# Review

*Daniel Anderson*
*Week 9, Class 1*

UNIVERSITY OF OREGON

```
library(here)
```

```
## here() starts at /Users/Daniel/Teaching/data_sci_specialization/c1-intro_data_s
```

```
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1
## ✔ ggplot2 3.1.0.9000     ✔ purrr   0.2.5
## ✔ tibble  1.4.2          ✔ dplyr   0.7.7
## ✔ tidyr   0.8.2          ✔ stringr 1.3.1
## ✔ readr   1.1.1          ✔ forcats 0.3.0
## ── Conflicts ────────────────────────────────────────── tidyverse_conflic
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
```

```
library(rio)

theme_set(theme_minimal(base_size = 20))

knitr::opts_chunk$set(fig.width = 13,
                      message = FALSE,
                      warning = FALSE)
options(pillar.sigfig = 7)
```



# Publishing slides w/GitHUb

*Demo*

(RMarkdown Slides and a quick note on {xaringan})



## Tidyverse: Providing a grammar for...

- Graphics (ggplot)
- Data manipulations (dplyr)
- Tidying data (tidyr)

# dplyr



## 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()`
- `group_by`: Group a data frame according to a specific variable (usually a factor)




## Arguments

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



## `select()`

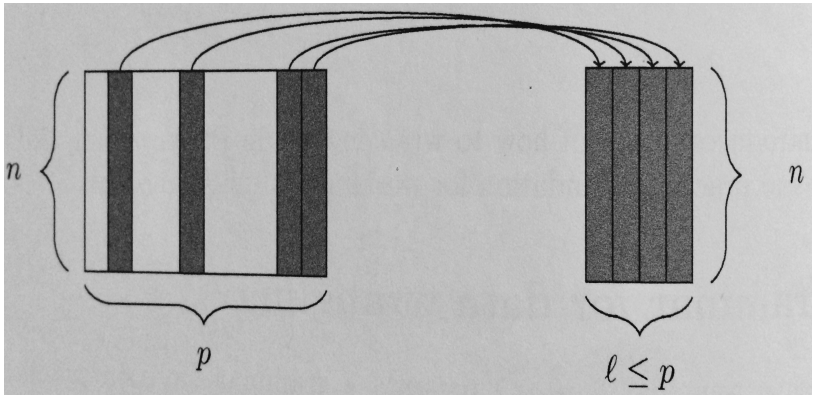



Figure from Baumer, Kaplan, & Horton, 2017

# `filter()`

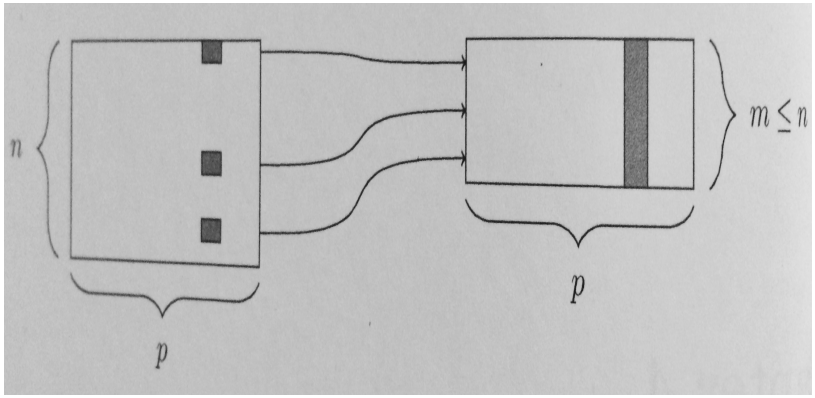



Figure from Baumer, Kaplan, & Horton, 2017



# Helper funs

- `starts_with()`
- `ends_with()`
- `contains()`



# Helper funs

- `starts_with()`
- `ends_with()`
- `contains()`

**Note**: These can be used in other functions as well (e.g., `gather`)



# Mix types

Mix types and helper funs. Use select to rearrange columns.

```
library(tidyverse)
diamonds %>%
    select(depth, 6, starts_with("c"), price)
```

```
## # A tibble: 53,940 x 7
##      depth table    carat cut       color clarity price
##      <dbl> <dbl>    <dbl> <ord>     <ord> <ord>   <int>
##  1 61.5       55 0.23     Ideal     E     SI2       326
##  2 59.8       61 0.21     Premium   E     SI1       326
##  3 56.9       65 0.23     Good      E     VS1       327
##  4 62.4       58 0.290000 Premium   I     VS2       334
##  5 63.3       58 0.31     Good      J     SI2       335
##  6 62.8       57 0.24     Very Good J     VVS2      336
##  7 62.3       57 0.24     Very Good I     VVS1      336
##  8 61.9       55 0.26     Very Good H     SI1       337
##  9 65.1000    61 0.22     Fair      E     VS2       337
## 10 59.4       61 0.23     Very Good H     VS1       338
## # ... with 53,930 more rows
```

# `filter()` boolean logic

Left circle == x, right circle == y

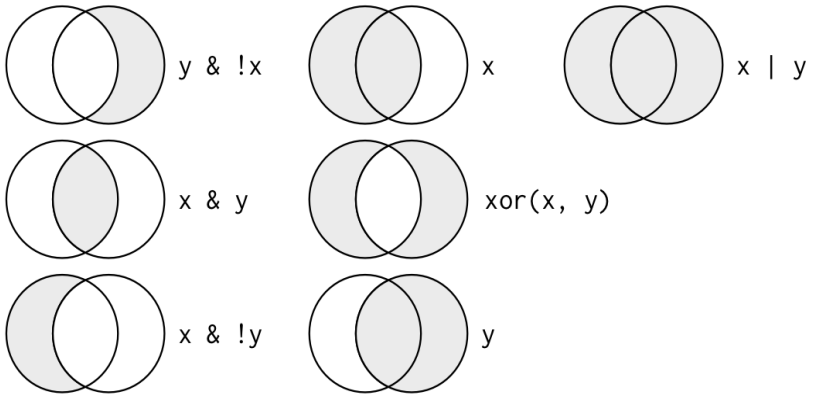



Figure from Wickham & Grolemund, 2017



# Create or overwrite variables: `mutate`

*First: Load the data*

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



# Calculate gain score

```
reads %>%
  mutate(gain = post_test_score - pre_test_score) %>%
  select(student_id, 5, 7, gain)
```

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



# Overwrite a variable

```
reads %>%
  mutate(pre_test_score = scale(pre_test_score),
         post_test_score = scale(post_test_score)) %>%
  select(student_id, 5, 7)
```

```
## # A tibble: 48 x 3
##    student_id pre_test_score post_test_score
##    <chr>                <dbl>           <dbl>
##  1 Virden 1         0.2241918      -0.4333331
##  2 Virden 2         0.6380844       0.4776153
##  3 Virden 3        -0.3276650      -1.723843
##  4 Virden 4        -0.8795218       1.312651
##  5 Virden 5         0.6380844      -0.9647197
##  6 Virden 6        -0.8795218      -0.5092455
##  7 Virden 7        -0.1897008      -0.1296836
##  8 Virden 8        -0.3276650      -1.799756
##  9 Virden 9        -0.1897008      -0.5851578
## 10 Virden 10        0.5001202      -0.8888073
## # ... with 38 more rows
```

# `summarize` column(s)

- Average pre/post and gain

```
reads %>%
  mutate(gain = post_test_score - pre_test_score) %>%
  summarize(av_pre = mean(pre_test_score, na.rm = TRUE),
            av_post = mean(post_test_score, na.rm = TRUE),
            av_gain = mean(gain, na.rm = TRUE))
```

```
## # A tibble: 1 x 3
##   av_pre  av_post  av_gain
##    <dbl>    <dbl>    <dbl>
## 1 41.375 97.70833 56.33333
```



# Summaries by group

- Use `group_by` to get the same summaries by a grouping factor. Mean pre/post and gain by test site

```
reads %>%
  mutate(gain = post_test_score - pre_test_score) %>%
  group_by(test_site) %>%
  summarize(av_pre = mean(pre_test_score, na.rm = TRUE),
            av_post = mean(post_test_score, na.rm = TRUE),
            av_gain = mean(gain, na.rm = TRUE))
```

```
## # A tibble: 6 x 4
##   test_site      av_pre   av_post  av_gain
##   <chr>           <dbl>     <dbl>    <dbl>
## 1 JONES        38.800    97.8     59
## 2 JONES ALL    39        98       59
## 3 VIRDEN       38.66667  92.46667 53.8
## 4 VIRDEN ALL   39        92       53
## 5 WESTSIDE     46.6     102.8667  56.26667
## 6 WESTSIDE ALL 47       103       56
```



# Conditional verbs

*I mentioned these, but we never covered them*

- `*_if`, `*_all` functions can help with efficiency

```
reads %>%
  select_if(is.numeric) %>%
  summarize_all(mean, na.rm = TRUE)
```

```
## # A tibble: 1 x 10
##   pre_test_score post_test_score unit_1_score unit_2_score unit_3_score
##            <dbl>           <dbl>        <dbl>        <dbl>        <dbl>
## 1         41.375        97.70833     3.270833     4.270833     5.395833
## # ... with 5 more variables: unit_4_score <dbl>, unit_5_6_score <dbl>,
## #   unit_7_score <dbl>, unit_8_score <dbl>, total_score <dbl>
```



- `*_at`

```
reads %>%
  summarize_at(vars(unit_1_score, unit_3_score, unit_7_score),
               funs(mean, sd))
```

```
## # A tibble: 1 x 6
##   unit_1_score_mean unit_3_score_mean unit_7_score_mean unit_1_score_sd
##               <dbl>             <dbl>             <dbl>           <dbl>
## 1          3.270833          5.395833            16.375        1.267257
## # ... with 2 more variables: unit_3_score_sd <dbl>, unit_7_score_sd <dbl>
```

# Data visualization



## Reviewing ggplot2 syntax

- `aes()` accesses the variables in your dataset.



## Reviewing ggplot2 syntax

- `aes()` accesses the variables in your dataset.
- Set aesthetics globally outside of `aes()`



## Reviewing ggplot2 syntax

- `aes()` accesses the variables in your dataset.
- Set aesthetics globally outside of `aes()`
- Aesthetics set in first call to `ggplot` will bleed through to subsequent layers.

# Reviewing ggplot2 syntax

- `aes()` accesses the variables in your dataset.
- Set aesthetics globally outside of `aes()`
- Aesthetics set in first call to `ggplot` will bleed through to subsequent layers.
- Change aesthetics within a specific layer by providing arguments within that specific function.



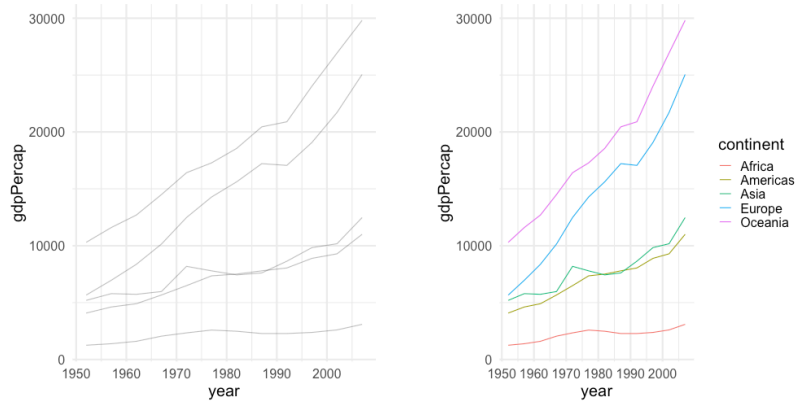
# Reviewing ggplot2 syntax

- `aes()` accesses the variables in your dataset.
- Set aesthetics globally outside of `aes()`
- Aesthetics set in first call to `ggplot` will bleed through to subsequent layers.
- Change aesthetics within a specific layer by providing arguments within that specific function.
- Can even change x/y axes, which can be helpful if you want it defined as categorical for one layer and continuous for a second. Or if you want to plot summary data over the top of the raw data (or vice versa).



# Pop Quiz 1

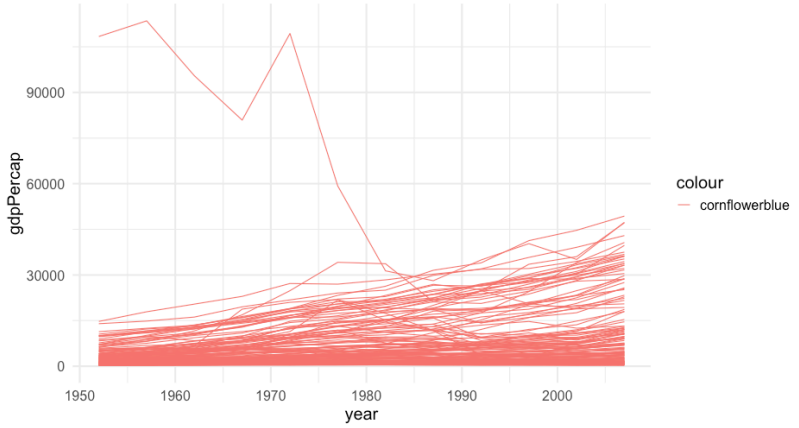
What's different about these plots






# Pop Quiz 2

What went wrong here?

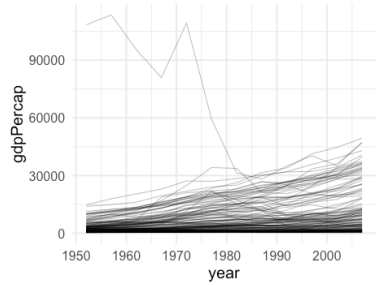


# Thinking about scales some

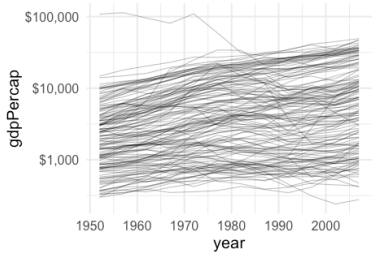
*A bit more new stuff*

- Same plot, different scales. Which can you see the relation better?

```
ggplot(gapminder, aes(year, gdpPerc
  geom_line(alpha = 0.3)
```




```
ggplot(gapminder, aes(year, gdpPerc
  geom_line(alpha = 0.3) +
  scale_y_log10(label = scales::dol
```

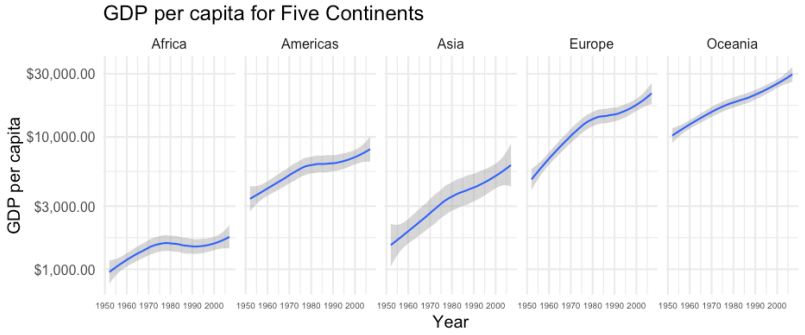





# Pop Quiz 3

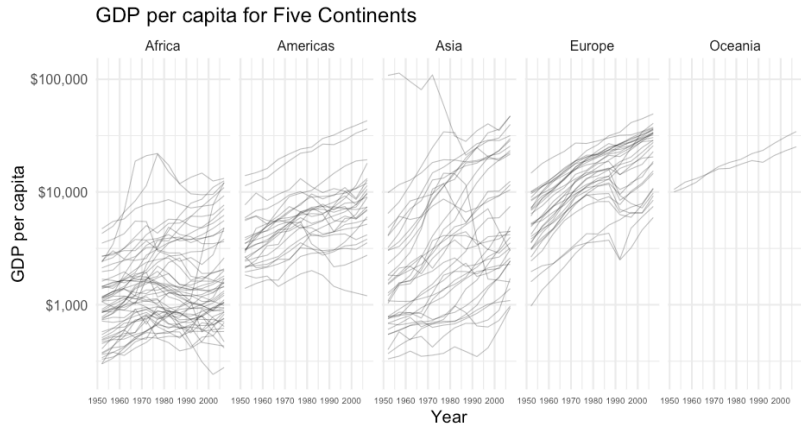
Can you reproduce this plot? (you'll need the gapminder package/data)

```
# install.packages(gapminder)
library(gapminder)
```



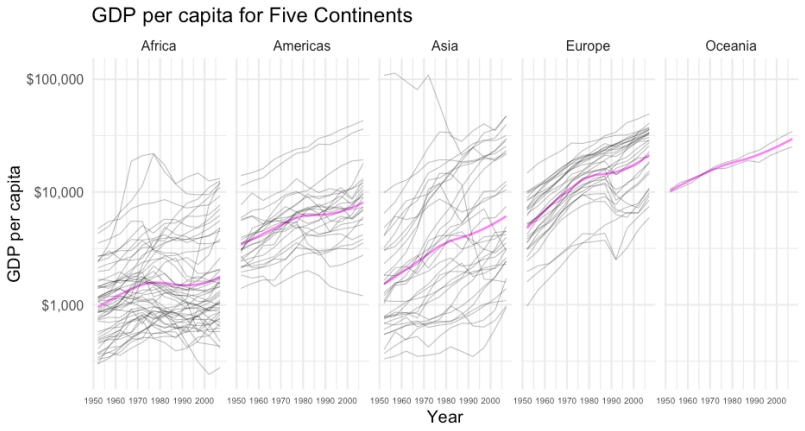



# What about this one?






# Can you combine them?



# tidy data and *tidyr*



# Review Homework 4

- Tidy the reads dataset, minus columns 5-9

```
reads %>%
  select(-5:-9)
```

```
## # A tibble: 48 x 20
##    test_year    test_type test_site student_id unit_1_score unit_1_percent
##    <chr>        <chr>     <chr>     <chr>              <int> <chr>
##  1 06/01/2016 … YEAR END  VIRDEN    Virden 1               3 12%
##  2 06/01/2016 … YEAR END  VIRDEN    Virden 2               5 20%
##  3 06/01/2016 … YEAR END  VIRDEN    Virden 3               4 16%
##  4 06/01/2016 … YEAR END  VIRDEN    Virden 4               4 16%
##  5 06/01/2016 … YEAR END  VIRDEN    Virden 5               2 8%
##  6 06/01/2016 … YEAR END  VIRDEN    Virden 6               5 20%
##  7 06/01/2016 … YEAR END  VIRDEN    Virden 7               5 20%
##  8 06/01/2016 … YEAR END  VIRDEN    Virden 8               4 16%
##  9 06/01/2016 … YEAR END  VIRDEN    Virden 9               6 24%
## 10 06/01/2016 … YEAR END  VIRDEN    Virden 10              4 16%
## # ... with 38 more rows, and 14 more variables: unit_2_score <int>,
## #   unit_2_percent <chr>, unit_3_score <int>, unit_3_percent <chr>,
## #   unit_4_score <int>, unit_4_percent <chr>, unit_5_6_score <int>,
## #   unit_5_6_percent <chr>, unit_7_score <int>, unit_7_percent <chr>,
## #   unit_8_score <int>, unit_8_percent <chr>, total_score <int>,
```



# Common mistakes

- Forget to filter out summary rows
- Multiple gathers



# Common thought process

- `gather` once for raw scores, and a second time for percentages.
- Anything wrong with this?

```
double_gather <- reads %>%
  select(4, 1:3, 10:23) %>%
  filter(student_id != "All Students (Average)") %>%
  gather(unit, score, contains("score")) %>%
  gather(unit2, percent, contains("percent")) %>%
  mutate(unit = readr::parse_number(unit)) %>%
  select(-unit2)
```

```
double_gather
```

```
## # A tibble: 2,205 x 7
##    student_id test_year            test_type test_site  unit score percent
##    <chr>      <chr>                <chr>     <chr>     <dbl> <int> <chr>
##  1 Virden 1   06/01/2016 12:00:00… YEAR END  VIRDEN        1     3 12%
##  2 Virden 2   06/01/2016 12:00:00… YEAR END  VIRDEN        1     5 20%
##  3 Virden 3   06/01/2016 12:00:00… YEAR END  VIRDEN        1     4 16%
##  4 Virden 4   06/01/2016 12:00:00… YEAR END  VIRDEN        1     4 16%
##  5 Virden 5   06/01/2016 12:00:00… YEAR END  VIRDEN        1     2 8%
##  6 Virden 6   06/01/2016 12:00:00… YEAR END  VIRDEN        1     5 20%
##  7 Virden 7   06/01/2016 12:00:00… YEAR END  VIRDEN        1     5 20%
##  8 Virden 8   06/01/2016 12:00:00… YEAR END  VIRDEN        1     4 16%
##  9 Virden 9   06/01/2016 12:00:00… YEAR END  VIRDEN        1     6 24%
## 10 Virden 10  06/01/2016 12:00:00… YEAR END  VIRDEN        1     4 16%
## # ... with 2,195 more rows
```



# What's going on?

```
double_gather %>%
  count(student_id)
```

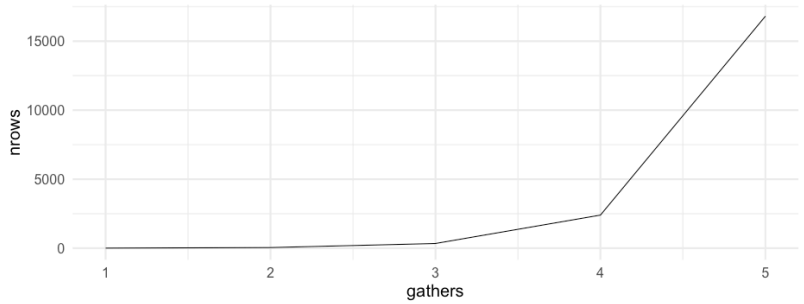
```
## # A tibble: 45 x 2
##    student_id     n
##    <chr>      <int>
##  1 Jones 1       49
##  2 Jones 10      49
##  3 Jones 11      49
##  4 Jones 12      49
##  5 Jones 13      49
##  6 Jones 14      49
##  7 Jones 15      49
##  8 Jones 2       49
##  9 Jones 3       49
## 10 Jones 4       49
## # ... with 35 more rows
```



# n rows by n gathers

- Assuming 7 columns are gathered each time:

```
expo <- tibble(gathers = 1:5, nrows = 7^(1:5))
ggplot(expo, aes(gathers, nrows)) + geom_line()
```






# Alternative

- Gather all, then spread. **BUT**, requires common names to parse out the variables that are currently stored in the column names.

```
reads_tidy <- reads %>%
  select(4, 1:3, 10:23) %>%
  rename("unit_56_score" = "unit_5_6_score",
         "unit_56_percent" = "unit_5_6_percent") %>%
  filter(student_id != "All Students (Average)") %>%
  gather(var, val, -1:-4) %>%
  separate(var, c("dis", "unit", "type"), sep = "_") %>%
  select(-dis) %>%
  spread(type, val)
```

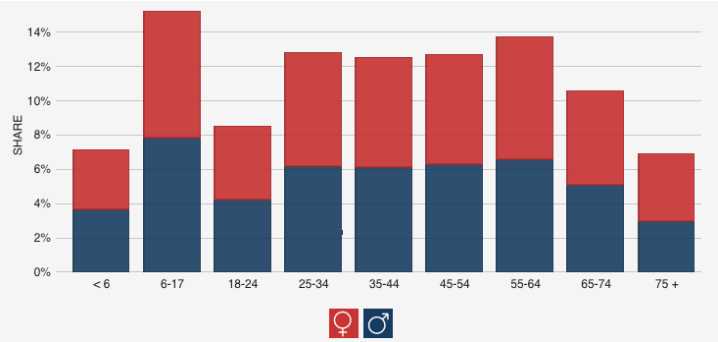
```
reads_tidy
```

```
## # A tibble: 315 x 7
##    student_id test_year            test_type test_site unit  percent score
##    <chr>      <chr>                <chr>     <chr>     <chr> <chr>   <chr>
##  1 Jones 1    06/01/2016 12:00:00… YEAR END  JONES     1     16%     4
##  2 Jones 1    06/01/2016 12:00:00… YEAR END  JONES     2     16%     4
##  3 Jones 1    06/01/2016 12:00:00… YEAR END  JONES     3     20%     5
##  4 Jones 1    06/01/2016 12:00:00… YEAR END  JONES     4     33%     10
##  5 Jones 1    06/01/2016 12:00:00… YEAR END  JONES     56    30%     9
##  6 Jones 1    06/01/2016 12:00:00… YEAR END  JONES     7     63%     22
##  7 Jones 1    06/01/2016 12:00:00… YEAR END  JONES     8     53%     16
##  8 Jones 10   06/01/2016 12:00:00… YEAR END  JONES     1     12%     3
##  9 Jones 10   06/01/2016 12:00:00… YEAR END  JONES     2     12%     3
## 10 Jones 10   06/01/2016 12:00:00… YEAR END  JONES     3     20%     5
## # ... with 305 more rows
```



# Quick challenge (15 minutes)

- The following plot comes from datausa for Oregon residents.
- The data are stored in a file called *insurance_coverage.csv*
  - Load the data into R and reproduce the plot as best you can






# My method

- Load the data

```
insurance <- import(here("data", "insurance_coverage.csv"),
                    skip =  3,
                    setclass = "tbl_df") %>%
  janitor::clean_names()
insurance
```

```
## # A tibble: 36 x 11
##    in_oregon_csvage_bucket insurance_name  insurance   year geo_name geo
##    <chr>                   <chr>           <chr>      <int> <chr>    <chr>
##  1 insurance_18to24        healthcare cov… has_insur…  2014 Oregon   0400…
##  2 insurance_18to24        healthcare cov… has_insur…  2014 Oregon   0400…
##  3 insurance_25to34        healthcare cov… has_insur…  2014 Oregon   0400…
##  4 insurance_25to34        healthcare cov… has_insur…  2014 Oregon   0400…
##  5 insurance_35to44        healthcare cov… has_insur…  2014 Oregon   0400…
##  6 insurance_35to44        healthcare cov… has_insur…  2014 Oregon   0400…
##  7 insurance_45to54        healthcare cov… has_insur…  2014 Oregon   0400…
##  8 insurance_45to54        healthcare cov… has_insur…  2014 Oregon   0400…
##  9 insurance_55to64        healthcare cov… has_insur…  2014 Oregon   0400…
## 10 insurance_55to64        healthcare cov… has_insur…  2014 Oregon   0400…
## # ... with 26 more rows, and 5 more variables: sex_name <chr>, sex <int>,
## #   hc_pop <int>, hc_pop_moe <dbl>, hc_pop_rca <dbl>
```



# Compute percentages

```
smry <- insurance %>%
  group_by(in_oregon_csvage_bucket, sex_name) %>%
  summarize(by_sex = sum(hc_pop)) %>%
  ungroup() %>%
  mutate(percentage = by_sex /sum(by_sex))

smry
```

```
## # A tibble: 18 x 4
##    in_oregon_csvage_bucket sex_name by_sex percentage
##    <chr>                   <chr>     <int>      <dbl>
##  1 insurance_18to24        Female   315069 0.04340083
##  2 insurance_18to24        Male     308047 0.04243355
##  3 insurance_25to34        Female   471116 0.06489635
##  4 insurance_25to34        Male     428039 0.05896248
##  5 insurance_35to44        Female   462859 0.06375894
##  6 insurance_35to44        Male     447112 0.06158979
##  7 insurance_45to54        Female   463471 0.06384325
##  8 insurance_45to54        Male     446924 0.06156389
##  9 insurance_55to64        Female   523872 0.07216350
## 10 insurance_55to64        Male     476654 0.06565921
## 11 insurance_65to74        Female   398558 0.05490146
## 12 insurance_65to74        Male     363970 0.05013696
## 13 insurance_6to17         Female   544234 0.07496837
```

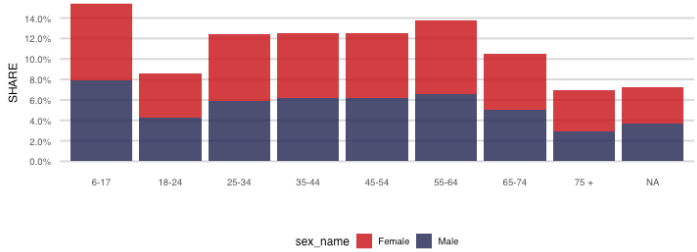
# Reorder and re-level age ranges

```
smry <- smry %>%
  separate(in_oregon_csvage_bucket, c("dis", "age_range")) %>%
  mutate(age_range = factor(age_range,
                            levels = c("under_6",
                                       "6to17",
                                       "18to24",
                                       "25to34",
                                       "35to44",
                                       "45to54",
                                       "55to64",
                                       "65to74",
                                       "75plus"),
                            labels = c("< 6",
                                       "6-17",
                                       "18-24",
                                       "25-34",
                                       "35-44",
                                       "45-54",
                                       "55-64",
                                       "65-74",
                                       "75 +")))
```



# Produce the plot

```
ggplot(smry, aes(age_range, percentage, fill = sex_name)) +
  geom_col(alpha = 0.9) +
  scale_fill_manual(values = c("firebrick3", "#323560")) +
  scale_y_continuous(name = "SHARE",
                     breaks = seq(0, 0.14, 0.02),
                     labels = scales::percent) +
  xlab("") +
  ggthemes::theme_hc()
```



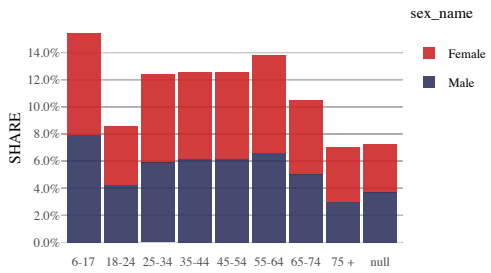



# Make it interactive

```
library(plotly)
p <- ggplot(smry, aes(age_range, percentage, fill = sex_name)) +
  geom_bar(stat = "identity",  alpha = 0.9) +
  scale_fill_manual(values = c("firebrick3", "#323560")) +
  scale_y_continuous(name = "SHARE",
                     breaks = seq(0, 0.14, 0.02),
                     labels = scales::percent) +
  xlab("") +
  ggthemes::theme_hc()
```



```
ggplotly(p)
```



# Lab

*Get some slides published!*