Introduction to data wrangling 1

dplyr and friends

Daniel Anderson
Week 3, Class 1

UNIVERSITY OF OREGON

Agenda

- Questions
- Basic data wrangling
- Lab

First: Cool stuff share!

- Equations within RMD look beautiful, but can be a pain
  - Must use \LaTeX (see examples here)

First: Cool stuff share!

- Equations within RMD look beautiful, but can be a pain
  - Must use \LaTeX (see examples here)

```
$$ \int_0^{2\pi} \sin x-dx $$
```

First: Cool stuff share!

- Equations within RMD look beautiful, but can be a pain
  - Must use \LaTeX (see examples here)

```
$$ \int_0^{2\pi} \sin x-dx $$
```

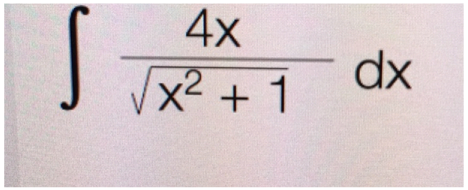
$$\int_0^{2\pi} \sin x - dx$$

{mathpix}

- See https://github.com/jonocarroll/mathpix

Usage

If you have an image you would rather properly encode in LaTeX, for example



then simply calling

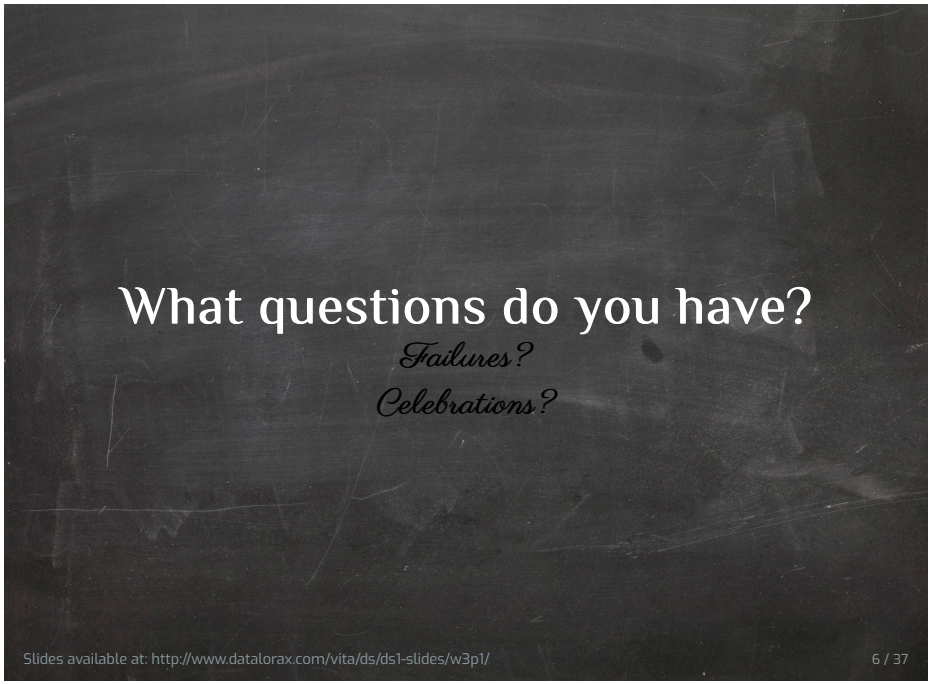
```
mathpix("./integral.jpg")
```

(with the appropriate path to the file) will insert a LaTeX block into your document which will render what the image represents

```
$$
 \int \frac { 4 x } { \sqrt { x ^ { 2 } + 1 } } d x
$$
```


Surface doesn't matter

(demo following)



Providing a grammar for...

- Graphics ({ggplot})
- Data manipulations ({dplyr}, {tidyr})
- Ever expanding specialized topics (some modeling, see {infer})

{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

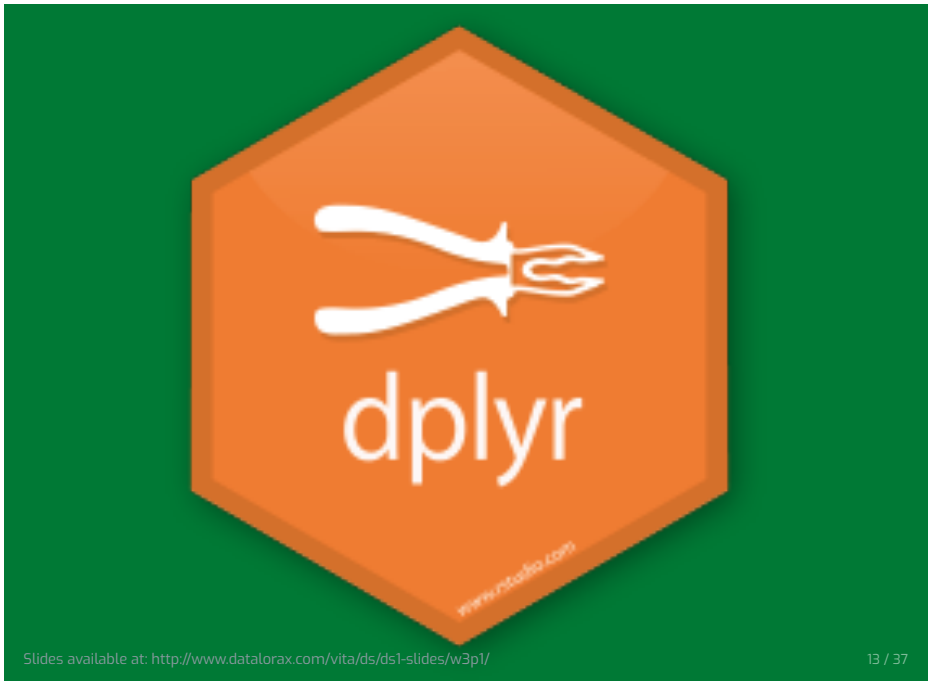
Little easier (minus maybe the first one). Guess what these do?

- `clean_names()`
- `remove_empty_rows()`
- `remove_empty_cols()`

{janitor}: Cleaning up common dirt

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

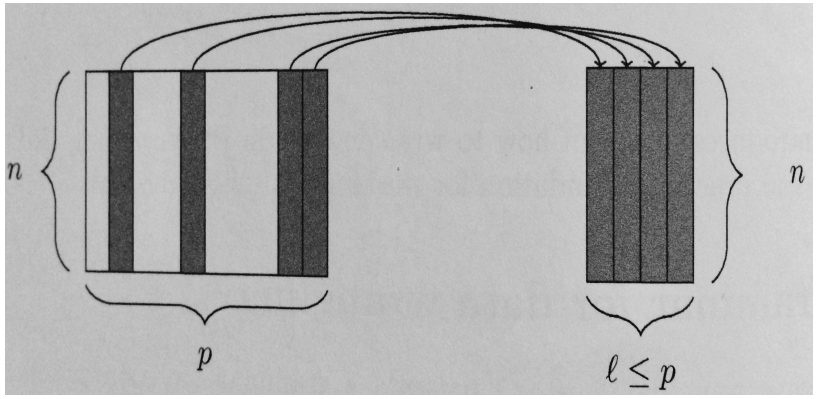
- `clean_names()`: Styles column names with snake_case
- `remove_empty_rows()`: Removes empty rows
- `remove_empty_cols()`: Removes empty columns
- `excel_numeric_to_date()`: Changes numeric dates imported from Excel to actual dates



Arguments

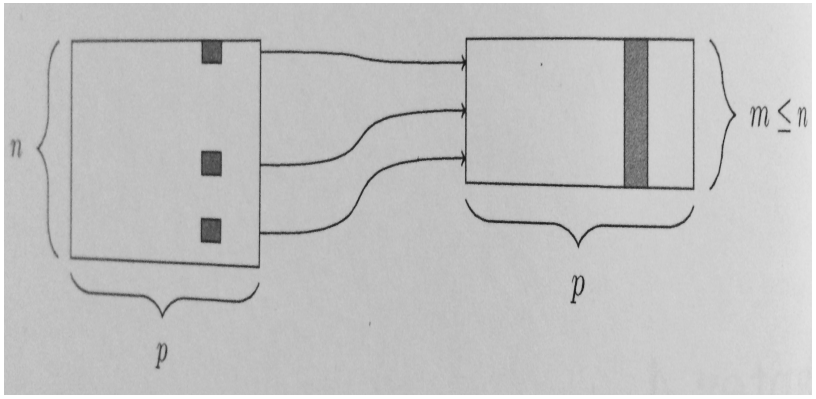
- *dplyr* always takes a data frame as the first argument.
- Subsequent argument tell *dplyr* what to do with the data frame.
- Returns the modified data frame

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
```

```
## # A tibble: 6 x 4
##   name       start      end        party
##   <chr>      <date>     <date>     <chr>
## 1 Eisenhower 1953-01-20 1961-01-20 Republican
## 2 Kennedy    1961-01-20 1963-11-22 Democratic
## 3 Johnson    1963-11-22 1969-01-20 Democratic
## 4 Nixon      1969-01-20 1974-08-09 Republican
## 5 Ford       1974-08-09 1977-01-20 Republican
## 6 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)
```

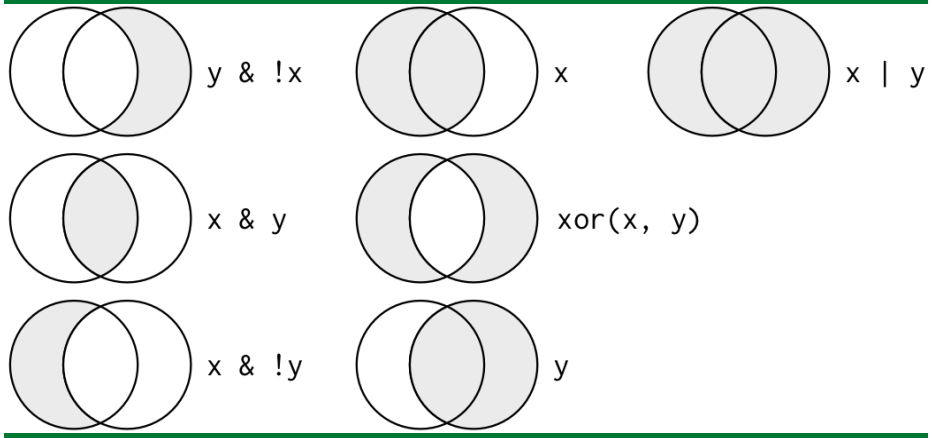
```
## # A tibble: 11 x 3
##    name       end        party
##    <chr>      <date>     <chr>
##  1 Eisenhower 1961-01-20 Republican
##  2 Kennedy    1963-11-22 Democratic
##  3 Johnson    1969-01-20 Democratic
##  4 Nixon      1974-08-09 Republican
##  5 Ford       1977-01-20 Republican
##  6 Carter     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
```


`filter()` boolean logic



(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
```

Note the use of == not =

Dems starting after 2000

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

```
## # A tibble: 1 x 4
##   name  start      end        party
##   <chr> <date>     <date>     <chr>
## 1 Obama 2009-01-20 2017-01-20 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.

Create a new variable

Let's go to a different dataset (we'll talk about this more next class)

```
library(rio)
library(here)
library(janitor)

reads <- import(here("data", "Project_Reads_Scores.csv"),
                setclass = "tbl_df") %>%
  clean_names()
head(reads)
```

```
## # A tibble: 6 x 25
##   test_year test_type test_site student_id pre_test_score pre_test_percent
##   <chr>     <chr>     <chr>     <chr>               <int> <chr>
## 1 06/01/20… YEAR END  VIRDEN    Virden 1               43 29%
## 2 06/01/20… YEAR END  VIRDEN    Virden 2               46 31%
## 3 06/01/20… YEAR END  VIRDEN    Virden 3               39 26%
## 4 06/01/20… YEAR END  VIRDEN    Virden 4               35 23%
## 5 06/01/20… YEAR END  VIRDEN    Virden 5               46 31%
## 6 06/01/20… YEAR END  VIRDEN    Virden 6               35 23%
## # ... with 19 more variables: post_test_score <int>,
## #   post_test_percent <chr>, percentage_change <chr>, unit_1_score <int>,
## #   unit_1_percent <chr>, unit_2_score <int>, unit_2_percent <chr>,
```


- Select the student id, test site, pre- and post-test scores

```
prepost <- reads %>%
    select(student_id, test_site, pre_test_score, post_test_score)

prepost
```

```
## # A tibble: 48 x 4
##    student_id test_site pre_test_score post_test_score
##    <chr>      <chr>              <int>           <int>
##  1 Virden 1   VIRDEN                43              92
##  2 Virden 2   VIRDEN                46             104
##  3 Virden 3   VIRDEN                39              75
##  4 Virden 4   VIRDEN                35             115
##  5 Virden 5   VIRDEN                46              85
##  6 Virden 6   VIRDEN                35              91
##  7 Virden 7   VIRDEN                40              96
##  8 Virden 8   VIRDEN                39              74
##  9 Virden 9   VIRDEN                40              90
## 10 Virden 10  VIRDEN                45              86
## # ... with 38 more rows
```


Add a variable

```
library(magrittr)
prepost %<>%
  mutate(gain = post_test_score - pre_test_score)

prepost
```

```
## # A tibble: 48 x 5
##    student_id test_site pre_test_score post_test_score  gain
##    <chr>      <chr>              <int>           <int> <int>
##  1 Virden 1   VIRDEN                43              92    49
##  2 Virden 2   VIRDEN                46             104    58
##  3 Virden 3   VIRDEN                39              75    36
##  4 Virden 4   VIRDEN                35             115    80
##  5 Virden 5   VIRDEN                46              85    39
##  6 Virden 6   VIRDEN                35              91    56
##  7 Virden 7   VIRDEN                40              96    56
##  8 Virden 8   VIRDEN                39              74    35
##  9 Virden 9   VIRDEN                40              90    50
## 10 Virden 10  VIRDEN                45              86    41
## # ... with 38 more rows
```

Quick note on %<>%

The following are equivalent

```
prepost %<>%
  mutate(gain = post_test_score - pre_test_score)
```

```
prepost <- prepost %>%
  mutate(gain = post_test_score - pre_test_score)
```


Order by gain: Least to greatest

```
prepost %>%
  arrange(gain)
```

```
## # A tibble: 48 x 5
##    student_id  test_site pre_test_score post_test_score  gain
##    <chr>       <chr>              <int>           <int> <int>
##  1 Jones 12    JONES                 27              32     5
##  2 Westside 6  WESTSIDE              57              82    25
##  3 Virden 8    VIRDEN                39              74    35
##  4 Virden 3    VIRDEN                39              75    36
##  5 Virden 5    VIRDEN                46              85    39
##  6 Virden 10   VIRDEN                45              86    41
##  7 Jones 3     JONES                 54              95    41
##  8 Virden 1    VIRDEN                43              92    49
##  9 Jones 13    JONES                 49              98    49
## 10 Westside 13 WESTSIDE              45              94    49
## # ... with 38 more rows
```


Order by gain: greatest to least

```
prepost %>%
  arrange(desc(gain))
```

```
## # A tibble: 48 x 5
##    student_id test_site pre_test_score post_test_score  gain
##    <chr>      <chr>              <int>           <int> <int>
##  1 Jones 11   JONES                 24             108    84
##  2 Virden 4   VIRDEN                35             115    80
##  3 Jones 4    JONES                 25              99    74
##  4 Jones 5    JONES                 36             110    74
##  5 Jones 9    JONES                 35             109    74
##  6 Virden 12  VIRDEN                31             102    71
##  7 Jones 1    JONES                 36             103    67
##  8 Virden 15  VIRDEN                35             101    66
##  9 Jones 6    JONES                 40             106    66
## 10 Westside 8 WESTSIDE              43             109    66
## # ... with 38 more rows
```


`summarize`

- Compute the mean and standard deviation of the gain

```
prepost %>%
  summarize(mean_gain = mean(gain),
            sd_gain = sd(gain))
```

```
## # A tibble: 1 x 2
##   mean_gain sd_gain
##       <dbl>   <dbl>
## 1      56.3    13.6
```

group_by

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

Compute average wins for each team

```
prepost %>%
  group_by(test_site) %>%
  summarize(mean_gain = mean(gain))
```

```
## # A tibble: 6 x 2
##   test_site    mean_gain
##   <chr>            <dbl>
## 1 JONES             59
## 2 JONES ALL         59
## 3 VIRDEN            53.8
## 4 VIRDEN ALL        53
## 5 WESTSIDE          56.3
## 6 WESTSIDE ALL      56
```


Final notes on {dplyr}

- We'll be using it all term long
- 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.