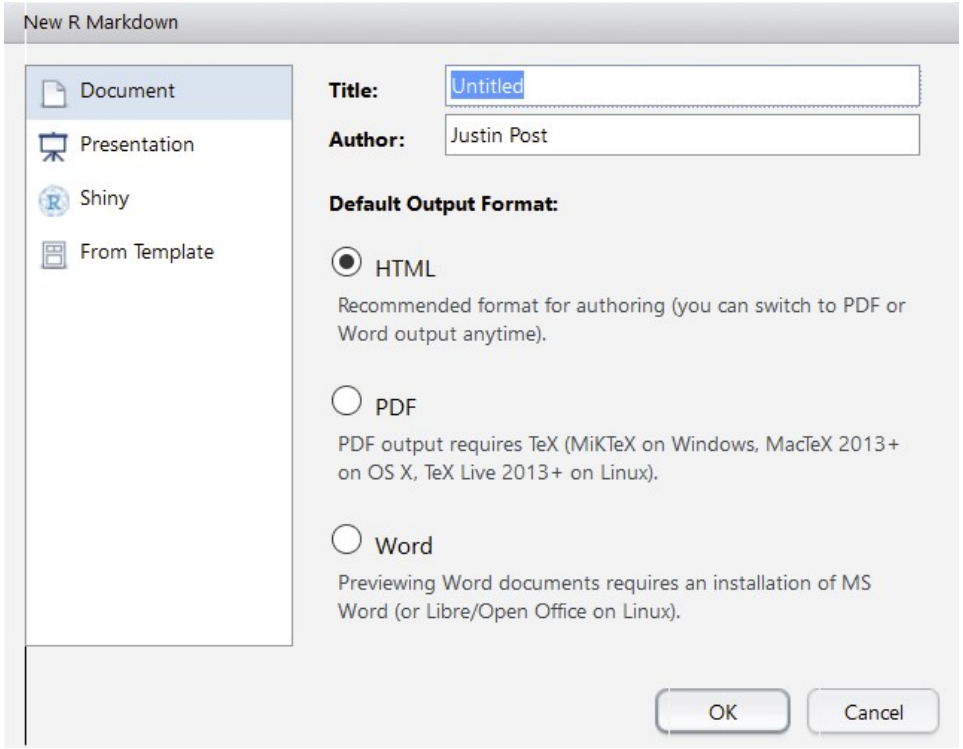
Creating an R Markdown Document

- R Studio makes it easy!



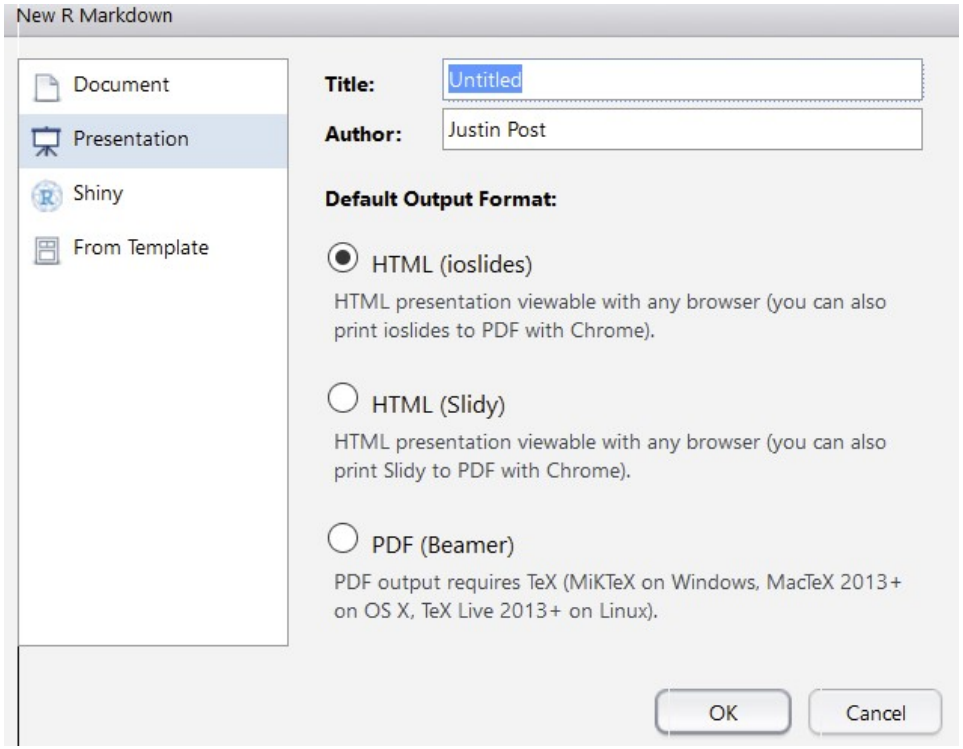
Creating an R Markdown Document

- Commonly used document types can be created



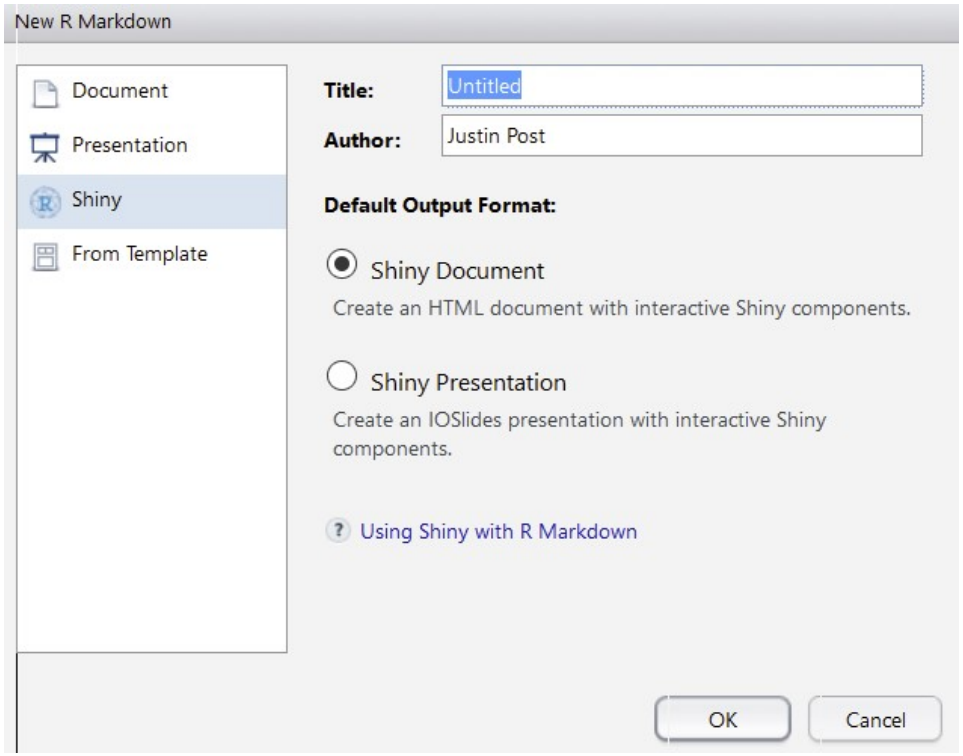
Creating an R Markdown Document

- Slide presentations



Creating an R Markdown Document

- Truly Interactive Documents/Pages (require R backend)



R Markdown - YAML Header

- Define settings for document

```
---
title: "Untitled"
author: "First Last"
date: "xxxx"
output: html_document
---
```

- CTRL/CMD + Shift + k **knits** (creates the output document) via this info

R Markdown - Code Chunks

- Below YAML header: ‘r chunk’

````
```{r ggplot,eval=FALSE}
select(iris, Sepal.Width)
ggplot(iris, aes(x = Sepal.Width, y = Sepal.Length)) +
geom_point()
```
````

- Start code chunk by typing ```{r} out or with CTRL/CMD + Alt/Option + I
- Code will be executed when document is created
- Can specify options on individual code chunks

R Markdown - Syntax

- Below code chunk is plain text with markdown sytnax

```
## R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax
for authoring HTML, PDF, and MS Word documents. For more details on
using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that
includes both content as well as the output of any embedded R code
chunks within the document.
```

- When file created, “##” becomes a header, “<…>” a link, and `**Knit**` bold font

R Markdown - Syntax

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document.

Where do we go from here?

Cheat sheet gives everything you need! We'll briefly investigate:

- Markdown syntax
- Code chunks and their options
- Changing type of output

R Markdown syntax

- `# Header 1` becomes a large font header
- `## Header 2` becomes a slightly smaller font header
- Goes to 6 headers
    - Use of headers can automatically create a Table of Contents!
- `**bold**` **and** `__bold__`
- `` `code` `` becomes `code`

R Markdown syntax

- Can do lists: be sure to end each line with two spaces!
    - Indent sub lists four spaces

```
* unordered list
* item 2
    + sub-item 1
    + sub-item 2

1. ordered list
2. item 2
    + sub-item 1
    + sub-item 2
```

- unordered list
- item 2
    - sub-item 1
    - sub-item 2

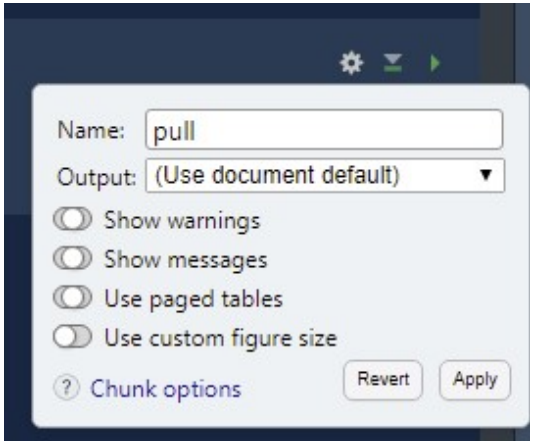
1. ordered list
2. item 2
    - sub-item 1
    - sub-item 2

Code chunks and their options

- Any R code can go into the chunk
- Chunks evaluate sequentially (can use output from prior chunk)
- Code can be added in line: Ex: The Iris dataset has 150 observations
- Added by beginning with back-tick `r` and ending with a back-tick: Iris has `` `r length(iris$Sepal.Length)` ``

Code chunks and their options

- Many options depending on chunk purpose!
- Can hide/show code with `echo = FALSE/TRUE`
- Can choose if code is evaluated with `eval = TRUE/FALSE`
- `message = TRUE/FALSE` and `warning = TRUE/FALSE` can turn on/off displaying messages/warnings



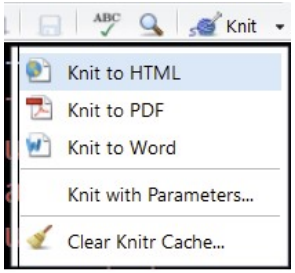
Changing type of output

R Markdown really flexible!

Changing type of output

Change output type in the YAML header:

- Use CTRL/CMD + Shift + k or the Knit menu:



- Use code explicity:
  `rmarkdown::render("file.Rmd", output_format = "html_document")`

- We'll just output to HTML for simplicity!

Quick Examples

- Go to the course files page and try Exercise 5 - Markdown

`tidyverse` for data manipulations

Now we can document everything: let's manipulate some data!

Overview of dplyr and tidyr packages

- `dplyr` 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`
- 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

`as_tibble()` - tidy data frame

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

```
#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
```

`as_tibble()` - tidy data frame

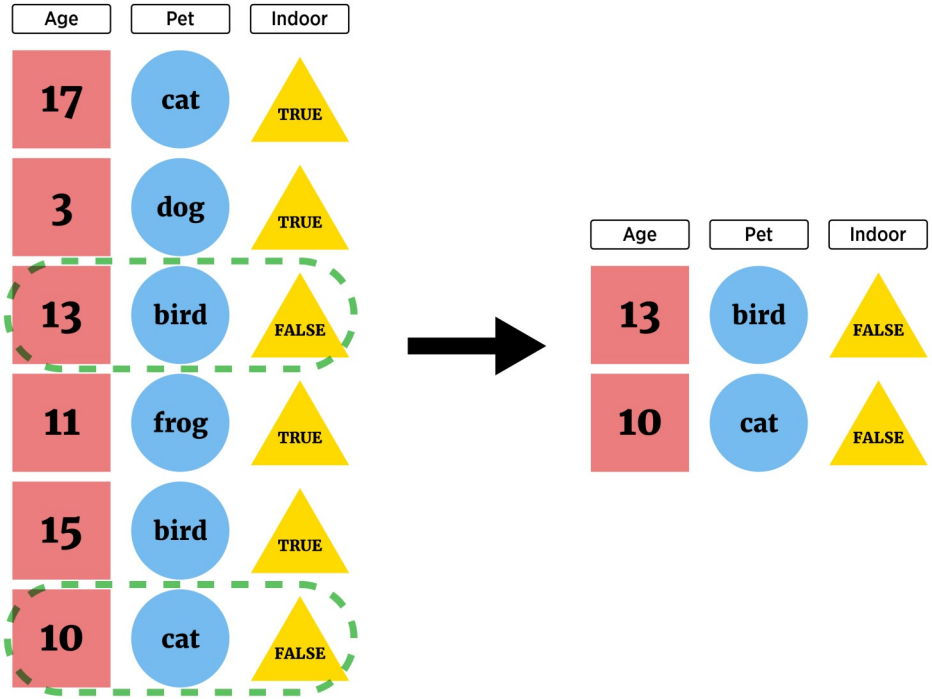
- Just 'wrap' a standard R data frame

```
myBatting <- as_tibble(Batting); myBatting
```

```
## # 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 abercda01   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 allisar01   1871     1 CL1    NA       29   137    28    40     4     5     0
## 4 allisdo01   1871     1 WS3    NA       27   133    28    44    10     2     2
## 5 ansonca01   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>
```

`filter()` - subset rows or columns

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



Filtering Rows Requires Logical Conditions

- **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
```

Filtering Rows Requires Logical Conditions

- **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

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

```
myBatting$G > 20 #vector indicating Games > 20
```

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

`filter()` - subset rows or columns

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

```
filter(myBatting, G > 20)
```

```
## # A tibble: 69,441 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 addybo01    1871     1 RC1    NA       25   118    30    32     6     0     0
## 2 allisar01   1871     1 CL1    NA       29   137    28    40     4     5     0
## 3 allisdo01   1871     1 WS3    NA       27   133    28    44    10     2     2
## 4 ansonca01   1871     1 RC1    NA       25   120    29    39    11     3     0
## 5 barnero01   1871     1 BS1    NA       31   157    66    63    10     9     0
## # ... with 69,436 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>
```

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

- Pull out those that played more than 20 games and played in 2015

```
(myBatting$G > 20) & (myBatting$yearID == 2015)
```

```
##     [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##    [13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##    [25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##    [37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE 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 FALSE FALSE FALSE FALSE FALSE FALSE
##   [157] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##   [169] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##   [181] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##   [193] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##   [205] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##   [217] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##   [229] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

`filter()` - subset rows or columns

- Pull out those that played more than 20 games and played in 2015

```
filter(myBatting, (G > 20) & (yearID == 2015))
```

```
## # A tibble: 949 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 aardsda01   2015     1 ATL    NL       33     1     0     0     0     0     0
## 2 abadfe01    2015     1 OAK    AL       62     0     0     0     0     0     0
## 3 abreujo02   2015     1 CHA    AL      154   613    88   178    34     3    30
## 4 ackledu01   2015     1 SEA    AL       85   186    22    40     8     1     6
## 5 ackledu01   2015     2 NYA    AL       23    52     6    15     3     2     4
## # ... with 944 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>
```

`filter()` - subset rows or columns

- `%in%` to choose any observations within a certain set

```
filter(myBatting, teamID %in% c("ATL", "PIT", "WSH"))
```

```
## # A tibble: 7,035 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 barklsa01   1887     1 PIT    NL       89   340    44    76    10     4     1
## 2 beeched01   1887     1 PIT    NL       41   169    15    41     8     0     2
## 3 bishobi01   1887     1 PIT    NL        3     9     0     0     0     0     0
## 4 brownto01   1887     1 PIT    NL       47   192    30    47     3     4     0
## 5 carrofr01   1887     1 PIT    NL      102   421    71   138    24    15     6
## # ... with 7,030 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>
```

`arrange()` - reorder rows

```
#reorder by teamID
arrange(myBatting, 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 berrych01   1884     1 ALT    UA        7    25     2     6     0     0     0
## 2 brownji01   1884     1 ALT    UA       21    88    12    22     2     2     1
## 3 carropa01   1884     1 ALT    UA       11    49     4    13     1     0     0
## 4 connojo01   1884     1 ALT    UA        3    11     0     1     0     0     0
## 5 crosscl01   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>
```

`arrange()` - reorder rows

```
#get secondary arrangement as well
arrange(myBatting, 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 daisege01   1884     1 ALT    UA        1     4     0     0     0     0     0
## 2 crosscl01   1884     1 ALT    UA        2     7     1     4     1     0     0
## 3 manloch01   1884     1 ALT    UA        2     7     1     3     0     0     0
## 4 connojo01   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>
```

`arrange()` - reorder rows

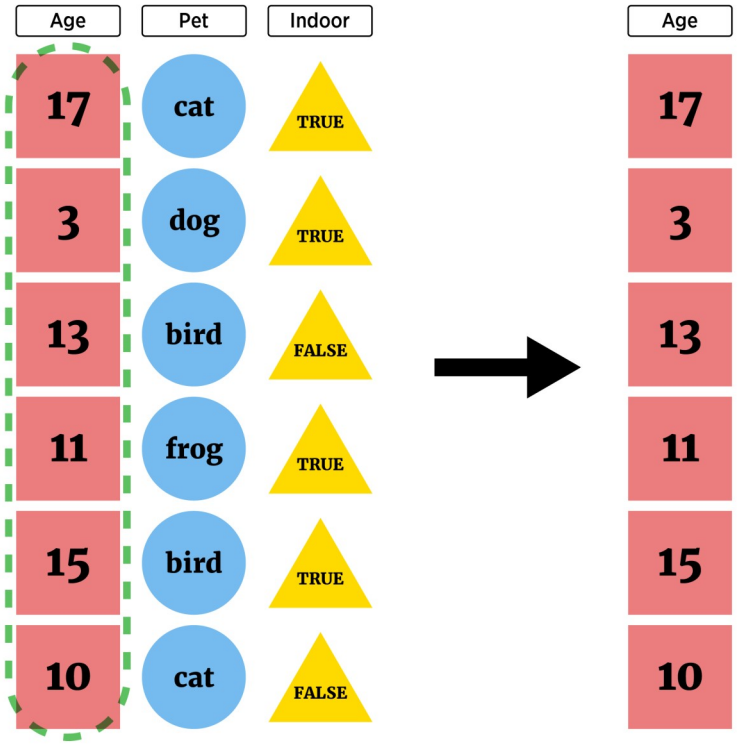
```
#descending instead
arrange(myBatting, teamID, desc(G))
```

```
## # 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 smithge01   1884     1 ALT    UA       25   108     9    34     8     1     0
## 2 harrifr01   1884     1 ALT    UA       24    95    10    25     2     1     0
## 3 doughch01   1884     1 ALT    UA       23    85     6    22     5     0     0
## 4 murphjo01   1884     1 ALT    UA       23    94    10    14     1     0     0
## 5 brownji01   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**)



`select()` - subset columns

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

```
#Choose a single column by name
select(myBatting, X2B)
```

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

`select()` - subset columns

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

```
#Choose a single column by name
select(myBatting, 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(myBatting, 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

```
myBatting %>%
  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!

`select()` - subset columns

- Great functionality for choosing variables

```
#all columns between
myBatting %>%
  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
```

`select()` - subset columns

- Great functionality for choosing variables

```
#all columns containing
myBatting %>%
  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
```

`select()` - subset columns

- Great functionality for choosing variables

```
#all columns starting with
myBatting %>%
  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
```

`select()` - subset columns

- Great functionality for choosing variables

```
#multiple selections
myBatting %>%
  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
```

`select()` - subset columns

- Can reorder variables

```
#reorder
myBatting %>%
  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 abercda01     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 allisar01     0   1871     1 CL1    NA       29   137    28    40     4     5
## 4 allisdo01     2   1871     1 WS3    NA       27   133    28    44    10     2
## 5 ansonca01     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>
```

`rename()` - rename variables

```
#rename our previous
myBatting %>%
  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

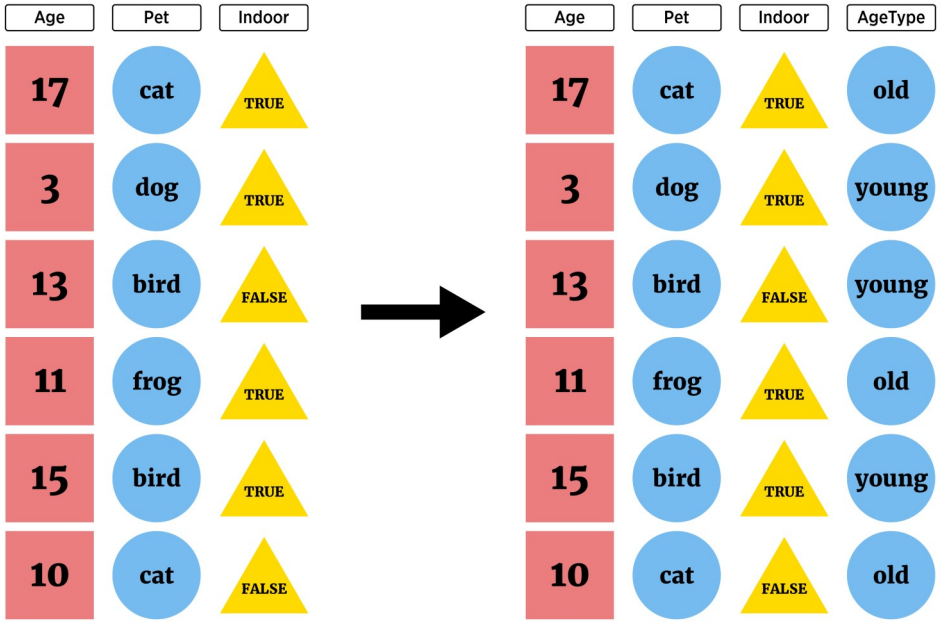
- Basic commands
  - `as_tibble()` - convert data frame to one with better printing
  - `filter()` - subset **rows**
  - `arrange()` - reorder **rows**
  - `select()` - subset **columns**
- Many `joins` to combine tibbles too! (Similar to SQL)

Quick Examples

- Go to the course files page and try Exercise 6 - dplyr

Data manipulation idea

- 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)
```

```
## '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       : chr  "a" "a" "a" "a" ...
```

Creating New Variables

Better method - use `dplyr`

- `mutate()` - add newly created **column(s)** to current data frame (doesn't overwrite the data frame)
- `transmute()` - create new data frame with created variable(s) only
- Syntax:

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

Creating New Variables

- Consider a data set on movie ratings

```
library(fivethirtyeight)
fandango
```

```
## # A tibble: 146 x 23
##   film     year rottentomatoes rottentomatoes_~ metacritic metacritic_user  imdb
##   <chr>   <dbl>          <int>            <int>      <int>           <dbl> <dbl>
## 1 Avenge~  2015             74               86         66             7.1   7.8
## 2 Cinder~  2015             85               80         67             7.5   7.1
## 3 Ant-Man  2015             80               90         64             8.1   7.8
## 4 Do You~  2015             18               84         22             4.7   5.4
## 5 Hot Tu~  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>
```

`mutate()` - create new column(s)

```
##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 Avenge~  2015             74               86         66             7.1   7.8
## 2 Cinder~  2015             85               80         67             7.5   7.1
## 3 Ant-Man  2015             80               90         64             8.1   7.8
## 4 Do You~  2015             18               84         22             4.7   5.4
## 5 Hot Tu~  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>
```

`mutate()` - create new column(s)

```
#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_us~ metacritic
##   <chr>              <dbl>     <dbl>          <int>               <int>      <int>
## 1 Avengers: Age of~  2015      80               74                  86         66
## 2 Cinderella         2015      82.5             85                  80         67
## 3 Ant-Man            2015      85               80                  90         64
## 4 Do You Believe?    2015      51               18                  84         22
## 5 Hot Tub Time Mac~  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>
```

`mutate()` - create new column(s)

- 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 Avengers: ~  2015      80      3.42             74               86         66
## 2 Cinderella   2015      82.5    3.55             85               80         67
## 3 Ant-Man      2015      85      3.62             80               90         64
## 4 Do You Bel~  2015      51      1.72             18               84         22
## 5 Hot Tub Ti~  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

`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

```
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

`mutate` and `transmute` can use any '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
```

Conditional Execution

- 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

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

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

```
## # 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
```

Conditional Execution

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)

Conditional Execution

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)

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
```

Conditional Execution

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)

Could try to `loop` through observations

Instead, use `if_else()` which works on an entire vector

`if_else()` - conditional execution

`if_else()` syntax:

- `if_else(condition, true, false)`
- `condition` is a vector of TRUE/FALSE
- `true` is what to do when TRUE occurs
- `false` is what to do when FALSE occurs
- A vector is then returned

if_else() with mutate()

```
myAirquality <- myAirquality %>%
  mutate(Status = if_else(Wind >= 15, "HighWind",
                          if_else(Wind >= 10, "Windy",
                                  if_else(Wind >= 6, "LightWind", "Calm"))))
myAirquality
```

```
## # 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
```

Creating New Variables Recap!

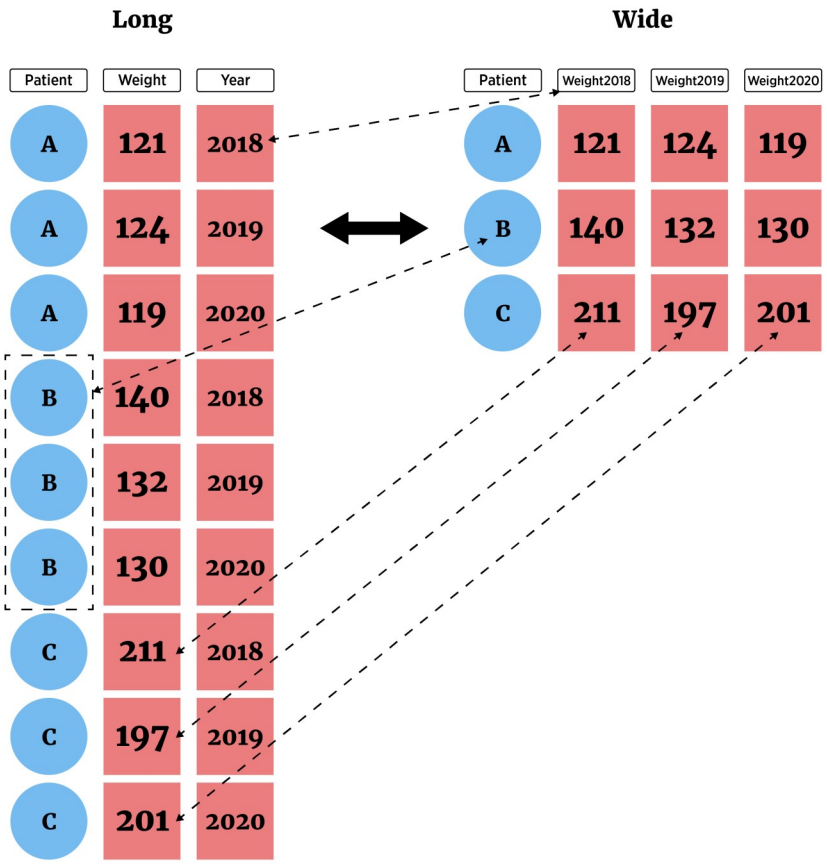
- `mutate()` - add newly created **column(s)** to current data frame
- `transmute()` - create new data frame with created variable(s)
  - Use `if_else()` to do conditional creation
  - Note: `cut()` can be used to categorize a numeric variable!

Quick Examples

- Go to the course files page and try Exercise 7 - Creating Variables

Reshaping Data

Long vs Wide format data



Reshaping Data

`tidyr` package

Easily allows for two very important actions

- `pivot_longer()` - lengthens data by increasing the number of rows and decreasing the number of columns
    - Most important as analysis methods often prefer this form
- `pivot_wider()` - widens data by increasing the number of columns and decreasing the number of rows

`tidyr` Package

- Data in 'Wide' form

```
tempsData <- read_table2(file = "https://www4.stat.ncsu.edu/~online/datasets/cityTemps.txt")
tempsData
```

```
## # A tibble: 6 x 8
##   city        sun   mon   tue   wed   thr   fri   sat
##   <chr>     <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 atlanta      81    87    83    79    88    91    94
## 2 baltimore    73    75    70    78    73    75    79
## 3 charlotte    82    80    75    82    83    88    93
## 4 denver       72    71    67    68    72    71    58
## 5 ellington    51    42    47    52    55    56    59
## 6 frankfort    70    70    72    70    74    74    79
```

Reshaping Data

```
## # A tibble: 6 x 8
##   city        sun   mon   tue   wed   thr   fri   sat
##   <chr>     <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 atlanta      81    87    83    79    88    91    94
## 2 baltimore    73    75    70    78    73    75    79
## 3 charlotte    82    80    75    82    83    88    93
## 4 denver       72    71    67    68    72    71    58
## 5 ellington    51    42    47    52    55    56    59
## 6 frankfort    70    70    72    70    74    74    79
```

- Switch to 'Long' form with `pivot_longer()`
  - `cols` = columns to pivot to longer format (`cols = 2:8`)
  - `names_to` = new name(s) for columns created (`names_to = "day"`)
  - `values_to` = new name(s) for data values (`values_to = "temp"`)

Reshaping Data

- Switch to 'Long' form with `pivot_longer()`
  - `cols` = columns to pivot to longer format (`cols = 2:8`)
  - `names_to` = new name(s) for columns created (`names_to = "day"`)
  - `values_to` = new name(s) for data values (`values_to = "temp"`)

```
tempsData %>% pivot_longer(cols = 2:8, names_to = "day", values_to = "temp")
```

```
## # A tibble: 42 x 3
##   city    day    temp
##   <chr>   <chr> <dbl>
## 1 atlanta sun      81
## 2 atlanta mon      87
## 3 atlanta tue      83
## 4 atlanta wed      79
## 5 atlanta thr      88
## # ... with 37 more rows
```

Reshaping Data

- Switch to 'Long' form with `pivot_longer()`
- Can provide columns in many ways!

```
newTempsData <- tempsData %>%
  pivot_longer(cols = sun:sat, names_to = "day", values_to = "temp")
newTempsData
```

```
## # A tibble: 42 x 3
##   city    day    temp
##   <chr>   <chr> <dbl>
## 1 atlanta sun      81
## 2 atlanta mon      87
## 3 atlanta tue      83
## 4 atlanta wed      79
## 5 atlanta thr      88
## # ... with 37 more rows
```

Reshaping Data

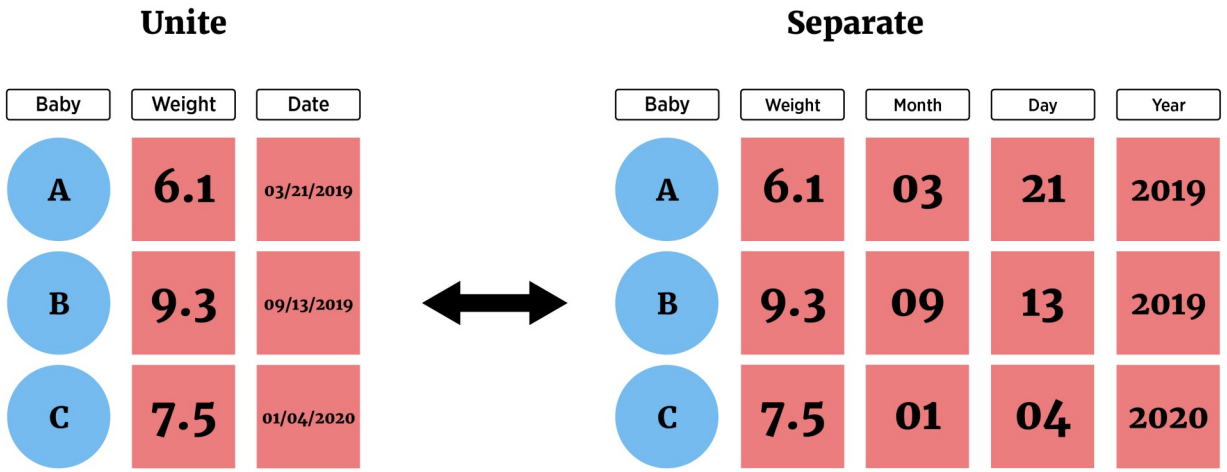
- Switch to 'Wide' form with `pivot_wider()`
  - `names_from` = column(s) to get the names used in the output columns (`names_from = "day"`)
  - `values_from` = column(s) to get the cell values from (`values_from = "temp"`)

```
newTempsData %>%
  pivot_wider(names_from = "day", values_from = "temp")
```

```
## # A tibble: 6 x 8
##   city        sun   mon   tue   wed   thr   fri   sat
##   <chr>     <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 atlanta      81    87    83    79    88    91    94
## 2 baltimore    73    75    70    78    73    75    79
## 3 charlotte    82    80    75    82    83    88    93
## 4 denver       72    71    67    68    72    71    58
## 5 ellington    51    42    47    52    55    56    59
## 6 frankfort    70    70    72    70    74    74    79
```

Reshaping Data

- Separate a column (or combine two columns) using `separate()` and `unite()`



Reshaping Data

- Separate a column (or combine two columns) using `separate()` and `unite()`
- Consider data set on air pollution in Chicago

```
chicagoData <- read_csv("https://www4.stat.ncsu.edu/~online/datasets/Chicago.csv")
chicagoData
```

```
## # A tibble: 1,461 x 11
##       X city  date     death  temp dewpoint  pm10    o3  time season  year
##   <dbl> <chr> <chr>    <dbl> <dbl>    <dbl> <dbl> <dbl> <dbl> <chr>  <dbl>
## 1  3654 chic  1/1/1997   137  36       37.5  13.1  5.66  3654 winter  1997
## 2  3655 chic  1/2/1997   123  45       47.2  41.9  5.53  3655 winter  1997
## 3  3656 chic  1/3/1997   127  40       38    27.0  6.29  3656 winter  1997
## 4  3657 chic  1/4/1997   146  51.5     45.5  25.1  7.54  3657 winter  1997
## 5  3658 chic  1/5/1997   102  27       11.2  15.3 20.8   3658 winter  1997
## # ... with 1,456 more rows
```

separate

- Can split columns with `separate`:

```
chicagoData %>%
  separate(date, c("Month", "Day", "Year"), sep = "/", convert = TRUE, remove = FALSE)
```

```
## # A tibble: 1,461 x 14
##       X city  date     Month   Day  Year death  temp dewpoint  pm10    o3  time
##   <dbl> <chr> <chr>    <int> <int> <int> <dbl> <dbl>    <dbl> <dbl> <dbl> <dbl>
## 1  3654 chic  1/1/1997     1     1  1997   137  36       37.5  13.1  5.66  3654
## 2  3655 chic  1/2/1997     1     2  1997   123  45       47.2  41.9  5.53  3655
## 3  3656 chic  1/3/1997     1     3  1997   127  40       38    27.0  6.29  3656
## 4  3657 chic  1/4/1997     1     4  1997   146  51.5     45.5  25.1  7.54  3657
## 5  3658 chic  1/5/1997     1     5  1997   102  27       11.2  15.3 20.8   3658
## # ... with 1,456 more rows, and 2 more variables: season <chr>, year <dbl>
```

unite

- Can combine two columns with `unite`:

```
chicagoData %>%
  separate(date, c("Month", "Day", "Year"), sep = "/", convert = TRUE, remove = FALSE) %>%
  unite(MonthDay, Month, Day, sep = "-")
```

```
## # A tibble: 1,461 x 13
##       X city  date  MonthDay  Year death  temp dewpoint  pm10    o3  time season
##   <dbl> <chr> <chr> <chr>    <int> <dbl> <dbl>    <dbl> <dbl> <dbl> <dbl> <chr>
## 1  3654 chic  1/1/~ 1-1       1997   137  36       37.5  13.1  5.66  3654 winter
## 2  3655 chic  1/2/~ 1-2       1997   123  45       47.2  41.9  5.53  3655 winter
## 3  3656 chic  1/3/~ 1-3       1997   127  40       38    27.0  6.29  3656 winter
## 4  3657 chic  1/4/~ 1-4       1997   146  51.5     45.5  25.1  7.54  3657 winter
## 5  3658 chic  1/5/~ 1-5       1997   102  27       11.2  15.3 20.8   3658 winter
## # ... with 1,456 more rows, and 1 more variable: year <dbl>
```

Recap!

- Data manipulation idea
- Documenting with Markdown
- Logical statements
- `dplyr`, `tidyr`, and creating new variables

Quick Examples

- Go to the course files page and try Exercise 8 - tidyr