

Data Wrangling in R with the Tidyverse (Part 1)

Jessica Minnier, PhD & Meike Niederhausen, PhD

OCTRI Biostatistics, Epidemiology, Research & Design (BERD) Workshop

2019/04/18 (Part 1) & 2019/04/25 (Part 2)

 slides: bit.ly/berd_tidy

 pdf: bit.ly/berd_tidy_pdf

Load files for today's workshop

- Open the slides of this workshop:
bit.ly/berd_tidy
- Open the pre-workshop homework: [find it here](#)
- Download & open the HW project file
`berd_tidyverse_project.Rproj`
- Open a new script and run the commands to the right

```
# install.packages("tidyverse")
library(tidyverse)
library(lubridate)
demo_data <-
read_csv("data/yrbss_demo.csv")
```



Learning objectives

Part 1:

- What is data wrangling?
- A few good practices in R/RStudio
- What is tidy data?
- What is tidyverse?
- Reshape and manipulate data frames

Part 2:

- Some data cleaning

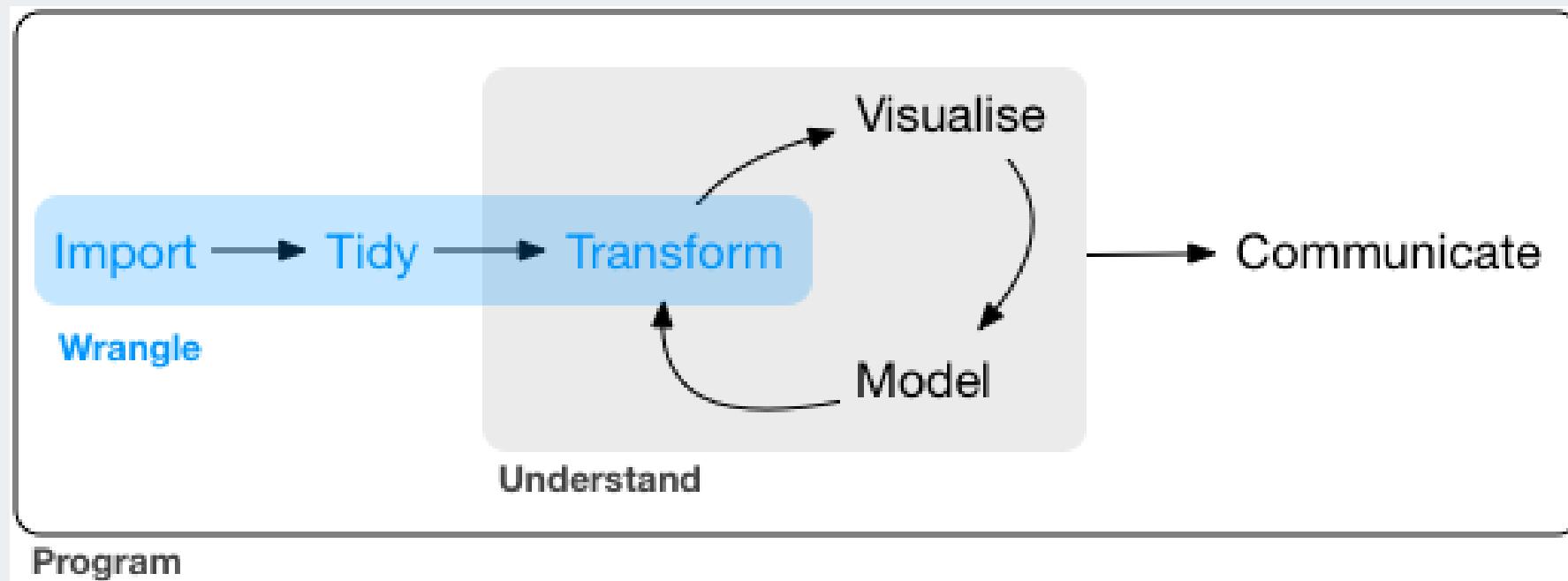
Getting started



Alison Horst

What is data wrangling?

- "data janitor work"
 - importing data
 - cleaning data
 - changing shape of data
- fixing errors and poorly formatted data elements
 - transforming columns and rows
 - filtering, subsetting



Good practices in RStudio

Use projects ([read this](#))

- Create an RStudio project for each data analysis project
- A project is associated with a directory folder
 - Keep data files there
 - Keep scripts there; edit them, run them in bits or as a whole
 - Save your outputs (plots and cleaned data) there
- Only use relative paths, never absolute paths
 - relative (good): `read_csv("data/mydata.csv")`
 - absolute (bad): `read_csv("/home/yourname/Documents/stuff/mydata.csv")`

Advantages of using projects

- standardize file paths
- keep everything together
- a whole folder can be shared and run on another computer

Useful keyboard shortcuts

action	mac	windows/linux
run code in script	cmd + enter	ctrl + enter
<-	option + -	alt + -
%>%	cmd+shift+m	ctrl + shift + m

Others: ([see list](#))

action	mac	windows/linux
interrupt currently executing command	esc	esc
in console, go to previously run code	up/down	up/down
keyboard shortcut help	option+shift+k	alt+shift+k

Try typing and running

```
y <- 5  
y
```

Now, in the console, press the up arrow.

Tibbles



hexbin

Data frames vs. tibbles

Previously we learned about *data frames*

```
data.frame(name = c("Sarah", "Ana", "Jose"),  
           rank = 1:3,  
           age = c(35.5, 25, 58),  
           city = c(NA, "New York", "LA"))
```

	name	rank	age	city
1	Sarah	1	35.5	<NA>
2	Ana	2	25.0	New York
3	Jose	3	58.0	LA

A *tibble* is a data frame but with perks

```
tibble(name = c("Sarah", "Ana", "Jose"),  
       rank = 1:3,  
       age = c(35.5, 25, 58),  
       city = c(NA, "New York", "LA"))
```

```
# A tibble: 3 x 4  
  name    rank    age   city  
  <chr> <int> <dbl> <chr>  
1 Sarah     1    35.5 <NA>  
2 Ana       2     25  New York  
3 Jose      3     58    LA
```

How are these two datasets different?

Import data as a data frame (try this)

Base R functions import data as data frames (`read.csv`, `read.table`, etc)

```
mydata_df <- read.csv("data/small_data.csv")
mydata_df
```

	id	age	sex	grade	race4
1	335340	17 years old	Female	10th	White
2	638618	16 years old	Female	9th	<NA>
3	922382	14 years old	Male	9th	White
4	923122	15 years old	Male	9th	White
5	923963	15 years old	Male	10th	Black or African American
6	925603	16 years old	Male	10th	All other races
7	933724	16 years old	Female	10th	All other races
8	935435	17 years old	Female	12th	All other races
9	1096564	15 years old	Male	10th	All other races
10	1108114	17 years old	Female	9th	Black or African American
11	1306150	16 years old	Male	10th	Hispanic/Latino
12	1307481	17 years old	Male	12th	Hispanic/Latino
13	1307872	17 years old	Male	11th	Hispanic/Latino
14	1311617	15 years old	Female	10th	Hispanic/Latino
15	1313153	16 years old	Female	11th	Hispanic/Latino
16	1313291	16 years old	Female	11th	White

Import data as a tibble (try this)

tidyverse functions import data as tibbles (read_csv, read_excel(), etc)

```
mydata_tib <- read_csv("data/small_data.csv")
mydata_tib
```

```
# A tibble: 20 x 11
  id age sex grade race4 bmi weight_kg text_while_driv...
  <dbl> <chr> <chr> <chr> <chr> <dbl>     <dbl> <chr>
1 3.35e5 17 y... Fema... 10th White  27.6      66.2 <NA>
2 6.39e5 16 y... Fema... 9th   <NA>  29.3      84.8 <NA>
3 9.22e5 14 y... Male   9th   White  18.2      57.6 <NA>
4 9.23e5 15 y... Male   9th   White  21.4      60.3 <NA>
5 9.24e5 15 y... Male   10th  Blac... 19.6      63.5 <NA>
6 9.26e5 16 y... Male   10th  All ... 22.2      70.3 <NA>
7 9.34e5 16 y... Fema... 10th  All ... 21.0      45.4 <NA>
8 9.35e5 17 y... Fema... 12th  All ... 17.5      43.1 <NA>
9 1.10e6 15 y... Male   10th  All ... 22.5      79.4 <NA>
10 1.11e6 17 y... Fema... 9th   Blac... 26.6      68.0 <NA>
11 1.31e6 16 y... Male   10th  Hisp... 21.2      67.1 0 days
12 1.31e6 17 y... Male   12th  Hisp... 19.5      56.2 1 or 2 days
13 1.31e6 17 y... Male   11th  Hisp... 20.6      61.7 1 or 2 days
14 1.31e6 15 y... Fema... 10th  Hisp... 27.5      70.3 0 days
```

Compare & contrast data frame and tibble

Run the code below

data frame

```
glimpse(mydata_df)
str(mydata_df)      # How are glimpse() and str() different?
head(mydata_df)
summary(mydata_df)
class(mydata_df)    # What information does class() give?
```

tibble

```
glimpse(mydata_tib)
str(mydata_tib)
head(mydata_tib)
summary(mydata_tib)
class(mydata_tib)
```

Tibble perks

Viewing tibbles:

- variable types are given (character, factor, double, integer, boolean, date)
- number of rows & columns shown are limited for easier viewing

Other perks:

- tibbles can typically be used anywhere a `data.frame` is needed
- `read_*`() functions don't read character columns as factors (no surprises)

Tidy Data



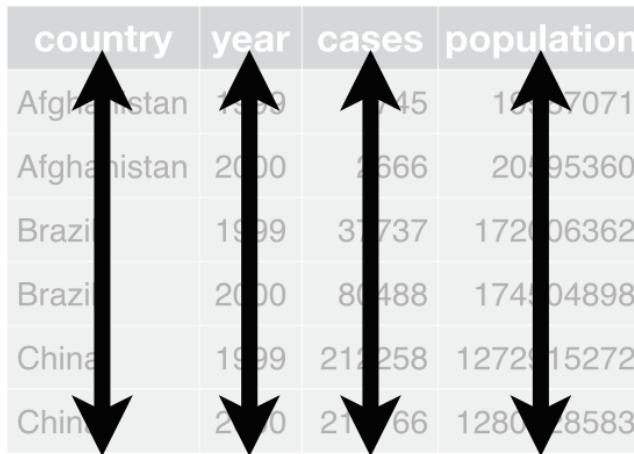
Allison Horst

What are tidy data?

1. Each variable forms a column
2. Each observation forms a row
3. Each value has its own cell

country	year	cases	population
Afghanistan	1990	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	128042583

variables



country	year	cases	population
Afghanistan	1990	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	128042583

observations



country	year	cases	population
Afghanistan	99	745	19987071
Afghanistan	00	2666	20595360
Brazil	99	37737	172006362
Brazil	00	80488	174504898
China	99	212258	1272915272
China	00	213766	128042583

values



Untidy data: example 1

```
untidy_data <- tibble(  
  name = c("Ana", "Bob", "Cara"),  
  meds = c("advil 500mg 2xday", "tylenol 1000mg 1xday", "advil 200mg 3xday")  
)  
untidy_data
```

```
# A tibble: 3 x 2  
  name   meds  
  <chr> <chr>  
1 Ana    advil 500mg 2xday  
2 Bob    tylenol 1000mg 1xday  
3 Cara   advil 200mg 3xday
```

Tidy data: example 1

You will learn how to do this!

```
untidy_data %>%  
  separate(col = meds, into = c("med_name", "dose_mg", "times_per_day"), sep = " ") %>%  
  mutate(times_per_day = as.numeric(str_remove(times_per_day, "xday")),  
         dose_mg = as.numeric(str_remove(dose_mg, "mg")))
```

```
# A tibble: 3 x 4  
  name   med_name dose_mg times_per_day  
  <chr>  <chr>     <dbl>        <dbl>  
1 Ana    advil      500          2  
2 Bob    tylenol    1000         1  
3 Cara   advil      200          3
```

Untidy data: example 2

```
untidy_data2 <- tibble(  
  name = c("Ana", "Bob", "Cara"),  
  wt_07_01_2018 = c(100, 150, 140),  
  wt_08_01_2018 = c(104, 155, 138),  
  wt_09_01_2018 = c(NA, 160, 142)  
)  
untidy_data2
```

```
# A tibble: 3 x 4  
  name   wt_07_01_2018 wt_08_01_2018 wt_09_01_2018  
  <chr>     <dbl>        <dbl>        <dbl>  
1 Ana         100          104          NA  
2 Bob         150          155          160  
3 Cara        140          138          142
```

Tidy data: example 2

You will learn how to do this!

```
untidy_data2 %>%  
  gather(key = "date", value = "weight", -name) %>%  
  mutate(date = str_remove(date, "wt_"),  
         date = dmy(date))      # dmy() is a function in the lubridate package
```

```
# A tibble: 9 x 3  
  name   date       weight  
  <chr> <date>     <dbl>  
1 Ana    2018-01-07     100  
2 Bob    2018-01-07     150  
3 Cara   2018-01-07     140  
4 Ana    2018-01-08     104  
5 Bob    2018-01-08     155  
6 Cara   2018-01-08     138  
7 Ana    2018-01-09     NA  
8 Bob    2018-01-09     160  
9 Cara   2018-01-09     142
```

How to tidy?



Allison Horst

Tools for tidying data

- tidyverse functions
 - **tidyverse** is a suite of packages that implement **tidy** methods for data importing, cleaning, and wrangling
 - load the **tidyverse** packages by running the code `library(tidyverse)`
 - see pre-workshop homework for code to install **tidyverse**
- Functions to easily work with rows and columns, such as
 - subset rows/columns
 - add new rows/columns
 - split apart or unite columns
 - join together different data sets (part 2)
 - make data *long* or *wide* (part 2)
- Often many steps to tidy data
 - string together commands to be performed sequentially
 - do this using pipes `%>%`

How to use the pipe %>%

The pipe operator %>% strings together commands to be performed sequentially

```
mydata_tib %>% head(n=3)      # prounounce %>% as "then"

# A tibble: 3 x 11
  id age sex   grade race4   bmi weight_kg text_while_driv...
  <dbl> <chr> <chr> <chr> <dbl>    <dbl> <chr>
1 335340 17 y... Fema... 10th  White  27.6     66.2 <NA>
2 638618 16 y... Fema... 9th   <NA>   29.3     84.8 <NA>
3 922382 14 y... Male   9th   White  18.2     57.6 <NA>
# ... with 3 more variables: smoked_ever <chr>, bullied_past_12mo <lgl>,
#   height_m <dbl>
```

- Always *first list the tibble* that the commands are being applied to
- Can use **multiple pipes** to run multiple commands in sequence
 - What does the following code do?

```
mydata_tib %>% head(n=3) %>% summary()
```

About the data

Data from the CDC's [Youth Risk Behavior Surveillance System \(YRBSS\)](#)

- complex survey data
- national school-based survey
 - conducted by CDC and state, territorial, and local education and health agencies and tribal governments
- monitors six categories of health-related behaviors
 - that contribute to the leading causes of death and disability among youth and adults
 - including alcohol & drug use, unhealthy & dangerous behaviors, sexuality, and physical activity
 - see [Questionnaires](#)
- the data in `yrbss_demo.csv` are a subset of data in the R package `yrbss`, which includes YRBSS from 1991-2013
- Look at your *Environment* tab to make sure `demo_data` is already loaded

```
demo_data <- read_csv("data/yrbss_demo.csv")
```

Subsetting data

Subset Observations (Rows)



Subset Variables (Columns)



filter() ~ rows

filter data based on rows

- math: >, <, >=, <=
- double = for "is equal to": ==
- & (and)
- | (or)
- ! in front negates the statement, as in != or !(grade=="9th")
- is.na() function to filter based on missing values
- %in% to filter based on group membership

```
demo_data %>% filter(bmi > 20)
```

	# A tibble: 10,375 x 8	record	age	sex	grade	race4	race7	bmi	stweight
		<dbl>	<chr>	<chr>	<chr>	<chr>	<chr>	<dbl>	<dbl>
1	333862	17	years	o...	Fema...	12th	White	20.2	57.2
2	1095530	15	years	o...	Male	10th	Black or Af...	28.0	85.7
3	1303997	14	years	o...	Male	9th	All other r...	24.5	66.7
4	926649	16	years	o...	Male	11th	All other r...	20.5	70.3
5	506337	18	years	o...	Male	12th	Hispanic/La...	33.1	123.
6	1307180	16	years	o...	Male	10th	Hispanic/La...	21.8	66.7

Compare to base R

- **Bracket method:** need to repeat tibble name
- Need to use \$
- Very nested and confusing to read

```
demo_data[demo_data$grade=="9th",]
```

```
# A tibble: 5,625 x 8
  record age      sex    grade race4
  <dbl> <chr>   <chr>  <chr> <chr>
1 1303997 14 years... Male   9th   All other
2 261619  17 years... Male   9th   All other
3 1096939 15 years... Male   9th   <NA>
4 180968  15 years... Male   9th   White
5 924270  15 years... Male   9th   All other
6 330828  15 years... Female 9th   Hispanic/
7 1311252 15 years... Female 9th   Hispanic/
8 36853   14 years... Female 9th   All other
9 1310689 14 years... Female 9th   Hispanic/
10 1310726 14 years... Female 9th  All other
# ... with 5,615 more rows
```

- **Pipe method:** list tibble name once
- No \$ needed since uses "non-standard evaluation": `filter()` knows `grade` is a column in `demo_data`

```
demo_data %>% filter(grade=="9th")
```

```
# A tibble: 5,219 x 8
  record age      sex    grade race4
  <dbl> <chr>   <chr>  <chr> <chr>
1 1303997 14 years... Male   9th   All other
2 261619  17 years... Male   9th   All other
3 1096939 15 years... Male   9th   <NA>
4 180968  15 years... Male   9th   White
5 924270  15 years... Male   9th   All other
6 330828  15 years... Female 9th   Hispanic/
7 1311252 15 years... Female 9th   Hispanic/
8 36853   14 years... Female 9th  All other
9 1310689 14 years... Female 9th   Hispanic/
10 1310726 14 years... Female 9th All other
# ... with 5,209 more rows
```

filter() practice

What do these commands do? Try them out:

```
demo_data %>% filter(bmi < 5)
demo_data %>% filter(bmi/stweight < 0.5)      # can do math
demo_data %>% filter((bmi < 15) | (bmi > 50))
demo_data %>% filter(bmi < 20, stweight < 50, sex == "Female") # filter on multiple variable

demo_data %>% filter(record == 506901)          # note the use of == instead of just =
demo_data %>% filter(sex == "Male")
demo_data %>% filter(!(grade == "9th"))
demo_data %>% filter(grade %in% c("10th", "11th"))

demo_data %>% filter(is.na(bmi))
demo_data %>% filter(!is.na(bmi))
```

Subset Variables (Columns)



tidyverse data wrangling cheatsheet

select() ~ columns

- select columns (variables)
- no quotes needed around variable names
- can be used to rearrange columns
- uses special syntax that is flexible and has many options

```
demo_data %>% select(record, grade)
```

```
# A tibble: 20,000 x 2
  record  grade
  <dbl> <chr>
1 931897 10th
2 333862 12th
3 36253 11th
4 1095530 10th
5 1303997 9th
6 261619 9th
7 926649 11th
8 1309082 12th
9 506337 12th
10 180494 10th
# ... with 19,990 more rows
```

Compare to base R

- Need brackets
- Need quotes around column names

```
demo_data[, c("record", "age", "sex")]
```

```
# A tibble: 20,000 x 3
  record    age      sex
  <dbl> <chr>   <chr>
1 931897 15 years old Female
2 333862 17 years old Female
3 36253 18 years old or older Male
4 1095530 15 years old Male
5 1303997 14 years old Male
6 261619 17 years old Male
7 926649 16 years old Male
8 1309082 17 years old Male
9 506337 18 years old or older Male
10 180494 14 years old Male
# ... with 19,990 more rows
```

- No quotes needed and easier to read.
- More flexible, either of following work:

```
demo_data %>% select(record, age, sex)
demo_data %>% select(record:sex)
```

```
# A tibble: 20,000 x 3
  record    age      sex
  <dbl> <chr>   <chr>
1 931897 15 years old Female
2 333862 17 years old Female
3 36253 18 years old or older Male
4 1095530 15 years old Male
5 1303997 14 years old Male
6 261619 17 years old Male
7 926649 16 years old Male
8 1309082 17 years old Male
9 506337 18 years old or older Male
10 180494 14 years old Male
# ... with 19,990 more rows
# A tibble: 20,000 x 3
```

Column selection syntax options

There are many ways to select a set of variable names (columns):

- `var1:var20`: all columns from `var1` to `var20`
- `one_of(c("a", "b", "c"))`: all columns with names in the specified character vector of names
- **Removing columns**
 - `-var1`: remove the column `var1`
 - `-(var1:var20)`: remove all columns from `var1` to `var20`
- **Select using text within column names**
 - `contains("date")`, `contains("_")`: all variable names that contain the specified string
 - `starts_with("a")` or `ends_with("last")`: all variable names that start or end with the specified string
- **Rearranging columns**
 - use `everything()` to select all columns not already named
 - example: `select(var1, var20, everything())` moves the column `var20` to the second position

See other examples in the [data wrangling cheatsheet](#).

select() practice

Which columns are selected & in what order using these commands?
First guess and then try them out.

```
demo_data %>% select(record:sex)
demo_data %>% select(one_of(c("age","stweight")))

demo_data %>% select(-grade,-sex)
demo_data %>% select(-(record:sex))

demo_data %>% select(contains("race"))
demo_data %>% select(starts_with("r"))
demo_data %>% select(-contains("r"))

demo_data %>% select(record, race4, race7, everything())
```

rename() ~ columns

- renames column variables

```
demo_data %>% rename(id = record) # order: new_name = old_name
```

```
# A tibble: 20,000 x 8
  id age      sex grade race4    race7     bmi stweight
  <dbl> <chr>    <chr> <chr> <chr>    <chr>    <dbl>    <dbl>
1 931897 15 years o... Fema... 10th  White   White    17.2    54.4
2 333862 17 years o... Fema... 12th  White   White    20.2    57.2
3 36253 18 years o... Male   11th Hispanic/La... Hispanic/La... NA      NA
4 1095530 15 years o... Male   10th Black or Af... Black or Af... 28.0    85.7
5 1303997 14 years o... Male   9th  All other r... Multiple - ... 24.5    66.7
6 261619 17 years o... Male   9th  All other r... <NA>     NA      NA
7 926649 16 years o... Male   11th All other r... Asian    20.5    70.3
8 1309082 17 years o... Male   12th  White   White    19.3    59.0
9 506337 18 years o... Male   12th Hispanic/La... Hispanic/La... 33.1    123.
10 180494 14 years o... Male  10th Black or Af... Black or Af... NA      NA
# ... with 19,990 more rows
```

Practice

Do the following data wrangling steps in order so that the output from the previous step is the input for the next step. Save the results in each step as `newdata`.

1. Import `demo_data.csv` in the `data` folder if you haven't already done so.
2. Subset the data so that only character variables are included and save the result as `newdata`.
3. Filter `newdata` to only keep Asian or Native Hawaiian/other PI subjects that are in the 9th grade, and save again as `newdata`.
4. Filter `newdata` to remove subjects younger than 13, and save as `newdata`.
5. Remove the column `race4`, and save as `newdata`.
6. How many rows does the resulting `newdata` have? How many columns?

Changing the data



Alison Horst

Make New Variables



tidyverse data wrangling cheatsheet

mutate()

Use `mutate()` to add new columns to a tibble

- many options in how to define new column of data

```
newdata <- demo_data %>%  
  mutate(height_m = sqrt(stweight / bmi))  # use = (not <- or ==) to define new variable  
  
newdata %>% select(record, bmi, stweight)
```

```
# A tibble: 20,000 x 3  
  record    bmi stweight  
  <dbl>   <dbl>    <dbl>  
1 931897   17.2     54.4  
2 333862   20.2     57.2  
3 36253     NA       NA  
4 1095530   28.0     85.7  
5 1303997   24.5     66.7  
6 261619     NA       NA  
7 926649   20.5     70.3  
8 1309082   19.3     59.0  
9 506337   33.1    123.  
10 180494    NA       NA
```

mutate() practice

What do the following commands do?

First guess and then try them out.

```
demo_data %>% mutate(bmi_high = (bmi > 30))

demo_data %>% mutate(male = (sex == "Male"))
demo_data %>% mutate(male = 1 * (sex == "Male"))

demo_data %>% mutate(grade_num = as.numeric(str_remove(grade, "th")))
```

case_when() with mutate()

Use `case_when()` to create multi-valued variables that depend on an existing column

- Example: create BMI groups based off of the `bmi` variable

```
demo_data2 <- demo_data %>%
  mutate(
    bmi_group = case_when(
      bmi < 18.5 ~ "underweight",                      # condition ~ new_value
      bmi >= 18.5 & bmi <= 24.9 ~ "normal",
      bmi > 24.9 & bmi <= 29.9 ~ "overweight",
      bmi > 29.9 ~ "obese")
  )
demo_data2 %>% select(bmi, bmi_group) %>% head()
```

```
# A tibble: 6 x 2
  bmi   bmi_group
  <dbl> <chr>
1 17.2 underweight
2 20.2 normal
3 NA   <NA>
4 28.0 overweight
5 24.5 normal
```

separate() and unite()

More commands to filter rows

Remove rows with missing data

`na.omit` removes *all* rows with *any* missing (NA) values in *any* column

```
demo_data %>% na.omit()
```

```
# A tibble: 12,897 x 8
  record age      sex grade race4      race7      bmi stweight
  <dbl> <chr>    <chr> <chr> <chr>    <chr>    <dbl>   <dbl>
1 931897 15 years o... Fema... 10th  White     White     17.2   54.4
2 333862 17 years o... Fema... 12th  White     White     20.2   57.2
3 1095530 15 years o... Male   10th  Black or Af... Black or Af... 28.0   85.7
4 1303997 14 years o... Male   9th   All other r... Multiple - ... 24.5   66.7
5 926649 16 years o... Male   11th  All other r... Asian    20.5   70.3
6 1309082 17 years o... Male   12th  White     White     19.3   59.0
7 506337 18 years o... Male   12th  Hispanic/La... Hispanic/La... 33.1   123.
8 1307180 16 years o... Male   10th  Hispanic/La... Hispanic/La... 21.8   66.7
9 1312128 15 years o... Fema... 10th  White     White     22.0   65.8
10 770177 16 years o... Fema... 10th  White     White     32.4   86.2
# ... with 12,887 more rows
```

We will discuss dealing with missing data more in part 2

Remove rows with duplicated data

`distinct()` removes rows that are duplicates of other rows

```
data_dups <- tibble(  
  name = c("Ana", "Bob", "Cara", "Ana"),  
  race = c("Hispanic", "Other", "White", "Hispanic"))
```

data_dups

```
# A tibble: 4 x 2  
  name   race  
  <chr>  <chr>  
1 Ana    Hispanic  
2 Bob    Other  
3 Cara   White  
4 Ana    Hispanic
```

data_dups %>% `distinct()`

```
# A tibble: 3 x 2  
  name   race  
  <chr>  <chr>  
1 Ana    Hispanic  
2 Bob    Other  
3 Cara   White
```

Order rows: arrange()

Use `arrange()` to order the rows by the values in specified columns

```
demo_data %>% arrange(bmi, stweight) %>% head(n=3)
```

```
# A tibble: 3 x 8
  record age      sex    grade race4       race7      bmi stweight
  <dbl> <chr>   <chr>  <chr> <chr>       <chr>     <dbl>   <dbl>
1 635432 13 years... Female 9th   Hispanic/Lat... Hispanic/Lat... 13.2   27.7
2 501608 15 years... Male   9th   All other ra... Asian        13.2   47.6
3 1097740 16 years... Male   9th   Black or Afr... Black or Afr... 13.3   45.4
```

```
demo_data %>% arrange(desc(bmi), stweight) %>% head(n=3)
```

```
# A tibble: 3 x 8
  record age      sex    grade race4       race7      bmi stweight
  <dbl> <chr>   <chr>  <chr> <chr>       <chr>     <dbl>   <dbl>
1 324452 16 years old Male  11th   Black or Af... Black or Af... 53.9   91.2
2 1310082 18 years ol... Male  11th   Black or Af... Black or Af... 53.5   160.
3 328160 18 years ol... Male  <NA>   Black or Af... Black or Af... 53.4   128.
```

practice slide

more column commands

mutate_

select_*() and rename_*() practice

- variants of `select()` and `rename()` that operate on a selection of columns
- which columns depends on a predicate, can be:
 - a function that returns TRUE/FALSE like `is.numeric()`, or
 - variable names through `vars()`

What do these commands do? Try them out:

```
demo_data %>% select_if(is.numeric)

demo_data %>% rename_all(toupper) # toupper() is a function
demo_data %>% rename_if(is.character, toupper)

demo_data %>% rename_at(vars(contains("race")), toupper)
```

The pipe operator %>% revisited

- a function performed on (usually) a data frame or tibble
- the result is a transformed data set as a **tibble**
- Suppose you want to perform a series of operations on a data.frame or tibble **mydata** using hypothetical functions **f()**, **g()**, **h()**:
 - Perform **f(mydata)**
 - use the output as an argument to **g()**
 - use the output as an argument to **h()**

One option:

```
h(g(f(mydata)))
```

A long tedious option:

```
fout <- f(mydata)
gout <- g(fout)
h(gout)
```

Use the pipe %>%

Instead, we can use the pipe operator (pronounced "then")

- Take `mydata` then
- perform `f()` then
- perform `g()` then
- perform `h()`

```
mydata %>%
  f() %>%
  g() %>%
  h()
```

Why use the pipe?

- makes code more readable
- `h(f(g(mydata)))` can get complicated with multiple arguments
 - i.e. `h(f(g(mydata, na.rm=T), print=FALSE), type = "mean")`

A real example:

```
mydata_new <- mydata_tib %>% select(id, weight_kg, bmi) %>%  
  mutate(height_m = sqrt(weight_kg /bmi))  
mydata_new %>% head(n=3)
```

```
# A tibble: 3 x 4  
  id   weight_kg   bmi height_m  
  <dbl>     <dbl> <dbl>     <dbl>  
1 335340      66.2  27.6     1.55  
2 638618      84.8  29.3     1.70  
3 922382      57.6  18.2     1.78
```

Resources - Tidyverse & Data Wrangling

Links

- [Learn the tidyverse](#)
- [Data wrangling cheatsheet](#)

Some of this is drawn from materials in online books/lessons:

- [R for Data Science](#) - by Garrett Grolemund & Hadley Wickham
- [Modern Dive](#) - An Introduction to Statistical and Data Sciences via R by Chester Ismay & Albert Kim
- [A gradual introduction to the tidyverse](#) - Workshop for Cascadia R 2017 by Chester Ismay and Ted Laderas
- [Cookbook for R](#) by Winston Chang
- ["Tidy Data"](#) by Hadley Wickham

Possible Future Workshop Topics?

- reproducible reports in R
- tables
- ggplot2 visualization
- advanced tidyverse: functions, purrr
- statistical modeling in R

Contact info:

Jessica Minnier: minnier@ohsu.edu

Meike Niederhausen: niederha@ohsu.edu

This workshop info:

- Code for these slides on github: [jminnier/berd_r_courses](https://github.com/jminnier/berd_r_courses)
- all the R code in an R script
- answers to practice problems can be found here: [html](#), [pdf](#)