Data Wrangling in R with the Tidyverse

Learning objectives

Part 1:

- What is data wrangling?
- A few good practices in R/RStudio What is tidy
- data?
- What is tidyverse? Manipulate data
-

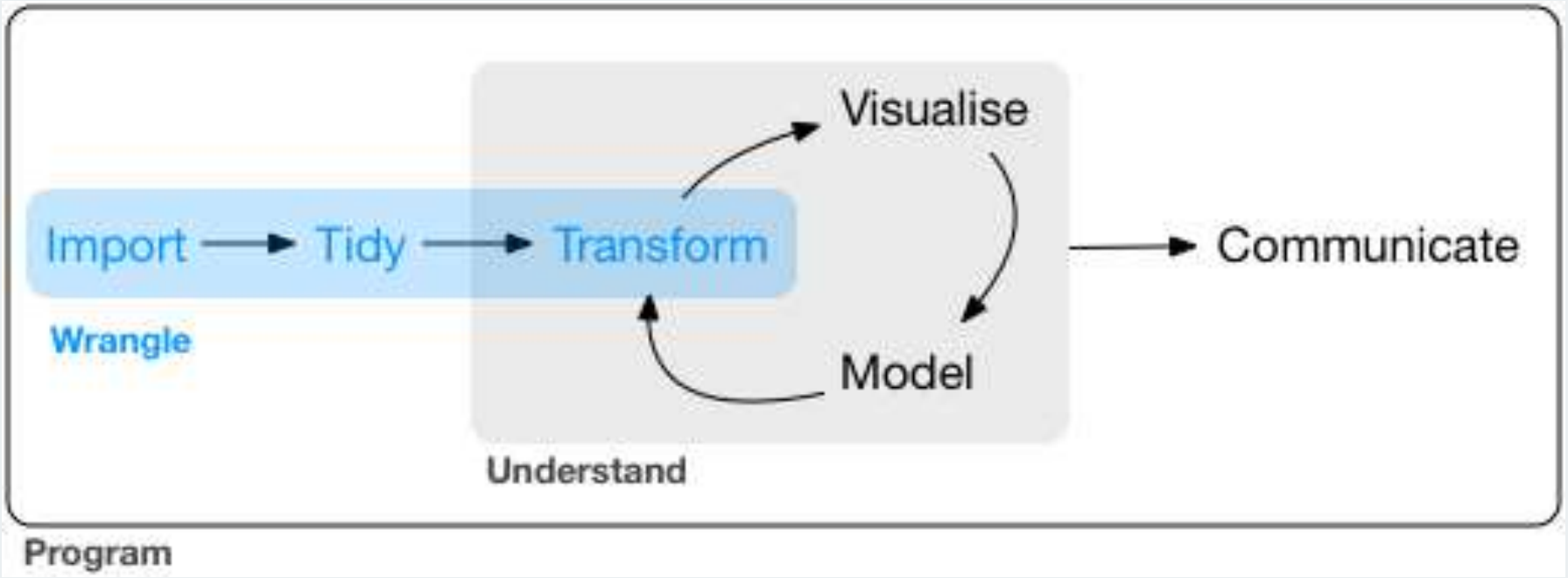
Part 2:

- Reshaping (long/wide format) data Join/merge
- data sets
- Data cleaning, including examples for dealing with:
- Missing data
  Strings/character vectors Factors/categorical
  - variables Dates
  -
  -

Getting started

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



G. Grolemond & H. Wickham's R for Data Science

Useful keyboard shortcuts

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

Try typing (with shortcut) and running

```
y <- 5
y
```

Now, in the console, press the up arrow.

Others: (see full 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

Tibbles

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
```

How are these two datasets different?

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
   nam    rank    age   city
   e
   <ch    <int   <dbl   <chr>
   r>        >      >
1  Sar       1   35.5   <NA>
   ah
2  Ana       2    25    New
                        York
3  Jos       3    58    LA
   e
```

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	grade	race4
1	335340	17 years old	10th	White
2	638618	16 years old	9th	<NA>
3	922382	14 years old	9th	White
4	923122	15 years old	9th	White
5	923963	15 years old	10th	Black or African American

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    gra    rac     bmi  weight_k  text_while_dri
                         de     e4                 g  v…
    <dbl>  <ch    <ch    <ch    <ch    <dbl      <dbl>  <chr>
           r>     r>     r>     r>        >
 1 3.35e5  1   y  Fem    10t    Whi    27.6       66.2  <NA>
           7   …  a…     h      te
 2 6.39e5  1   y  Fem    9th    <NA    29.3       84.8  <NA>
           6   …  a…           >
 3 9.22e5  1   y  Mal    9th    Whi    18.2       57.6  <NA>
           4   …  e           te
 4 9.23e5  1   y  Mal    9th    Whi    21.4       60.3  <NA>
           5   …  e           te
 5 9.24e5  1   y  Mal    10t    Bla    19.6       63.5  <NA>
           5   …  e      h      c…
 6 9.26e5  1   y  Mal    10t    All    22.2       70.3  <NA>
```

Compare & contrast data frame and tibble

Run the code below

data frame

```
glimpse(mydata_df)

str(mydata_df)       #    How are glimpse()    str()
                                  and           different?

head(mydata_df)

summary(mydata_df)

class(mydata_df)     #    What information     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

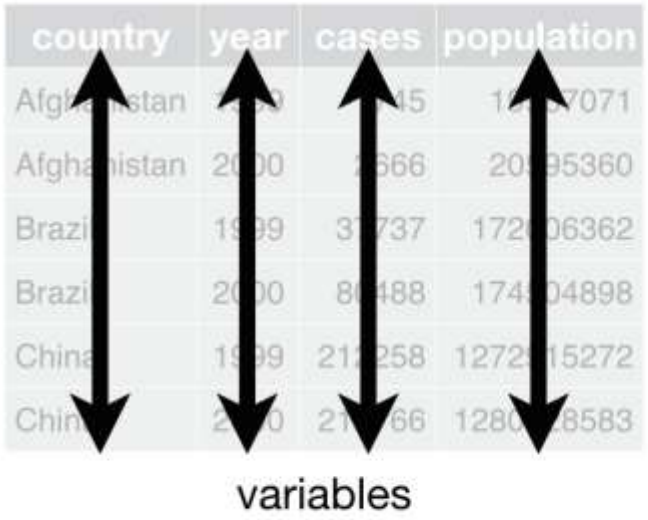
Other perks:

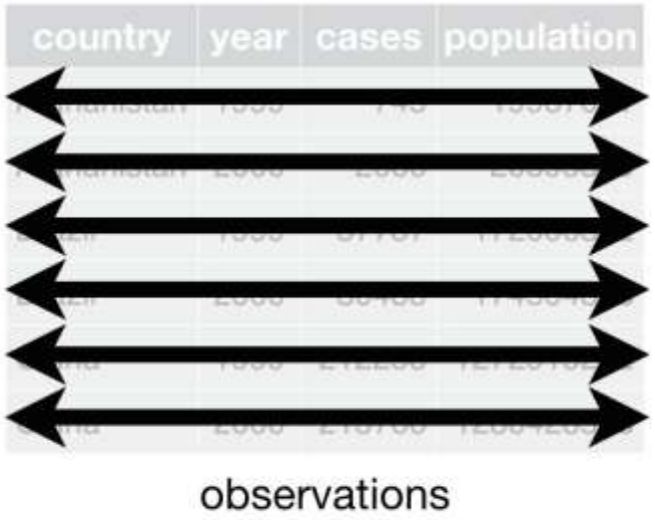
- tibbles can typically be used anywhere a `data.frame` is needed
- `read_*()` functions don't read character columns as factors (no surprises)

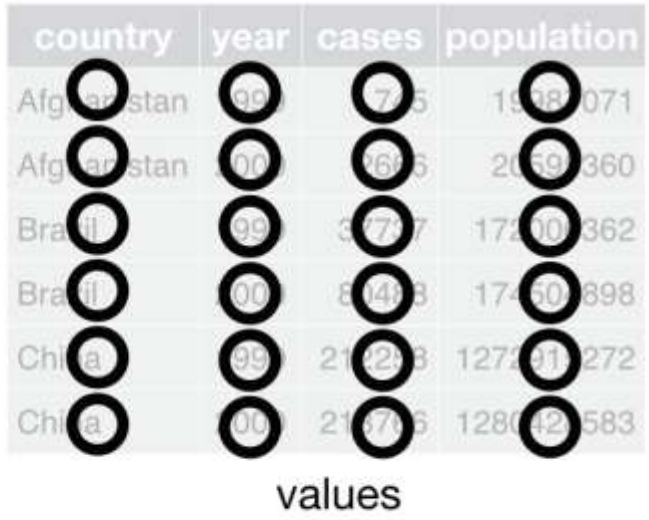
Tidy Data

What are tidy data?

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







G. Grolemond & H. Wickham's R for Data Science

Untidy data: example 1

```
untidy_data <- tibble(
  name = c("Ana","Bob","Cara"),
  meds = c("advil 600mg 2xday","tylenol 650mg 4xday", "advil 200mg 3xday")
)
untidy_data
```

```
# A tibble: 3 x 2 name  meds
<chr> <chr>
1 Ana   advil 600mg 2xday
2 Bob   tylenol 650mg 4xday
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           600         2
 2 Bob   tylenol         650         4
 3       advil           200         3
Cara
```

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
```

How to tidy?

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> <chr> <dbl>     <dbl> <chr>
   <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                  smoked_ever <chr>, bullied_past_12mo
# variables:
# physical_fights
```

- 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
  -

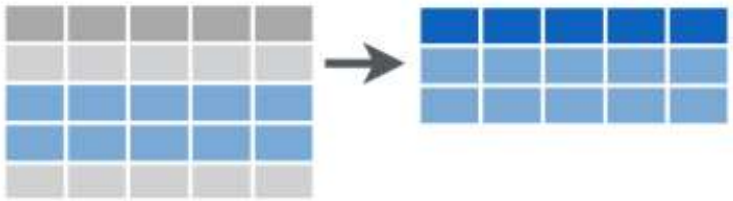
  the data in `yrbss_demo.csv` are a subset of data in the R package `yrbss`, which includes
- YRBSS from 1991-2013

  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)

[tidyverse data wrangling cheatsheet]

`filter()` ~ rows

filter data based on rows

- math: `>`, `<`, `>=`, `<=`
- double `=` for "is equal to": `==`
- `&` (and)
- `|` (or)
- `!=` (not equal)

- `is.na()` to filter based on missing values
- `%in%` to filter based on group membership
- `!` in front negates the statement, as in
  - `!is.na(age)`
  - `!(grade %in% c("9th","10th"))`

```
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… Female 12th   White  White          20.2      57.2
 2 1307188 16 years o… Male   10th   Black  Black or Af…    21.8      66.7
 3 1095390 15 years o… Male   10th   Hispa… Hispanic/La…    23.0      85.7
 4 1312128 15 years o… Female 10th   White  White           22.0      65.8
```

Compare to base R

- **Bracket method**: need to repeat tibble name
- Need to use `$`
- Very nested and confusing to read
- Keeps `NA`s

```
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
    2  261619 17    years… Male  9th
    3 1096939 15    years… Male  9th
    4  180968 15    years… Male  9th
    5  924270 15    years… Male  9th
    6  330828 15    years… Fema… 9th
```

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

```
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
   2  261619 17    years… Male  9th
   3 1096939 15    years… Male  9th
   4  180968 15    years… Male  9th
   5  924270 15    years… Male  9th
   6  330828 15    years… Female 9th
   7 1310252 15    years… Female 9th
   8  246830 14    years… Female 9th
   9 1310689 14    years… Female 9th
```

`filter()` practice

What do these commands do? Try them out:

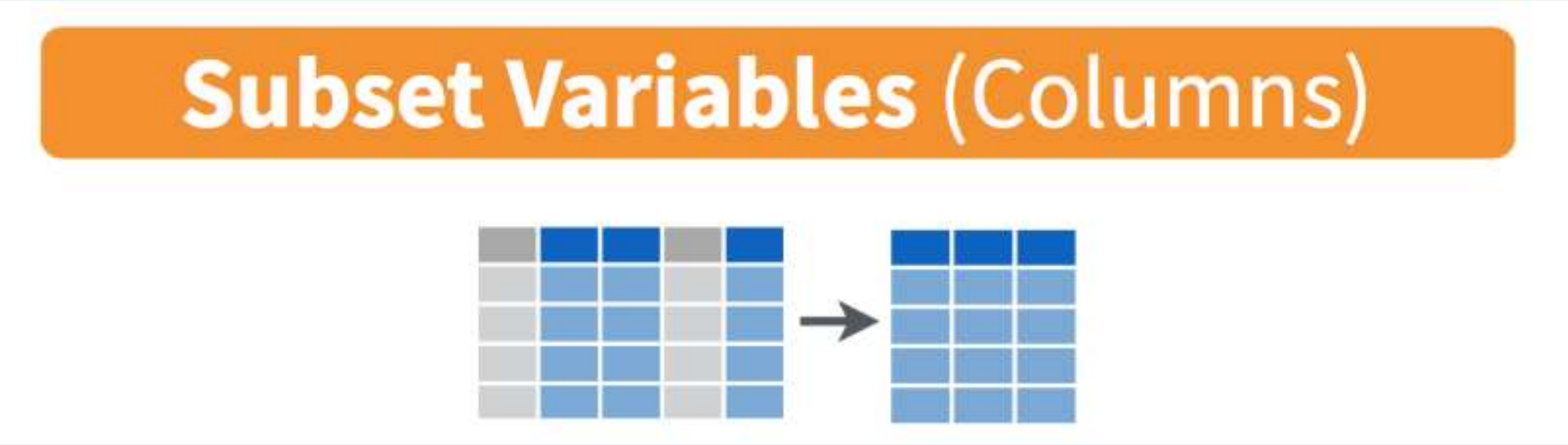
```
demo_da    %    filter(bmi < 5)
ta         >    filter(bmi/stweight < 0.5)    #  can do math
demo_da    %
ta         %
           >
           %

demo_da    %    filter((bmi < 15) | (bmi >
ta         >    50))                                sex == "Male") # filter on multiple
demo_da    %    filter(bmi < 20, stweight <         variables
ta         %    50,
           >
           %

demo_da    %    filter(record == 506901)      #  note the use of == instead of just =
ta         >    filter(sex == "Female")
demo_da    %
ta         %
           >
           %

demo_dat   %>   filter(!(grade == "9th"))
   a       %
```

Subset by 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
 <dbl> <chr>                      sex
 1  931897 15 years old           <chr>
 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 specificed 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_dat  %>  select(record:sex)
    a     %
demo_dat  %>  select(one_of(c("age","stweight")))
    a     %
demo_dat  %>  select(-grade,-sex)
    a     %
demo_dat  %>  select(-(record:sex))
    a     %
demo_dat  %>  select(contains("race"))
    a     %
demo_dat  %>  select(starts_with("r"))
    a     %
demo_dat  %>  select(-contains("r"))
    a     %
demo_dat  %>  select(record, race4, race7,
    a     %   everything())
```

rename() ~ columns

- renames column variables

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

```
# A tibble: 20,000 x 8
        id age          sex                race4       race7           bmi                 stweight
                                                                                            <dbl>
                                                                                                1
                                                                                                2
                                                                                                7
                                                                                                7
                                                                                                3
                                                                                                )
  # … with
```

Practice

```
# Remember: to save output into the same tibble you would use <- newdata <- newdata %>%
select(-record)

# Useful to see what categories are available demo_data %>% janitor::tabyl(race7)
```

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. Filter `newdata` to only keep "Asian" or "Native Hawaiian/other PI" subjects that are in the 9th grade, and save again as `newdata`.
3. Filter `newdata` to remove subjects younger than 13, and save as `newdata`.
4. Remove the column `race4`, and save as `newdata`.
5. 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
```

`mutate()` practice

What do the following commands do?
First guess and then try them out.

```
demo_dat %>% mutate(bmi_high = (bmi > 30))
    a      %
demo_dat %>% mutate(male = (sex == "Male"))
    a      %
demo_dat %>% mutate(male = 1 * (sex == "Male"))
    a      %
demo_dat %>% mutate(grade_num = as.numeric(str_remove(grade,
    a      %  "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()`

`separate()`: one column to many

- when one column has multiple types of information
- removes original column by default

```
demo_data %>%
  separate(age,c("a","y","o","w","w2"),
           sep = " ") %>% select(a:w2)
```

```
# A tibble: 20,000 x 5
   a     y     o     w     w2
   <chr> <chr> <chr> <chr> <chr>
 1 15    years old   <NA>  <NA>
                     >
 2 17    years old   <NA>  <NA>
                     >
 3 18    years old   or    older
 4 15    years old   <NA>  <NA>
```

`unite()`: many columns to one

- paste columns together using a separator
- removes original columns by default

```
demo_data %>%
  unite("sexgr", sex, grade, sep=":") %>%
  select(sexgr)
```

```
# A tibble: 20,000 x 1
   sexgr
<chr>
1 Female:10th
  2
  Female:12th
  3 Male:11th

  4 Male:10th

  5 Male:9th
```

`separate()` and `unite()` practice

What do the following commands do? First guess and then try them out.

```
demo_dat  %>%  separate(age, c("agenum","yrs"), sep = " ")
       a  %%
demo_dat  %>%  separate(age, c("agenum","yrs"), sep = " ", remove = FALSE)
       a  %%
demo_dat  %>%  separate(grade, c("grade_n"), sep = "th")
       a  %%
demo_dat  %>%  separate(grade, c("grade_n"), sep = "t")
       a  %%
demo_dat  %>%  separate(race4, c("race4_1", "race4_2"), sep = "/")
       a  %%
demo_dat  %>%  unite("sex_grade", sex, grade, sep = "::::")
       a  %%
demo_dat  %>%  unite("sex_grade", sex, grade)        # what is the default `sep` for unite?
       a  %%
demo_dat  %>%  unite("race", race4, race7)           # what happens to NA values?
       a  %%
```

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 …   Fema…  10th   White  White         17.2      54.4
 2 333862  17 years …   Fema…  12th   White  White         20.2      57.2
 3 1095530 15 years …   Male   10th   Black… Black…        28.0      85.7
 4 1303997 14 years …   Male   9th    All o… Multiple -…   24.5      66.7
 5 926648  16 years …   Male   11th   All o… Asian         20.5      70.3
 6 1309082 17 years …   Male   12th   White  White         19.3      59.0
# … 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/La… Hispanic/La…  13.2      27.7
2  501808 15 years… Male   9th   All other ra… Asian          13.2      47.6
```

```
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… o Af  53.9      91.2
```

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. Create a variable called `grade_num` that has the numeric grade number (use `as.numeric()`).
3. Filter the data to keep only students in grade 11 or higher.
4. Filter out rows when `bmi` is `NA`.
5. Create a binary variable called `bmi_normal` that is equal to 1 when `bmi` is between 18.5 to 24.9 and 0 when it is outside that range.
6. Arrange by `grade_num` from highest to lowest
7. Save all output to `newdata`.

Advanced column commands

Mutating multiple columns at once: `mutate_*`

- variants of `mutate()` that are useful for mutating multiple columns at once
  - `mutate_at()`, `mutate_if()`, `mutate_all()`, etc.
- which columns get mutated 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:

```
# mutate_if
demo_data %>% mutate_if(is.numeric, as.character)  # as.character() is a function
demo_data %>% mutate_if(is.double, round, digits=0) # arguments to function can go after
demo_data %>% mutate_if(is.character, tolower)      # tolower() is a function
# mutate_at
demo_data %>% mutate_at(vars(age:grade), toupper)  # toupper() is a function demo_data
%>% mutate_at(vars(bmi,stweight), log)
demo_data %>% mutate_at(vars(contains("race")), str_detect, pattern = "White")

# mutate_all
demo_data %>% mutate_all(as.character)
```

Selecting & renaming multiple columns

- `select_*()` & `rename_*()` are variants of `select()` and `rename()`
- use like `mutate_*()` options on previous slide

What do these commands do? Try them out:

```
demo_dat  %>  select_if(is.numeric)
    a     %
demo_dat  %>  rename_all(toupper)
    a     %
demo_dat  %>  rename_if(is.character, toupper)
    a     %
demo_dat  %>  rename_at(vars(contains("race")),
    a     %   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()`: `g(f(mydata))`
  - use the output as an argument to `h()`: `h(g(f(mydata)))`

One option:

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

A long tedious option:

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

Using pipes - easier to read:

```
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")`

tidyverse way:

```
demo_data2 <- demo_data %>%
  na.omit %>%
  mutate(
    height_m = sqrt(stweight/bmi),
    bmi_high = 1*(bmi>30)
  ) %>%
  select_if(is.numeric)
demo_data2
```

base R way:

```
demo_data3 <- na.omit(demo_data)
demo_data3$height_m <- sqrt(demo_data3$stweight/demo
demo_data3$bmi_high <- 1*(demo_data3$bmi>30)
demo_data3 <- demo_data3[,c("record","bmi","stweight
demo_data3
```

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
- "Tidy Data" by Hadley Wickham