dplyr (and a bit of checking in)

Daniel Anderson

Tidyverse: Providing a grammar for...

- Graphics (*ggplot*)
- Data manipulations (*dplyr*, *tidyr*)
- Ever expanding specialized topics (not quite modeling yet, although there is some)

dplyr: A grammar for data wrangling

Take a guess - what do you think the following do? Discuss with your neighbor.

- `select()`
- `filter()`
- `mutate()`
- `arrange()`
- `summarize()`
- `group_by()`

dplyr: A grammar for data wrangling

- `select()`: A subset of columns
- `filter()`: A subset of rows
- `mutate()`: Add or modify existing columns
- `arrange()`: Rows in a ascending or descending order
- `summarize()`: A variable according to other functions (e.g., `mean()`, `sd()`). Often used in conjunction with `group_by()`

janitor: Cleaning up common dirt

Little easier (minus maybe the first one). Guess what these do?

- `clean_names()`
- `remove_empty_rows()`
- `remove_empty_cols()`
- `excel_numeric_to_date()`
- `get_dupes()`
- `tabyl()`
- `crosstabs()`

janitor: Cleaning up common dirt

Little easier (minus maybe the first one). Guess what these do?

- `clean_names()`: Styles column names with snake_case
- `remove_empty_rows()`: Removes empty rows
- `remove_empty_cols()`: Removes empty columns
- `excel_numeric_to_date()`: Changes numeric dates imported from Excel to actual dates
- `get_dupes()`: Provides duplicates for any variable(s) provided
- `tabyl()`: Frequncies. Similar to `dplyr::count()`
- `crosstabs()`: Cross-tabulation table.

There are also `adorn_*()` to spruce up the last two.

Examples from janitor README

Import the data (first start a new project, and put the data in a data folder, then..)

```
library(rio)
dirty <- import("./data/dirty_data.xlsx")
dirty
```

```
##       First Name Last Name Employee Status     Subject Hire Date % Allocated
## 1          Jason    Bourne         Teacher          PE     39690         0.75
## 2          Jason    Bourne         Teacher    Drafting     39690         0.25
## 3         Alicia      Keys         Teacher       Music     37118         1.00
## 4            Ada  Lovelace         Teacher        <NA>     27515         1.00
## 5          Desus      Nice  Administration        Dean     41431         1.00
## 6  Chien-Shiung        Wu         Teacher     Physics     11037         0.50
## 7  Chien-Shiung        Wu         Teacher   Chemistry     11037         0.50
## 8           <NA>      <NA>            <NA>        <NA>        NA           NA
## 9          James     Joyce         Teacher     English     32994         0.50
## 10          Hedy    Lamarr         Teacher     Science     27919         0.50
## 11        Carlos    Boozer           Coach  Basketball     42221           NA
## 12         Young    Boozer           Coach        <NA>     34700           NA
## 13       Micheal    Larsen         Teacher     English     40071         0.80
##    Full time? do not edit! ---> Certification Certification__1
## 1          Yes                NA   Physical ed          Theater
```

clean_names()

- Function I use the most. I *really* recommend getting in the habit of sticking with a consistent style, and if that style is snake_case, then this is a great helper function.

```
library(janitor)
clean_names(dirty)
```

```
##      first_name last_name employee_status    subject hire_date
## 1         Jason    Bourne         Teacher         PE     39690
## 2         Jason    Bourne         Teacher   Drafting     39690
## 3        Alicia      Keys         Teacher      Music     37118
## 4           Ada  Lovelace         Teacher       <NA>     27515
## 5         Desus      Nice  Administration       Dean     41431
## 6  Chien-Shiung        Wu         Teacher    Physics     11037
## 7  Chien-Shiung        Wu         Teacher  Chemistry     11037
## 8          <NA>      <NA>            <NA>       <NA>        NA
## 9         James     Joyce         Teacher    English     32994
## 10         Hedy    Lamarr         Teacher    Science     27919
## 11       Carlos    Boozer           Coach Basketball     42221
## 12        Young    Boozer           Coach       <NA>     34700
## 13      Micheal    Larsen         Teacher    English     40071
##    percent_allocated full_time do_not_edit certification certification_1
```

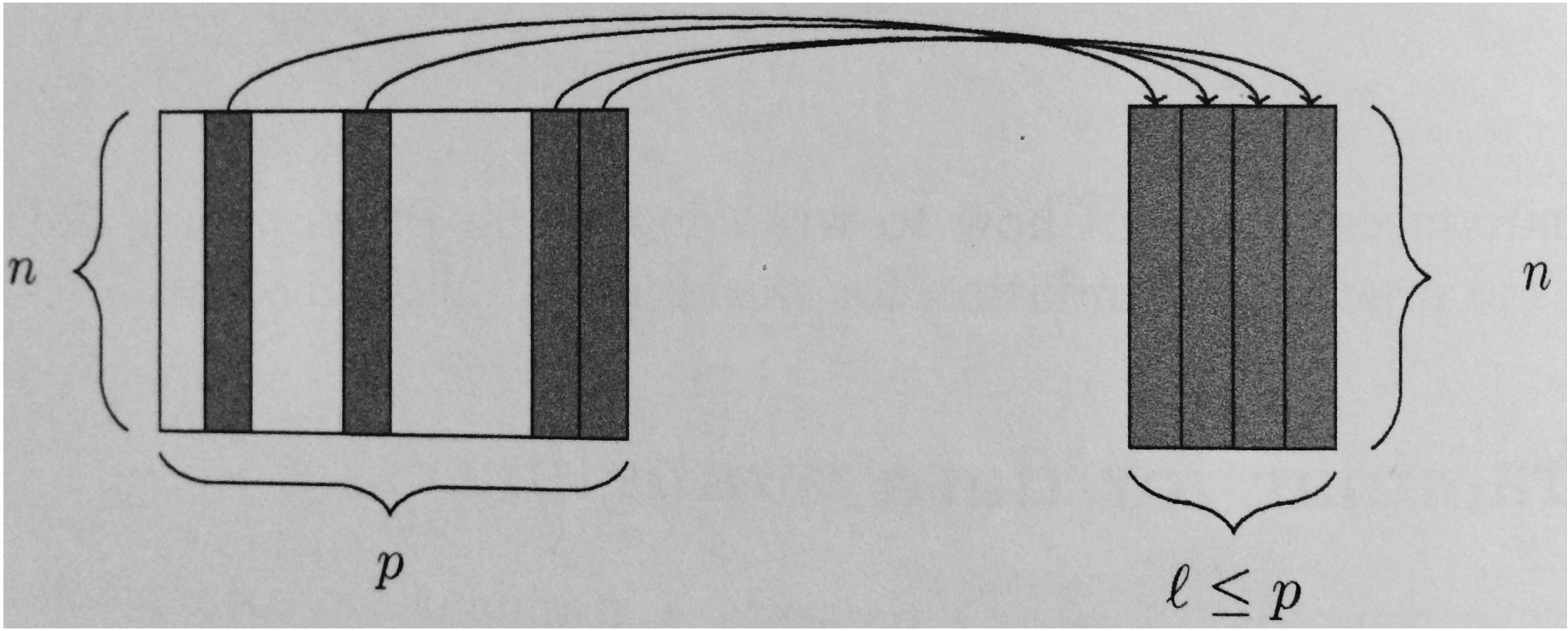
- Because the janitor functions are relatively straightforward, we'll move on to dplyr. But we'll be using *janitor* fairly frequently throughout the term.
- Good habit to get into:
    - import data
    - clean names
    - remove empty rows
    - remove empty columns

dplyr

Arguments

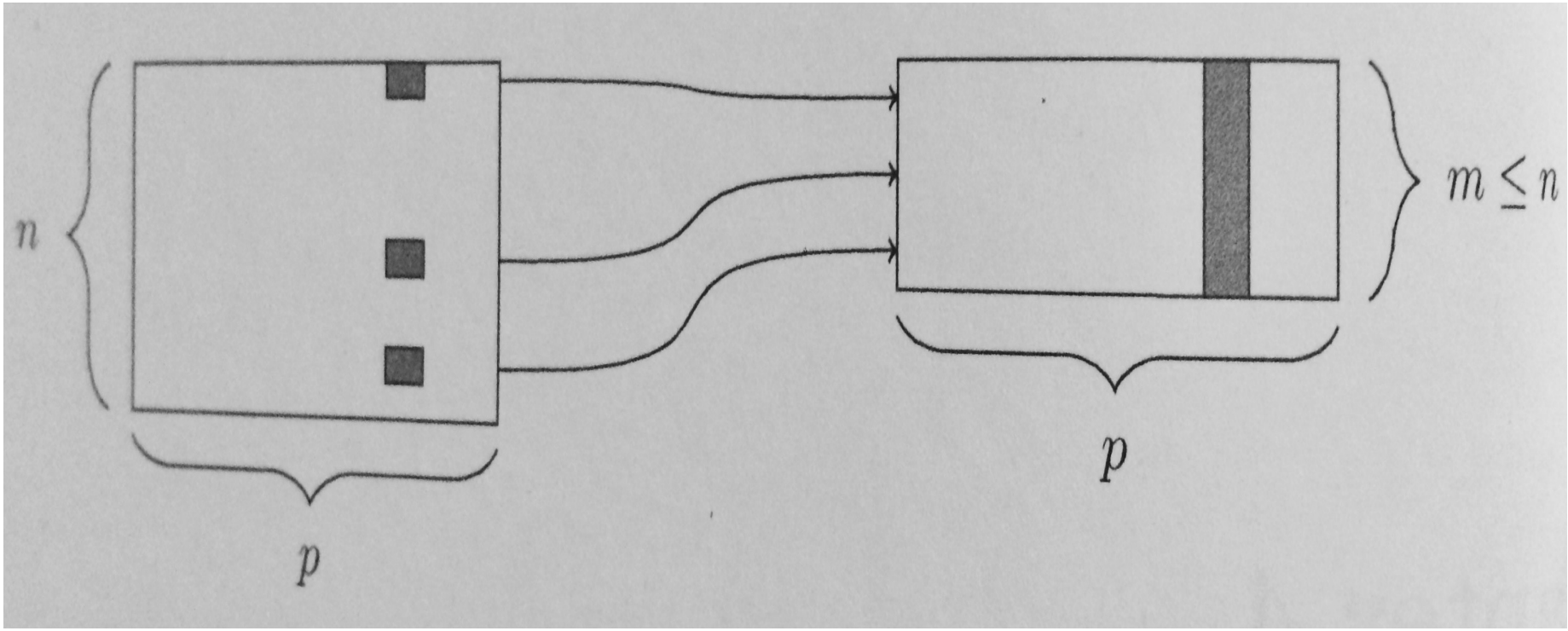
- *dplyr* always takes a data frame as the first argument.
- Subsequent argument tell *dplyr* what to do with the data frame.
- Returns the modified data frame

`select()`



(Figure from Baumer, Kaplan, & Horton, 2017)

`filter()`



(Figure from Baumer, Kaplan, & Horton, 2017)

Examples (follow along!)

We'll start with the presidential dataset from the *mdsr* package.

```
install.packages("mdsr")
library(mdsr)
presidential
```

NAME	START	END	PARTY
Eisenhower	1953-01-20	1961-01-20	Republican
Kennedy	1961-01-20	1963-11-22	Democratic
Johnson	1963-11-22	1969-01-20	Democratic
Nixon	1969-01-20	1974-08-09	Republican
Ford	1974-08-09	1977-01-20	Republican
Carter	1977-01-20	1981-01-20	Democratic

Select president name and party

```
select(presidential, name, party)
```

```
## # A tibble: 11 x 2
##          name      party
##         <chr>      <chr>
##  1 Eisenhower Republican
##  2    Kennedy Democratic
##  3    Johnson Democratic
##  4      Nixon Republican
##  5       Ford Republican
##  6     Carter Democratic
##  7     Reagan Republican
##  8       Bush Republican
##  9    Clinton Democratic
## 10       Bush Republican
## 11      Obama Democratic
```

Use negation

```
select(presidential, -start)
```

```
## # A tibble: 11 x 3
##          name        end      party
##         <chr>     <date>      <chr>
##  1 Eisenhower 1961-01-20 Republican
##  2    Kennedy 1963-11-22 Democratic
##  3    Johnson 1969-01-20 Democratic
##  4      Nixon 1974-08-09 Republican
##  5       Ford 1977-01-20 Republican
##  6     Carter 1981-01-20 Democratic
##  7     Reagan 1989-01-20 Republican
##  8       Bush 1993-01-20 Republican
##  9    Clinton 2001-01-20 Democratic
## 10       Bush 2009-01-20 Republican
## 11      Obama 2017-01-20 Democratic
```

Use indexing

```
select(presidential, 1:3)
```

```
## # A tibble: 11 x 3
##          name      start        end
##         <chr>     <date>     <date>
##  1 Eisenhower 1953-01-20 1961-01-20
##  2    Kennedy 1961-01-20 1963-11-22
##  3    Johnson 1963-11-22 1969-01-20
##  4      Nixon 1969-01-20 1974-08-09
##  5       Ford 1974-08-09 1977-01-20
##  6     Carter 1977-01-20 1981-01-20
##  7     Reagan 1981-01-20 1989-01-20
##  8       Bush 1989-01-20 1993-01-20
##  9    Clinton 1993-01-20 2001-01-20
## 10       Bush 2001-01-20 2009-01-20
## 11      Obama 2009-01-20 2017-01-20
```

`select()` helper funs

- `starts_with()`
- `ends_with()`
- `contains()`
- `matches()`
- `num_range()`

Project Reads data

```
reads <- import("./data/Project_Reads_Scores.csv")
reads <- clean_names(reads)
reads
```

```
##                 test_year test_type    test_site             student_id
## 1  06/01/2016 12:00:00 AM  YEAR END       VIRDEN               Virden 1
## 2  06/01/2016 12:00:00 AM  YEAR END       VIRDEN               Virden 2
## 3  06/01/2016 12:00:00 AM  YEAR END       VIRDEN               Virden 3
## 4  06/01/2016 12:00:00 AM  YEAR END       VIRDEN               Virden 4
## 5  06/01/2016 12:00:00 AM  YEAR END       VIRDEN               Virden 5
## 6  06/01/2016 12:00:00 AM  YEAR END       VIRDEN               Virden 6
## 7  06/01/2016 12:00:00 AM  YEAR END       VIRDEN               Virden 7
## 8  06/01/2016 12:00:00 AM  YEAR END       VIRDEN               Virden 8
## 9  06/01/2016 12:00:00 AM  YEAR END       VIRDEN               Virden 9
## 10 06/01/2016 12:00:00 AM  YEAR END       VIRDEN              Virden 10
## 11 06/01/2016 12:00:00 AM  YEAR END       VIRDEN              Virden 11
## 12 06/01/2016 12:00:00 AM  YEAR END       VIRDEN              Virden 12
## 13 06/01/2016 12:00:00 AM  YEAR END       VIRDEN              Virden 13
## 14 06/01/2016 12:00:00 AM  YEAR END       VIRDEN              Virden 14
## 15 06/01/2016 12:00:00 AM  YEAR END       VIRDEN              Virden 15
## 16 06/01/2016 12:00:00 AM  YEAR END        JONES                Jones 1
```

starts_with

```
select(reads, starts_with("test"))
```

```
##                 test_year test_type    test_site
## 1  06/01/2016 12:00:00 AM  YEAR END       VIRDEN
## 2  06/01/2016 12:00:00 AM  YEAR END       VIRDEN
## 3  06/01/2016 12:00:00 AM  YEAR END       VIRDEN
## 4  06/01/2016 12:00:00 AM  YEAR END       VIRDEN
## 5  06/01/2016 12:00:00 AM  YEAR END       VIRDEN
## 6  06/01/2016 12:00:00 AM  YEAR END       VIRDEN
## 7  06/01/2016 12:00:00 AM  YEAR END       VIRDEN
## 8  06/01/2016 12:00:00 AM  YEAR END       VIRDEN
## 9  06/01/2016 12:00:00 AM  YEAR END       VIRDEN
## 10 06/01/2016 12:00:00 AM  YEAR END       VIRDEN
## 11 06/01/2016 12:00:00 AM  YEAR END       VIRDEN
## 12 06/01/2016 12:00:00 AM  YEAR END       VIRDEN
## 13 06/01/2016 12:00:00 AM  YEAR END       VIRDEN
## 14 06/01/2016 12:00:00 AM  YEAR END       VIRDEN
## 15 06/01/2016 12:00:00 AM  YEAR END       VIRDEN
## 16 06/01/2016 12:00:00 AM  YEAR END        JONES
## 17 06/01/2016 12:00:00 AM  YEAR END        JONES
## 18 06/01/2016 12:00:00 AM  YEAR END        JONES
```

ends_with

```
select(reads, ends_with("score"))
```

```
##    pre_test_score post_test_score unit_1_score unit_2_score unit_3_score
## 1              43              92            3            4            6
## 2              46             104            5            5            6
## 3              39              75            4            4            6
## 4              35             115            4            4            6
## 5              46              85            2            5            6
## 6              35              91            5            5            7
## 7              40              96            5            5            6
## 8              39              74            4            5            5
## 9              40              90            6            4            5
## 10             45              86            4            5            5
## 11             32              91            5            5            6
## 12             31             102            4            5            7
## 13             33              86            4            4            7
## 14             41              99            5            4            7
## 15             35             101            3            5            8
## 16             36             103            4            4            5
## 17             37              99            2            3            5
## 18             54              95            2            4            6
```

contains

```
select(reads, contains("test"))
```

```
##                 test_year test_type    test_site pre_test_score
## 1  06/01/2016 12:00:00 AM  YEAR END       VIRDEN             43
## 2  06/01/2016 12:00:00 AM  YEAR END       VIRDEN             46
## 3  06/01/2016 12:00:00 AM  YEAR END       VIRDEN             39
## 4  06/01/2016 12:00:00 AM  YEAR END       VIRDEN             35
## 5  06/01/2016 12:00:00 AM  YEAR END       VIRDEN             46
## 6  06/01/2016 12:00:00 AM  YEAR END       VIRDEN             35
## 7  06/01/2016 12:00:00 AM  YEAR END       VIRDEN             40
## 8  06/01/2016 12:00:00 AM  YEAR END       VIRDEN             39
## 9  06/01/2016 12:00:00 AM  YEAR END       VIRDEN             40
## 10 06/01/2016 12:00:00 AM  YEAR END       VIRDEN             45
## 11 06/01/2016 12:00:00 AM  YEAR END       VIRDEN             32
## 12 06/01/2016 12:00:00 AM  YEAR END       VIRDEN             31
## 13 06/01/2016 12:00:00 AM  YEAR END       VIRDEN             33
## 14 06/01/2016 12:00:00 AM  YEAR END       VIRDEN             41
## 15 06/01/2016 12:00:00 AM  YEAR END       VIRDEN             35
## 16 06/01/2016 12:00:00 AM  YEAR END        JONES             36
## 17 06/01/2016 12:00:00 AM  YEAR END        JONES             37
## 18 06/01/2016 12:00:00 AM  YEAR END        JONES             54
```

matches

Matches a regular expression (here, contains a digit))

```
select(reads, matches("\\d"))
```

```
##    unit_1_score unit_1_percent unit_2_score unit_2_percent unit_3_score
## 1             3            12%            4            16%            6
## 2             5            20%            5            20%            6
## 3             4            16%            4            16%            6
## 4             4            16%            4            16%            6
## 5             2             8%            5            20%            6
## 6             5            20%            5            20%            7
## 7             5            20%            5            20%            6
## 8             4            16%            5            20%            5
## 9             6            24%            4            16%            5
## 10            4            16%            5            20%            5
## 11            5            20%            5            20%            6
## 12            4            16%            5            20%            7
## 13            4            16%            4            16%            7
## 14            5            20%            4            16%            7
## 15            3            12%            5            20%            8
## 16            4            16%            4            16%            5
```

num_range

Only works if the number is at the end of the column. So, first drop "_score" from each of the score columns, then select units 2-4 and 7.

```
names(reads)[grep("score", names(reads))] <- gsub("_score", "",
    names(reads)[grep("score", names(reads))])

select(reads, num_range("unit_", c(2:4, 7)))
```

```
##    unit_2 unit_3 unit_4 unit_7
## 1       4      6      8     22
## 2       5      6      8     20
## 3       4      6      9     19
## 4       4      6     10     18
## 5       5      6      7     19
## 6       5      7      7     23
## 7       5      6      7     21
## 8       5      5      8     16
## 9       4      5      8     18
## 10      5      5      7     21
## 11      5      6      8     22
## 12      5      7      8     17
```

Last thing on `select`

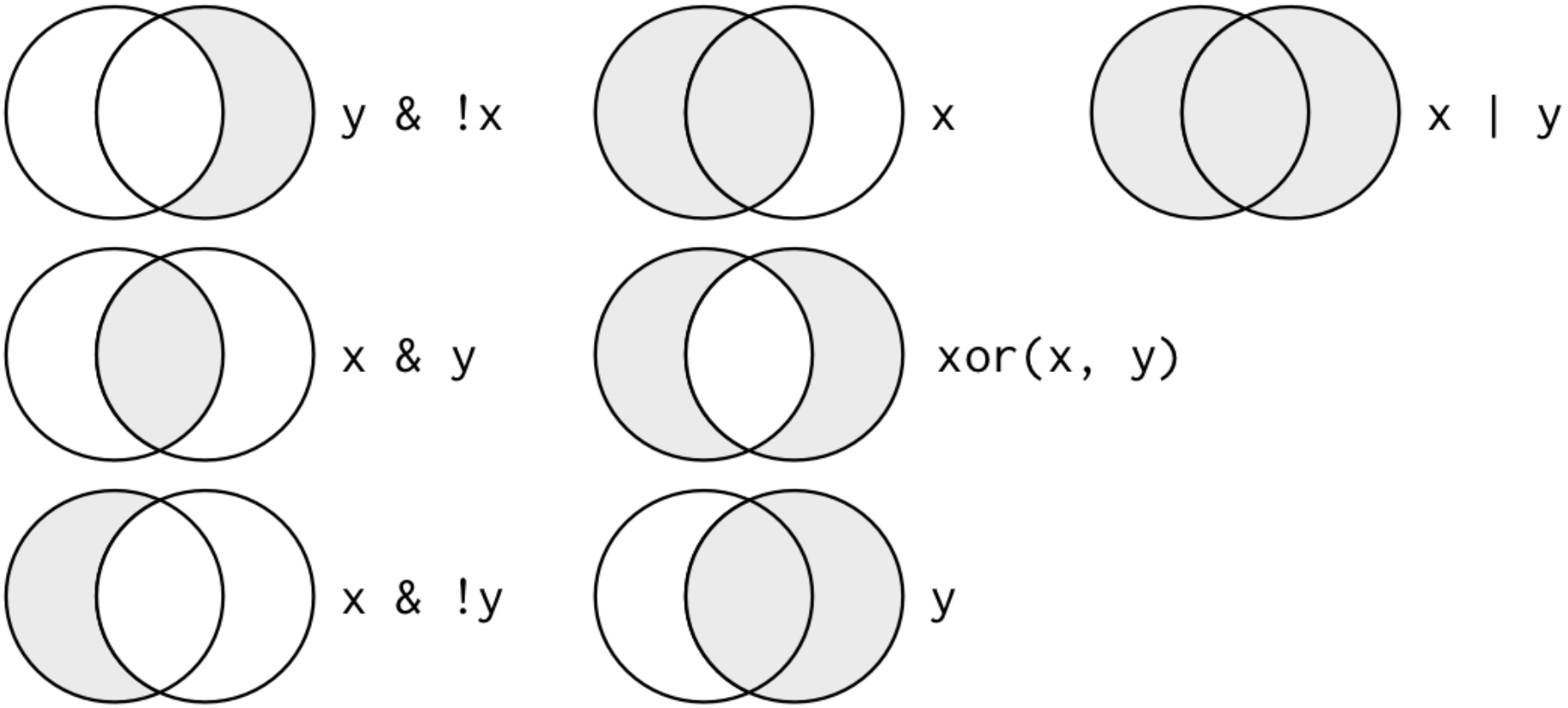
You can mix types and helper funs. You can also use select to rearrange your columns.

```
select(reads, student_id, 1, starts_with("total"))
```

```
##                student_id              test_year total
## 1               Virden 1 06/01/2016 12:00:00 AM   207
## 2               Virden 2 06/01/2016 12:00:00 AM   224
## 3               Virden 3 06/01/2016 12:00:00 AM   193
## 4               Virden 4 06/01/2016 12:00:00 AM   223
## 5               Virden 5 06/01/2016 12:00:00 AM   198
## 6               Virden 6 06/01/2016 12:00:00 AM   210
## 7               Virden 7 06/01/2016 12:00:00 AM   211
## 8               Virden 8 06/01/2016 12:00:00 AM   180
## 9               Virden 9 06/01/2016 12:00:00 AM   206
## 10             Virden 10 06/01/2016 12:00:00 AM   206
## 11             Virden 11 06/01/2016 12:00:00 AM   199
## 12             Virden 12 06/01/2016 12:00:00 AM   202
## 13             Virden 13 06/01/2016 12:00:00 AM   189
## 14             Virden 14 06/01/2016 12:00:00 AM   223
## 15             Virden 15 06/01/2016 12:00:00 AM   211
## 16               Jones 1 06/01/2016 12:00:00 AM   209
```

`filter()` boolean logic

Left circle == x, right circle == y



(Figure from Wickham & Grolemund, 2017)

filter for democrats

```
filter(presidential, party == "Democratic")
```

```
## # A tibble: 5 x 4
##      name      start        end      party
##     <chr>     <date>     <date>      <chr>
## 1 Kennedy 1961-01-20 1963-11-22 Democratic
## 2 Johnson 1963-11-22 1969-01-20 Democratic
## 3  Carter 1977-01-20 1981-01-20 Democratic
## 4 Clinton 1993-01-20 2001-01-20 Democratic
## 5   Obama 2009-01-20 2017-01-20 Democratic
```

filter for dems between 1970 and 2000

```
filter(presidential, party == "Democratic" &
                     (start > as.Date("1970-01-01") &
                     start < as.Date("2000-01-01")))
```

```
## # A tibble: 2 x 4
##      name      start        end      party
##     <chr>     <date>     <date>      <chr>
## 1  Carter 1977-01-20 1981-01-20 Democratic
## 2 Clinton 1993-01-20 2001-01-20 Democratic
```

Multiple args to filter

- You can supply multiple arguments to filter and they will be stapled together with `&`.
- The below is equivalent to the code on the previous slide

```
filter(presidential, party == "Democratic",
                     start > as.Date("1970-01-01"),
                     start < as.Date("2000-01-01"))
```

```
## # A tibble: 2 x 4
##      name      start        end      party
##     <chr>     <date>     <date>      <chr>
## 1  Carter 1977-01-20 1981-01-20 Democratic
## 2 Clinton 1993-01-20 2001-01-20 Democratic
```

Chaining arguments

- What if we wanted to select and filter a dataset?
- Select name and party of presidents who began their term after 2000

Two step method

```
after_2000 <- filter(presidential, start > as.Date("2000-01-01"))
select(after_2000, name, party)
```

```
## # A tibble: 2 x 2
##    name      party
##   <chr>      <chr>
## 1  Bush Republican
## 2 Obama Democratic
```

Chaining arguments

- Alternatively, we could wrap `select` around `filter`

```
select(filter(presidential, start > as.Date("2000-01-01")), name, party)
```

```
## # A tibble: 2 x 2
##    name      party
##   <chr>      <chr>
## 1  Bush Republican
## 2 Obama Democratic
```

Chaining arguments

- Or, we could use another function to help increase the readability of our code: `%>%`
- Called the "pipe" operator and "piping functions"

```
filter(presidential, start > as.Date("2000-01-01")) %>%
  select(name, party)
```

```
## # A tibble: 2 x 2
##    name      party
##   <chr>      <chr>
## 1  Bush Republican
## 2 Obama Democratic
```

Chaining arguments

Generally when using the pipe, the first argument is the dataset, which gets piped through the corresponding functions. So the code on the prior slide would more typically be written

```
presidential %>%
  filter(start > as.Date("2000-01-01")) %>%
  select(name, party)
```

```
## # A tibble: 2 x 2
##    name      party
##   <chr>      <chr>
## 1  Bush Republican
## 2 Obama Democratic
```

Note the indentation and line breaks to help keep things straight.

Your turn

- Install and load the package *Lahman*, which will give you access to the dataset *Teams*

```
install.packages("Lahman")
library(Lahman)
head(Teams)
```

- Using piping, produce a subset of the data that has the following characteristics:
  - Only one team (your choice)
  - data from 1980 to present (or as present as the dataset gets)
  - Includes 5 columns: name, yearID, W, L, R, RA

(The variables above correspond to the team name, the year, wins, losses, runs scored, and runs allowed)

- Make sure you select a team that is currently still around, or it probably won't be interesting.

Baseball data example

(one possible method)

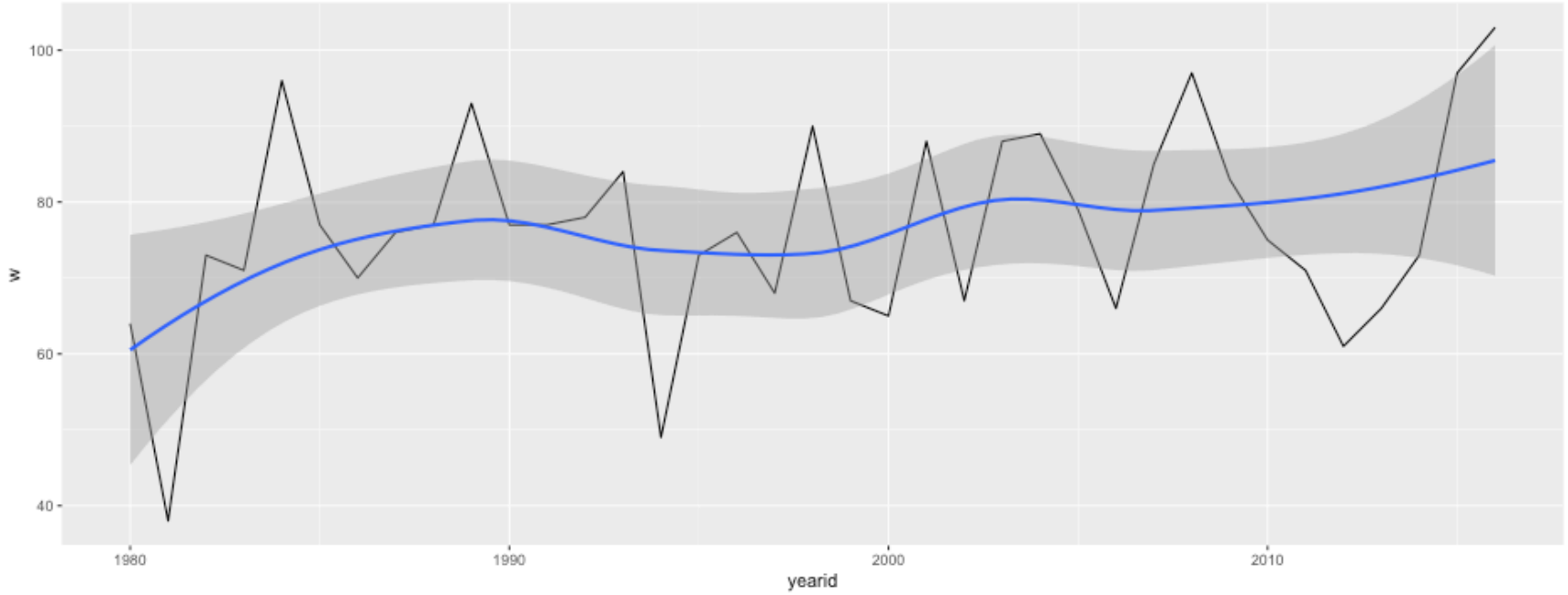
```
teams <- Teams %>%
  clean_names()

cubs <- teams %>%
    filter(name == "Chicago Cubs" & yearid >= 1980) %>%
    select(name, yearid, w, l, r, ra)
cubs
```

```
##            name yearid   w   l   r  ra
## 1  Chicago Cubs   1980  64  98 614 728
## 2  Chicago Cubs   1981  38  65 370 483
## 3  Chicago Cubs   1982  73  89 676 709
## 4  Chicago Cubs   1983  71  91 701 719
## 5  Chicago Cubs   1984  96  65 762 658
## 6  Chicago Cubs   1985  77  84 686 729
## 7  Chicago Cubs   1986  70  90 680 781
## 8  Chicago Cubs   1987  76  85 720 801
## 9  Chicago Cubs   1988  77  85 660 694
## 10 Chicago Cubs   1989  93  69 702 623
## 11 Chicago Cubs   1990  77  85 690 774
```

Quick Aside

Time series plot (where we're heading)

```
library(ggplot2)
ggplot(cubs, aes(yearid, w)) +
  geom_line() +
  geom_smooth()
```



Finishing up with *dplyr*

- Create a new variable

Let's compute the winning percentage for the team you chose over time

$$w_{pct} = \frac{wins}{wins + losses}$$

```
cubs <- cubs %>%
    mutate(w_pct = w / (w + l))

cubs
```

```
##             name yearid   w   l   r  ra     w_pct
## 1  Chicago Cubs   1980  64  98 614 728 0.3950617
## 2  Chicago Cubs   1981  38  65 370 483 0.3689320
## 3  Chicago Cubs   1982  73  89 676 709 0.4506173
## 4  Chicago Cubs   1983  71  91 701 719 0.4382716
## 5  Chicago Cubs   1984  96  65 762 658 0.5962733
## 6  Chicago Cubs   1985  77  84 686 729 0.4782609
## 7  Chicago Cubs   1986  70  90 680 781 0.4375000
## 8  Chicago Cubs   1987  76  85 720 801 0.4720497
## 9  Chicago Cubs   1988  77  85 660 694 0.4753086
```

Order by winning percentage: Least to greatest

```
cubs %>% arrange(w_pct)
```

```
##            name yearid   w   l   r  ra     w_pct
## 1  Chicago Cubs   1981  38  65 370 483 0.3689320
## 2  Chicago Cubs   2012  61 101 613 759 0.3765432
## 3  Chicago Cubs   1980  64  98 614 728 0.3950617
## 4  Chicago Cubs   2000  65  97 764 904 0.4012346
## 5  Chicago Cubs   2006  66  96 716 834 0.4074074
## 6  Chicago Cubs   2013  66  96 602 689 0.4074074
## 7  Chicago Cubs   1999  67  95 747 920 0.4135802
## 8  Chicago Cubs   2002  67  95 706 759 0.4135802
## 9  Chicago Cubs   1997  68  94 687 759 0.4197531
## 10 Chicago Cubs   1994  49  64 500 549 0.4336283
## 11 Chicago Cubs   1986  70  90 680 781 0.4375000
## 12 Chicago Cubs   1983  71  91 701 719 0.4382716
## 13 Chicago Cubs   2011  71  91 654 756 0.4382716
## 14 Chicago Cubs   1982  73  89 676 709 0.4506173
## 15 Chicago Cubs   2014  73  89 614 707 0.4506173
## 16 Chicago Cubs   2010  75  87 685 767 0.4629630
## 17 Chicago Cubs   1996  76  86 772 771 0.4691358
## 18 Chicago Cubs   1987  76  85 720 801 0.4720497
```

Order by winning percentage: greatest to least

```
cubs %>% arrange(desc(w_pct))
```

```
##            name yearid   w   l   r  ra     w_pct
## 1  Chicago Cubs   2016 103  58 808 556 0.6397516
## 2  Chicago Cubs   2008  97  64 855 671 0.6024845
## 3  Chicago Cubs   2015  97  65 689 608 0.5987654
## 4  Chicago Cubs   1984  96  65 762 658 0.5962733
## 5  Chicago Cubs   1989  93  69 702 623 0.5740741
## 6  Chicago Cubs   1998  90  73 831 792 0.5521472
## 7  Chicago Cubs   2004  89  73 789 665 0.5493827
## 8  Chicago Cubs   2001  88  74 777 701 0.5432099
## 9  Chicago Cubs   2003  88  74 724 683 0.5432099
## 10 Chicago Cubs   2007  85  77 752 690 0.5246914
## 11 Chicago Cubs   1993  84  78 738 739 0.5185185
## 12 Chicago Cubs   2009  83  78 707 672 0.5155280
## 13 Chicago Cubs   1995  73  71 693 671 0.5069444
## 14 Chicago Cubs   2005  79  83 703 714 0.4876543
## 15 Chicago Cubs   1992  78  84 593 624 0.4814815
## 16 Chicago Cubs   1991  77  83 695 734 0.4812500
## 17 Chicago Cubs   1985  77  84 686 729 0.4782609
## 18 Chicago Cubs   1988  77  85 660 694 0.4753086
```

summarize

- Compute the mean and standard deviation of winning percentage

```
cubs %>%
    summarize(mean_winning_pct = mean(w_pct),
              sd_winning_pct = sd(w_pct))
```

```
##   mean_winning_pct sd_winning_pct
## 1        0.4811027     0.06864337
```

group_by

- Conduct an operation *by* each level of a grouping factor

Compute average wins for each team

```
teams %>%
    group_by(name) %>%
    summarize(av_wins = mean(w, na.rm = TRUE))
```

```
## # A tibble: 139 x 2
##                      name  av_wins
##                     <chr>    <dbl>
##  1 Altoona Mountain City  6.00000
##  2        Anaheim Angels 83.00000
##  3  Arizona Diamondbacks 79.10526
##  4        Atlanta Braves 81.84314
##  5    Baltimore Canaries 26.00000
##  6   Baltimore Marylands  0.00000
##  7 Baltimore Monumentals 58.00000
##  8     Baltimore Orioles 76.87952
##  9   Baltimore Terrapins 65.50000
## 10      Boston Americans 75.42857
```

Putting it all together

- Move back to full dataset
- Use `group_by` in conjunction with `summary` to
    - compute the average and standard deviation of winning percentage for each team.
- Order by highest winning percentage

```
teams %>%
  mutate(w_pct = w / (w + l)) %>%
  group_by(name) %>%
  summarize(n = n(),
            mean_winning_pct = mean(w_pct, na.rm = TRUE),
            sd_winning_pct = sd(w_pct, na.rm = TRUE)) %>%
  arrange(desc(mean_winning_pct))
```

```
## # A tibble: 139 x 4
##                        name     n mean_winning_pct sd_winning_pct
##                       <chr> <int>            <dbl>          <dbl>
##  1     Boston Red Stockings     5        0.7733718     0.09110570
##  2   Cincinnati Outlaw Reds     1        0.6571429             NA
##  3              Boston Reds     3        0.6163020     0.07903104
##  4         Providence Grays     8        0.6092069     0.08552465
##  5  Chicago White Stockings    17        0.6090423     0.11726548
##  6 Cincinnati Red Stockings     8        0.5888868     0.06395065
##  7          Boston Red Caps     7        0.5790099     0.09764495
##  8         New York Yankees   104        0.5760731     0.06993827
##  9  Brooklyn Ward's Wonders     1        0.5757576             NA
## 10      Philadelphia Whites     3        0.5744543     0.09339407
## # ... with 129 more rows
```

Final notes on *dplyr*

- We'll be using *dplyr* all term long
- Discussion today was fairly superficial
- Discussion today was fast (especially at the end)
- Verbs can help you gain fluency
- There are also conditional and all-inclusive versions of `mutate`, `select`, and `summarize`.
    - For example `mutate_if(is.character, as.numeric)`, `select_if(is.numeric)`, etc. We'll talk about these more starting Thursday.

R Markdown & R Studio Projects

Discussion

- What struggles are you having?
- Have you tried getting a pdf to render?
- How comfortable are you with it at this point?

Pop Quiz (sort of)

Discuss with your neighbor to come up with answers to the following:

- How do you specify a Level 1 header? Level 2? Level 3?
- How would you create a bulleted list that includes indentations (nesting)? What about a numbered list?
- What is a YAML and what are two basic pieces that are generally included in the YAML? Can you think of other things that might be included there?

Reminder - best practices

- Always name your code chunks (you won't end up with a bunch of "untitled_chunk" files on your computer)
    - Don't include spaces in your chunk names
    - Keep code chunk names unique or rendering will fail
- Include lots of small chunks, rather than one big one
    - Helps you diagnose where an error in rendering is occuring, if one occurs
- Render frequently
- Keep code chunks close to relevant text (helps you when you come back to it later) (we'll discuss more chunk options, etc., Week 5)

R Studio Projects

- Struggles?
- Think about working directories.
- Keep everything in the project that you need for the analysis
- Should have a new R Studio project for each project (homework, lab, final project, etc.)
- Ask yourself regularly if you would be able to reproduce the work you've done if you came into it with no history with the project.
    - If yes, good, if no, make changes until the answer is yes

For Next Time

- Homework 1 due
- Homework 2 assigned
- We'll discuss importing data
- Last minute questions?