Day 3 Part 3

~~Day 4~~ BSTA 511/611 Fall 202~~2~~3

Some more data wrangling - reference slides

Meike Niederhausen, PhD

Recap of last time

Day 3 focused on EDA (exploratory data analysis)

- First look at dataset (dimensions, data types, column names)

- Data visualizations with ggplot

  - numerical data
  - categorical data
  - relationships between multiple variables

- Summarizing data

  - numerical:
    - summary statistics such as mean, sd, median, IQR
  - categorical:
    - frequency tables, proportions (relative frequencies)

Tools for wrangling data

- `tidyverse` functions

    - `tidyverse` is a suite of packages that implement `tidy` methods for data importing, cleaning, wrangling, and visualizing
    - load the `tidyverse` packages by running the code `library(tidyverse)`
        - Don't forget to first install `tidyverse`!

- Functions to easily work with rows and columns, such as

    - subset rows/columns
    - add new rows/columns
    - join together different data sets
    - make data *long* or *wide*

- There are often many steps to tidy data

    - we string together commands
    - to be performed sequentially
    - using pipes `%>%`

Summary of data wrangling so far

- The pipe `%>%` to string together commands in sequence
- `mutate()` to add a new variable to a dataset
- `select()` to select columns (or deselect columns with -variable)
- `filter()` to select specific rows
- `pivot_wider()` to reshape a dataset from a long to a wide format

Summarizing data

- `tabyl()` from `janitor` package to make frequency tables of categorical variables
- `summarize()` to get summary statistics of variables
- `group_by()` to group data by categorical variables before finding summaries

Goals for today

More data wrangling examples and tools

- Subsetting data
  - `filter`ing rows
  - `select`ing columns
- More wrangling for columns:
  - `relocate` columns
  - `rename` columns
- Creating new variables
  - `mutate`
- Reshaping data
  - wide vs long data
    - make wide data long
    - make long data wide

Case study: discrimination in developmental disability support (1.7.1)

- In the US, individuals with developmental disabilities typically receive services and support from state governments.
  - California allocates funds to developmentally disabled residents through the Department of Developmental Services (DDS)
  - Recipients of DDS funds are referred to as "consumers."
- Dataset `dds.discr`
  - sample of 1,000 DDS consumers (out of a total of ~ 250,000)
  - age, gender, race/ethnicity, and DDS annual financial support per consumer
- **Previous research**
  - Researchers examined expenditures on consumers by ethnicity
  - Found that the mean annual expenditures on Hispanics was less than that on White non-Hispanics.
- Result: an allegation of ethnic discrimination was brought against the California DDS.

- **Question: Are the data sufficient evidence of ethnic discrimination?**

- See Section 1.7.1 for more details

Load dataset `dds.discr` from package `oibiostat`

- The textbook's datasets are in the R package `oibiostat`

- If you haven't already installed the package `oibiostat`, then first do so using directions in previous slide.

- Load the `oibiostat` package and the dataset `dds.discr`

    - the code below needs to be run *every time* you restart R or knit an Rmd file

```r
library(oibiostat)
data("dds.discr")
```

- After loading the dataset `dds.discr` using `data("dds.discr")`, you will see `dds.discr` in the Data list of the Environment window.

Getting to know the dataset

```
dim(dds.discr)
```

```
## [1] 1000    6
```

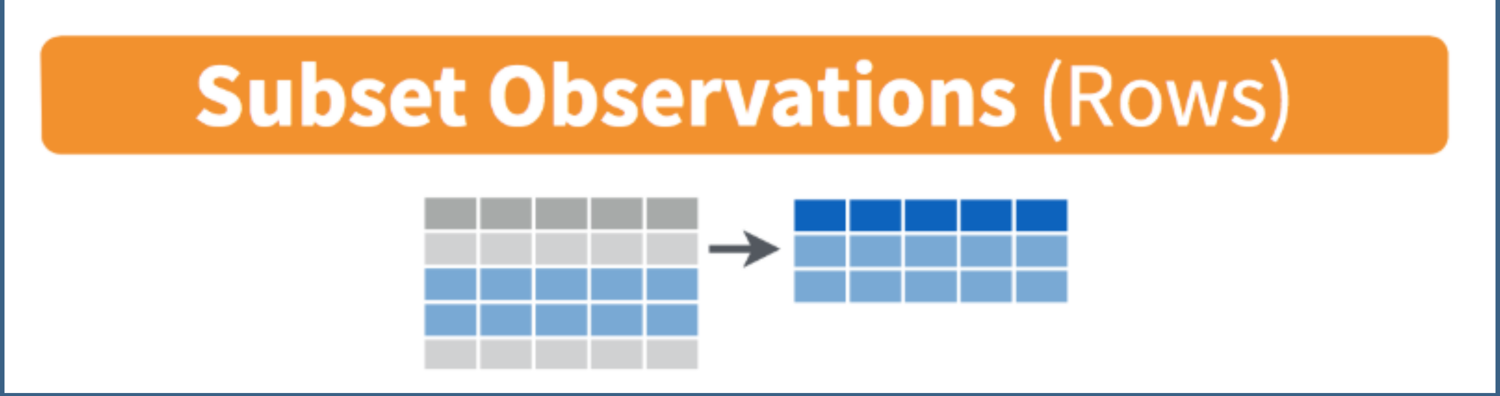
```
names(dds.discr)
```

```
## [1] "id"           "age.cohort"   "age"          "gender"       "expenditures"
## [6] "ethnicity"
```

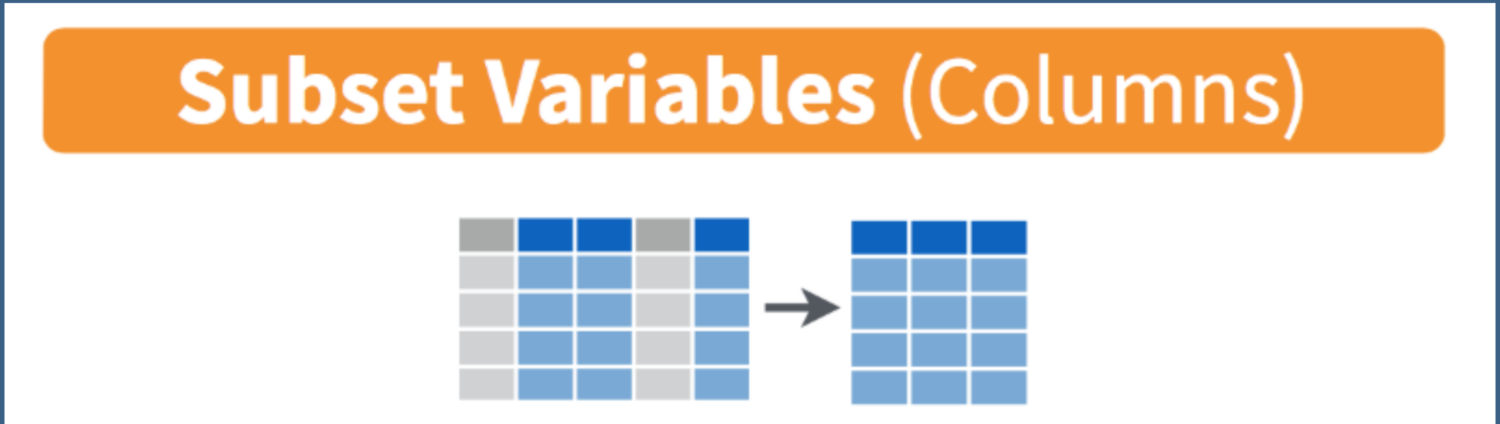
```
length(unique(dds.discr$id)) # How many unique id's are there?
```

```
## [1] 1000
```

Subsetting data



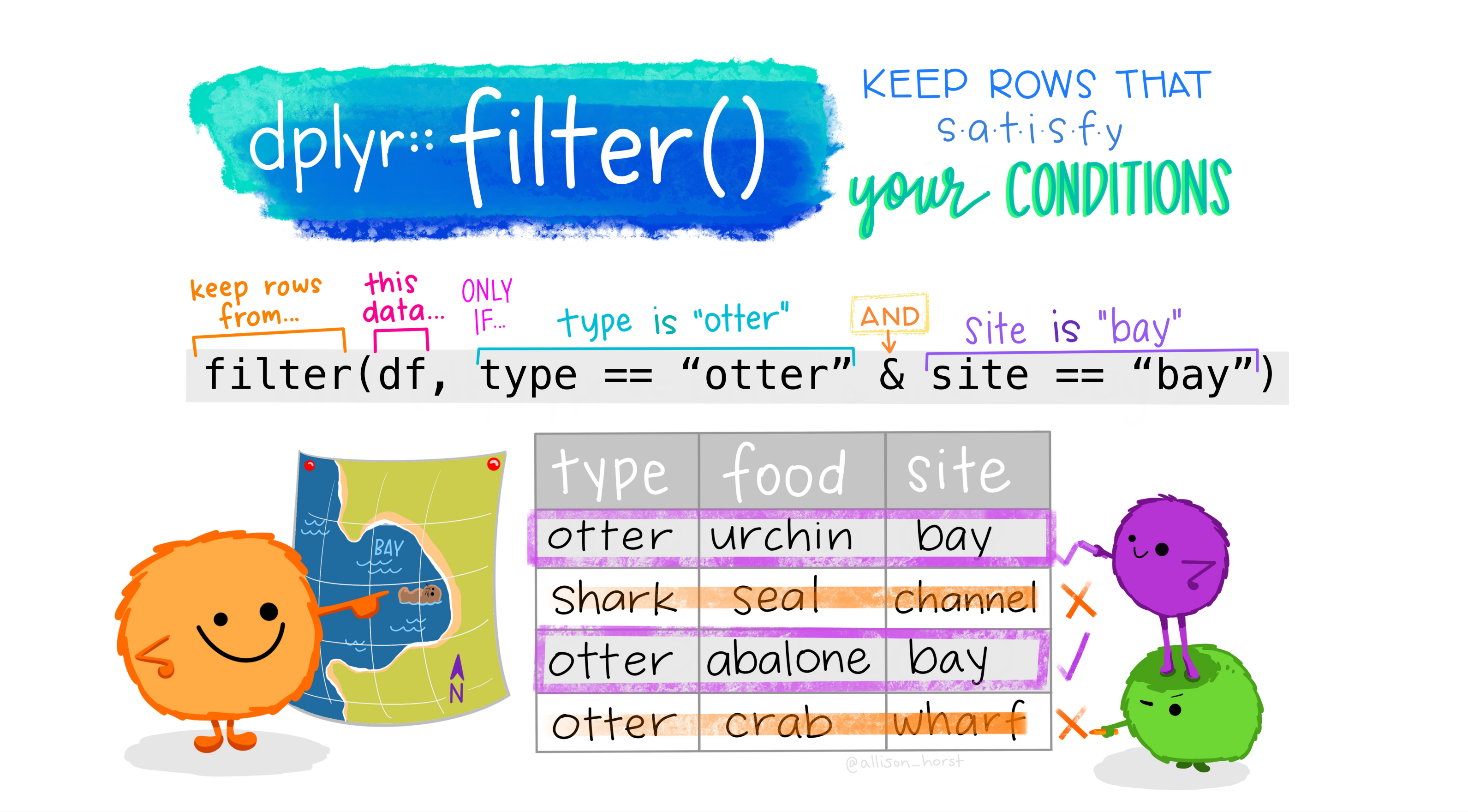
Subset Observations (Rows)



Subset Variables (Columns)

data transformation cheatsheet

`filter()` rows that satisfy specified conditions



```
filter(df, type == "otter" & site == "bay")
```

type	food	site
otter	urchin	bay
shark	seal	channel
otter	abalone	bay
otter	crab	wharf

Allison Horst

`filter()` to select rows

filter data based on rows

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

- `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)`
  - `!(ethnicity %in% c("Asian","Black"))`

```
# Note: the output from the command below is not being saved
# since it's not being assigned to a variable using <-
dds.discr %>% filter(age > 90)
```

```
## # A tibble: 4 × 6
##      id age.cohort   age gender expenditures ethnicity
##   <int> <fct>      <int> <fct>         <int> <fct>
## 1 19250 51+           94 Female        60871 Hispanic
## 2 46726 51+           95 Male          55187 Hispanic
## 3 55056 51+           95 Female        54680 Black
## 4 87737 51+           91 Male          54481 Asian
```

filter() practice

What do these commands do? Try them out:

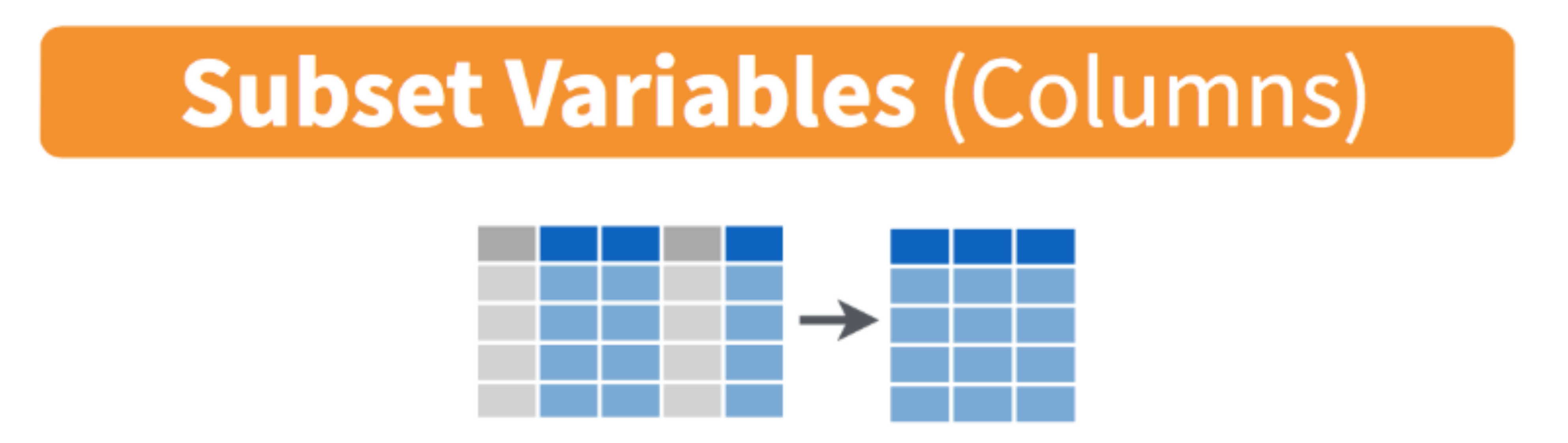
```
dds.discr %>% filter(age < 5)
dds.discr %>% filter(age/expenditures < 0.5)    # can do math within filter comman
dds.discr %>% filter((age < 15) | (age > 50))

# simultaneously filter on multiple variables
dds.discr %>% filter(age < 20, expenditures > 1000, gender == "Male")

dds.discr %>% filter(id == 10210) # note the use of == instead of just =
dds.discr %>% filter(gender == "Female")
dds.discr %>% filter(!(age.cohort == "51+"))
dds.discr %>% filter(age.cohort %in% c("0-5", "6-12"))

dds.discr %>% filter(is.na(age))
dds.discr %>% filter(!is.na(age))
```

Subset by columns



data transformation cheatsheet

`select()` to choose 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

```
dds.discr %>% select(id, expenditures, ethnicity)
```

```
## # A tibble: 1,000 × 3
##       id expenditures ethnicity
##    <int>        <int> <fct>
##  1 10210         2113 White not Hispanic
##  2 10409        41924 White not Hispanic
##  3 10486         1454 Hispanic
##  4 10538         6400 Hispanic
##  5 10568         4412 White not Hispanic
##  6 10690         4566 Hispanic
##  7 10711         3915 White not Hispanic
##  8 10778         3873 Black
##  9 10820         5021 White not Hispanic
## 10 10823         2887 Hispanic
## # … with 990 more rows
```

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 by specifying text within column names**
  - `contains("date")`, `contains("_")`: all variable names that contain the specified string of characters
  - `starts_with("a")` or `ends_with("last")`: all variable names that start or end with the specified string

See other examples in the data transformation cheatsheet.

select() practice

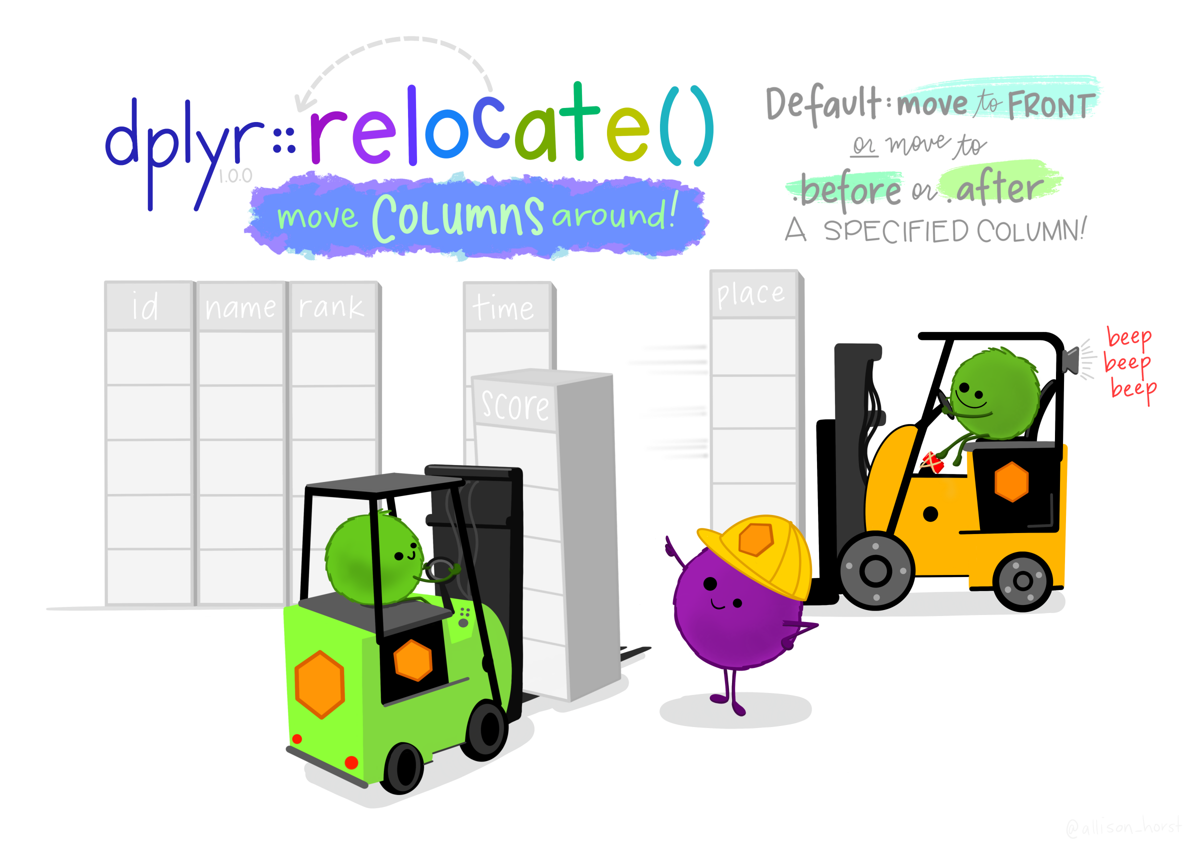
Which columns are selected & in what order using these commands?
First guess and then try them out.

```r
dds.discr %>% select(id:gender)
dds.discr %>% select(one_of(c("age","expenditures", "notindata")))

dds.discr %>% select(-age.cohort,-gender)
dds.discr %>% select(-(id:gender))

dds.discr %>% select(contains("age"))
dds.discr %>% select(starts_with("a"))
dds.discr %>% select(-contains("a"))
```

relocate() columns to move them around



Allison Horst

`relocate()` to change order of columns

- change the order of columns in dataset
- specified column names get put first,
  - and unspecified column names after that in original order
- no quotes needed around variable names
- similar options as with `select()`,
  - plus special ones such as `.before` and `.after`

```
dds.discr %>% relocate(age.cohort, ethnicity)
```

```
## # A tibble: 1,000 × 6
##    age.cohort ethnicity                id   age gender expenditures
##    <fct>      <fct>                 <int> <int> <fct>         <int>
##  1 13-17      White not Hispanic 10210    17 Female         2113
##  2 22-50      White not Hispanic 10409    37 Male          41924
##  3 0-5        Hispanic              10486     3 Male           1454
##  4 18-21      Hispanic              10538    19 Female         6400
##  5 13-17      White not Hispanic 10568    13 Male           4412
##  6 13-17      Hispanic              10690    15 Female         4566
##  7 13-17      White not Hispanic 10711    13 Female         3915
##  8 13-17      Black                 10778    17 Male           3873
##  9 13-17      White not Hispanic 10820    14 Female         5021
```

relocate() practice

What order are the columns in using these commands?
First guess and then try them out.

```
dds.discr %>% relocate(age:ethnicity)

dds.discr %>% relocate(where(is.numeric))
dds.discr %>% relocate(where(is.factor))
# note: the next command doesn't do anything
# since there are no character type variables in the dataset
dds.discr %>% relocate(where(is.character))

dds.discr %>% relocate(age,.before = ethnicity)
dds.discr %>% relocate(ethnicity, .after = age.cohort)
dds.discr %>% relocate(age, .after = last_col())
```

rename() columns

- renames column variables

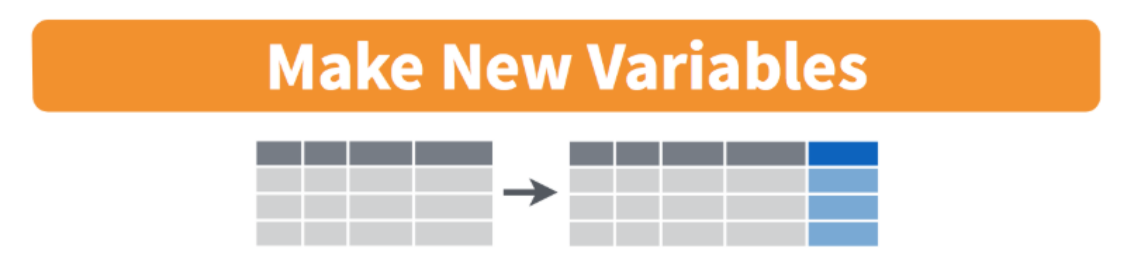
```
dds.discr %>% rename(IDnumber = id)    # order: new_name = old_name
```

```
## # A tibble: 1,000 × 6
##    IDnumber age.cohort    age gender expenditures ethnicity
##       <int> <fct>       <int> <fct>         <int> <fct>
##  1    10210 13-17          17 Female         2113 White not Hispanic
##  2    10409 22-50          37 Male          41924 White not Hispanic
##  3    10486 0-5             3 Male           1454 Hispanic
##  4    10538 18-21          19 Female         6400 Hispanic
##  5    10568 13-17          13 Male           4412 White not Hispanic
##  6    10690 13-17          15 Female         4566 Hispanic
##  7    10711 13-17          13 Female         3915 White not Hispanic
##  8    10778 13-17          17 Male           3873 Black
##  9    10820 13-17          14 Female         5021 White not Hispanic
## 10    10823 13-17          13 Male           2887 Hispanic
## # … with 990 more rows
```

Make new variables

mutate:
add column(s),
keep existing.
A
1
5
10
0
B
2
-3
4
9
C
3
2
14
9
8
THE PLAN:
A B C
+ =
Horst '18

Alison Horst



data transformation cheatsheet

mutate()

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

- Many options in how to define new column of data

```
# use = to define new a variable within mutate (not <- or ==)
newdata <- dds.discr %>%
  mutate(
    log_expenditures = log(expenditures),
    expend_per_yearage = expenditures / age)


newdata %>% select(id, age, expenditures, log_expenditures, expend_per_yearage)
```

```
## # A tibble: 1,000 × 5
##        id   age expenditures log_expenditures expend_per_yearage
##     <int> <int>        <int>            <dbl>              <dbl>
##   1 10210    17         2113             7.66               124.
##   2 10409    37        41924            10.6               1133.
##   3 10486     3         1454             7.28               485.
##   4 10538    19         6400             8.76               337.
##   5 10568    13         4412             8.39               339.
##   6 10690    15         4566             8.43               304.
```

mutate() practice

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

```
dds.discr %>% mutate(age_young = (age < 18))

dds.discr %>% mutate(male = (gender == "Male"))
dds.discr %>% mutate(male = 1 * (gender == "Male"))
```

case_when() to create multi-valued variables

- Example: create age groups based off of the age variable

```
dds.discr2 <- dds.discr %>%
  mutate(
    age_group = case_when(
      age < 6 ~ "0 to 5",                # condition ~ new_value
      age >= 6 & age < 13 ~ "6 to 12",
      age >= 13 & age < 18 ~ "13 to 17",
      age >= 18 & age < 22 ~ "18 to 21",
      age >= 22 & age < 51 ~ "22 to 50",
      age >= 51 ~ "51 plus")
    )

dds.discr2 %>% select(age, age_group) %>% head()
```

```
## # A tibble: 6 × 2
##     age age_group
##   <int> <chr>
## 1    17 13 to 17
## 2    37 22 to 50
## 3     3 0 to 5
## 4    19 18 to 21
```

arrange() to order rows

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

```
dds.discr %>% arrange(age, expenditures) %>% head(n=3)
```

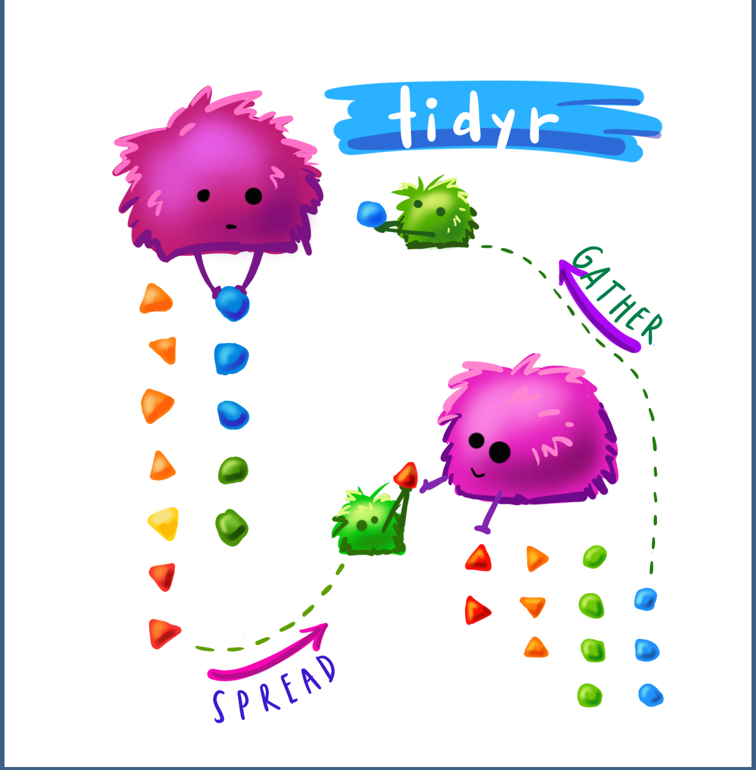
```
## # A tibble: 3 × 6
##        id age.cohort   age gender expenditures ethnicity
##     <int> <fct>      <int> <fct>         <int> <fct>
## 1 39131 0-5            0 Female          685 White not Hispanic
## 2 25613 0-5            0 Male            741 Hispanic
## 3 19917 0-5            0 Male            904 White not Hispanic
```

```
dds.discr %>% arrange(desc(age), expenditures) %>% head(n=3)
```

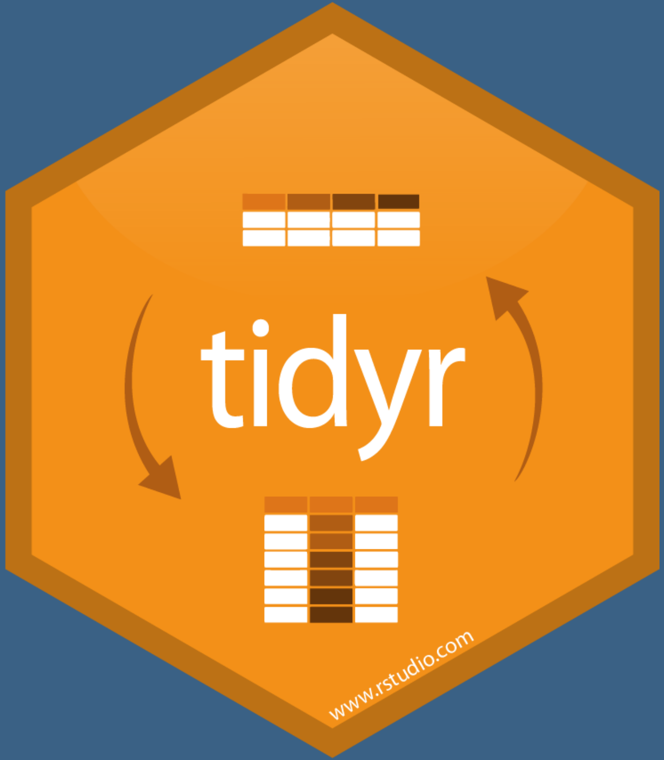
```
## # A tibble: 3 × 6
##        id age.cohort   age gender expenditures ethnicity
##     <int> <fct>      <int> <fct>         <int> <fct>
## 1 55056 51+           95 Female        54680 Black
## 2 46726 51+           95 Male          55187 Hispanic
## 3 19250 51+           94 Female        60871 Hispanic
```

Reshaping data

wide vs. long data



Allison Horst

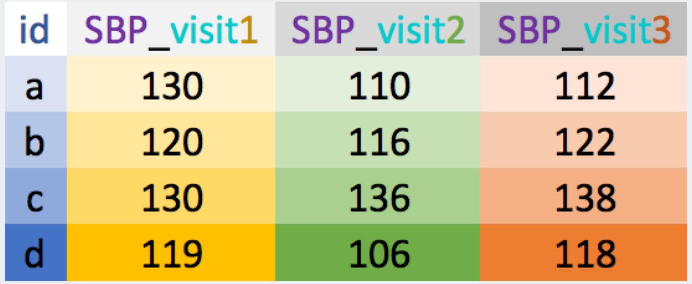


tidyr

Wide vs. long data

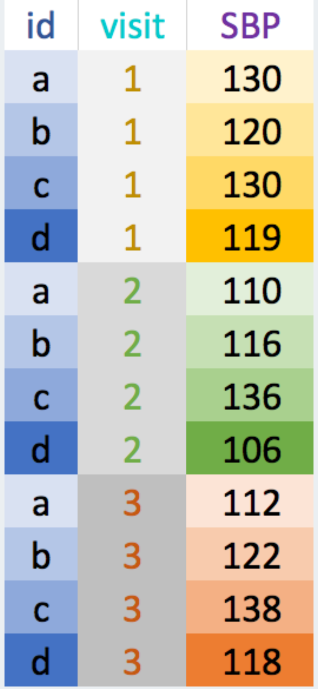
- **Wide** data has one row per individual,
  - with multiple columns for their repeated measurements
- **Long** data has multiple rows per individual,
  - with one column for the measurement variable and
  - another indicating from when/where the repeated measures are from

wide



id	SBP_visit1	SBP_visit2	SBP_visit3
a	130	110	112
b	120	116	122
c	130	136	138
d	119	106	118

long



id	visit	SBP
a	1	130
b	1	120
c	1	130
d	1	119
a	2	110
b	2	116
c	2	136
d	2	106
a	3	112
b	3	122
c	3	138
d	3	118

DDS example

Mean expenditures by ethnicity and age cohort (from Day 3 slides 57-58)

`mean_expend` dataset is "long"

```
mean_expend <-
  dds.discr_Hips_WhnH %>%
  group_by(
    ethnicity, age.cohort)%>%
  summarize(
    ave = mean(expenditures))
```

`mean_expend_wide` is "wide"

```
mean_expend_wide <-
  mean_expend %>%
  pivot_wider(
    names_from = ethnicity,
    values_from = ave)
```

```
mean_expend_wide
```

```
## # A tibble: 6 x 3
##   age.cohort Hispanic `White not Hispanic`
##   <fct>         <dbl>                <dbl>
## 1 0-5           1393.                1367.
## 2 6-12          2312.                2052.
## 3 13-17         3955.                3904.
## 4 18-21         9960.               10133.
## 5 22-50        40924.               40188.
## 6 51+          55585                52670.
```

Example wide toy dataset

Copy and paste the code below into R to create this example dataset

```
SBP_wide <- tibble(id = letters[1:4],
                     sex = c("F", "M", "M", "F"),
                     SBP_v1 = c(130, 120, 130, 119),
                     SBP_v2 = c(110, 116, 136, 106),
                     SBP_v3 = c(112, 122, 138, 118))
SBP_wide
```

```
## # A tibble: 4 × 5
##   id    sex   SBP_v1 SBP_v2 SBP_v3
##   <chr> <chr>  <dbl>  <dbl>  <dbl>
## 1 a     F        130    110    112
## 2 b     M        120    116    122
## 3 c     M        130    136    138
## 4 d     F        119    106    118
```

- What do you think the data in the table are measures of?
- How can we tell the data are wide?

Wide to long: `pivot_longer()`

```r
SBP_wide
```

```
## # A tibble: 4 × 5
##   id    sex   SBP_v1 SBP_v2 SBP_v3
##   <chr> <chr>  <dbl>  <dbl>  <dbl>
## 1 a     F        130    110    112
## 2 b     M        120    116    122
## 3 c     M        130    136    138
## 4 d     F        119    106    118
```

For `pivot_longer` we need to **specify**:

- **cols**: which columns to make long
- **names_to**: the name of the variable that will be created from the data stored in the *column names*
- **values_to**: the name of the variable that will be created from the data stored in the *cell values*

```r
SBP_long <- SBP_wide %>%
  pivot_longer(
    cols=c(SBP_v1,SBP_v2,SBP_v3),
    names_to = "visit",
    values_to = "SBP")
SBP_long
```

```
## # A tibble: 12 × 4
##    id    sex   visit    SBP
##    <chr> <chr> <chr>  <dbl>
##  1 a     F     SBP_v1   130
##  2 a     F     SBP_v2   110
##  3 a     F     SBP_v3   112
##  4 b     M     SBP_v1   120
##  5 b     M     SBP_v2   116
##  6 b     M     SBP_v3   122
##  7 c     M     SBP_v1   130
##  8 c     M     SBP_v2   136
##  9 c     M     SBP_v3   138
## 10 d     F     SBP_v1   119
## 11 d     F     SBP_v2   106
```

Long to wide: `pivot_wider()`

```
SBP_long
```

```
## # A tibble: 12 × 4
##    id    sex   visit    SBP
##    <chr> <chr> <chr>  <dbl>
##  1 a     F     SBP_v1   130
##  2 a     F     SBP_v2   110
##  3 a     F     SBP_v3   112
##  4 b     M     SBP_v1   120
##  5 b     M     SBP_v2   116
##  6 b     M     SBP_v3   122
##  7 c     M     SBP_v1   130
##  8 c     M     SBP_v2   136
##  9 c     M     SBP_v3   138
## 10 d     F     SBP_v1   119
## 11 d     F     SBP_v2   106
## 12 d     F     SBP_v3   118
```

For `pivot_wider` we need to **specify**:

- **names_from**: which column contains the names for the *new columns*
- **values_from**: which column contains the values that will fill in the *cell values*

```
SBP_wide2 <- SBP_long %>%
  pivot_wider(names_from = "visit",
              values_from = "SBP")
SBP_wide2
```

```
## # A tibble: 4 × 5
##   id    sex   SBP_v1 SBP_v2 SBP_v3
##   <chr> <chr>  <dbl>  <dbl>  <dbl>
## 1 a     F        130    110    112
## 2 b     M        120    116    122
## 3 c     M        130    136    138
```

Clean up `visit` column in the long data (1/3)

```
SBP_long
```

```
## # A tibble: 12 × 4
##    id    sex   visit    SBP
##    <chr> <chr> <chr>  <dbl>
##  1 a     F     SBP_v1   130
##  2 a     F     SBP_v2   110
##  3 a     F     SBP_v3   112
##  4 b     M     SBP_v1   120
##  5 b     M     SBP_v2   116
##  6 b     M     SBP_v3   122
##  7 c     M     SBP_v1   130
##  8 c     M     SBP_v2   136
##  9 c     M     SBP_v3   138
## 10 d     F     SBP_v1   119
## 11 d     F     SBP_v2   106
## 12 d     F     SBP_v3   118
```

Goal: remove the string "`SBP_`" from the `visit` variable's values.

Method #1: tidy the `visit` column *after* making the data long

Method #2: tidy the `visit` column *while* making the data long

Clean up visit column in the long data (2/3)

Method #1: tidy the `visit` column *after* making the data long

```
SBP_long2 <- SBP_long %>%
  mutate(
    visit = str_replace(
      visit,
      pattern = "SBP_",
      replacement = "")
    )
SBP_long2
```

```
## # A tibble: 12 × 4
##    id    sex   visit   SBP
##    <chr> <chr> <chr> <dbl>
##  1 a     F     v1      130
##  2 a     F     v2      110
##  3 a     F     v3      112
##  4 b     M     v1      120
##  5 b     M     v2      116
##  6 b     M     v3      122
```

1. Note that `mutate` is replacing the existing `visit` column with new values
2. If I wanted to keep the original `visit` column instead of overwriting it, I would call the new column something else, such as `visit_clean`
3. Within `str_replace()`, double quotes need to be used around the characters specifying 1) the string of text to replace (`"SBP_"`) and 2) what to replace the text with, where `""` is used for *no text*
4. Could instead use `str_remove()`: `mutate(visit = str_remove(visit,"SBP_"))`

Clean up `visit` column in the long data (3/3)

Method #2: tidy the `visit` column *while* making the data long

```
SBP_long3 <- SBP_wide %>%
 pivot_longer(cols = c(SBP_v1, SBP_v2,SBP_v3),
              names_to = "visit",
              names_prefix = "SBP_",
              values_to = "SBP")
SBP_long3
```

```
## # A tibble: 12 × 4
##      id    sex   visit   SBP
##      <chr> <chr> <chr> <dbl>
##    1 a     F     v1      130
##    2 a     F     v2      110
##    3 a     F     v3      112
##    4 b     M     v1      120
##    5 b     M     v2      116
##    6 b     M     v3      122
##    7 c     M     v1      130
##    8 c     M     v2      136
```

Remarks:

1. Note the new parameter `names_prefix` specifying what the prefix is that *needs to be stripped*.
2. More complex `pivot_longer()` examples are shown at https://tidyr.tidyverse.org/articles *(the end of the url is articles/pivot.html)*

Specifying cols in pivot_longer

In the example creating `SBP_long`, the columns to make the tibble longer by were explicitly listed using `cols=c(SBP_v1,SBP_v2,SBP_v3)`.

```
SBP_long <- SBP_wide %>%
pivot_longer(cols = c(SBP_v1, SBP_v2,SBP_v3),
              names_to = "visit",
              values_to = "SBP")
SBP_long
```

```
## # A tibble: 12 × 4
##    id    sex   visit    SBP
##    <chr> <chr> <chr>  <dbl>
##  1 a     F     SBP_v1   130
##  2 a     F     SBP_v2   110
##  3 a     F     SBP_v3   112
##  4 b     M     SBP_v1   120
##  5 b     M     SBP_v2   116
##  6 b     M     SBP_v3   122
##  7 c     M     SBP_v1   130
##  8 c     M     SBP_v2   136
```

However, we can specify the columns in many different ways, just like with `select()`:

- `cols = c(SBP_v1:SBP_v3)`
- `cols = c(-id, -sex)`
- `cols = starts_wth("SBP")`
- `cols = contains("SBP")`

Notes on `pivot_*()` commands

- `pivot_longer()` and `pivot_wider()` are relatively new commands
- previously we used `gather()` and `spread()`,
    - which have different function parameters and are less intuitive to use
- if you search for help making data longer or wider,
    - you might still see references for `gather()` and `spread()`
- see my workshop slides for `gather()` and `spread()` usage

- see https://tidyr.tidyverse.org/articles/pivot.html for more `pivot` examples

Summary of data wrangling commands

command	purpose
`%>%`	join (pipe) together commands
`filter()`	subset rows
`arrange()`	sort rows
`select()`	select or rearrange columns
`rename()`	columns
`mutate()`	create new columns
`tabyl()`	summarize categorical data
`summarize()`	summarize data; for both categorical and numerical data
`group_by()`	group data by a categorical variable
`pivot_longer`	make a wide dataset long;
`pivot_wider`	to make a long dataset wide

This week we covered a *lot*, but learning R gets easier!

Allison Horst