Data Wrangling in R

materials: https://z.umn.edu/latisdwR

Alicia Hofelich Mohr, PhD David Olsen Seth Mayotte

Before we start:
1. Download & unzip folder from link
2. Install dplyr package in RStudio (install.packages("dplyr"))

2019-03-08

R & Data Science

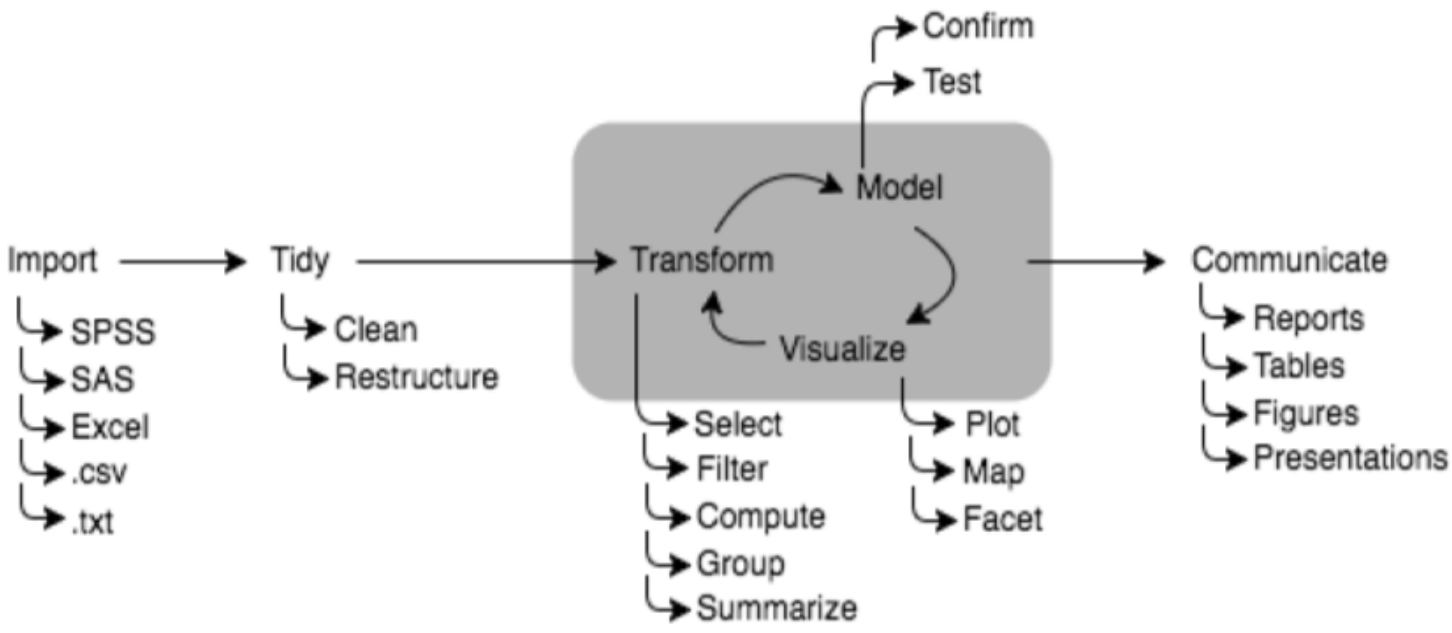

Figure Adapted from R for Data Science by Ethan Young

How dplyr fits in

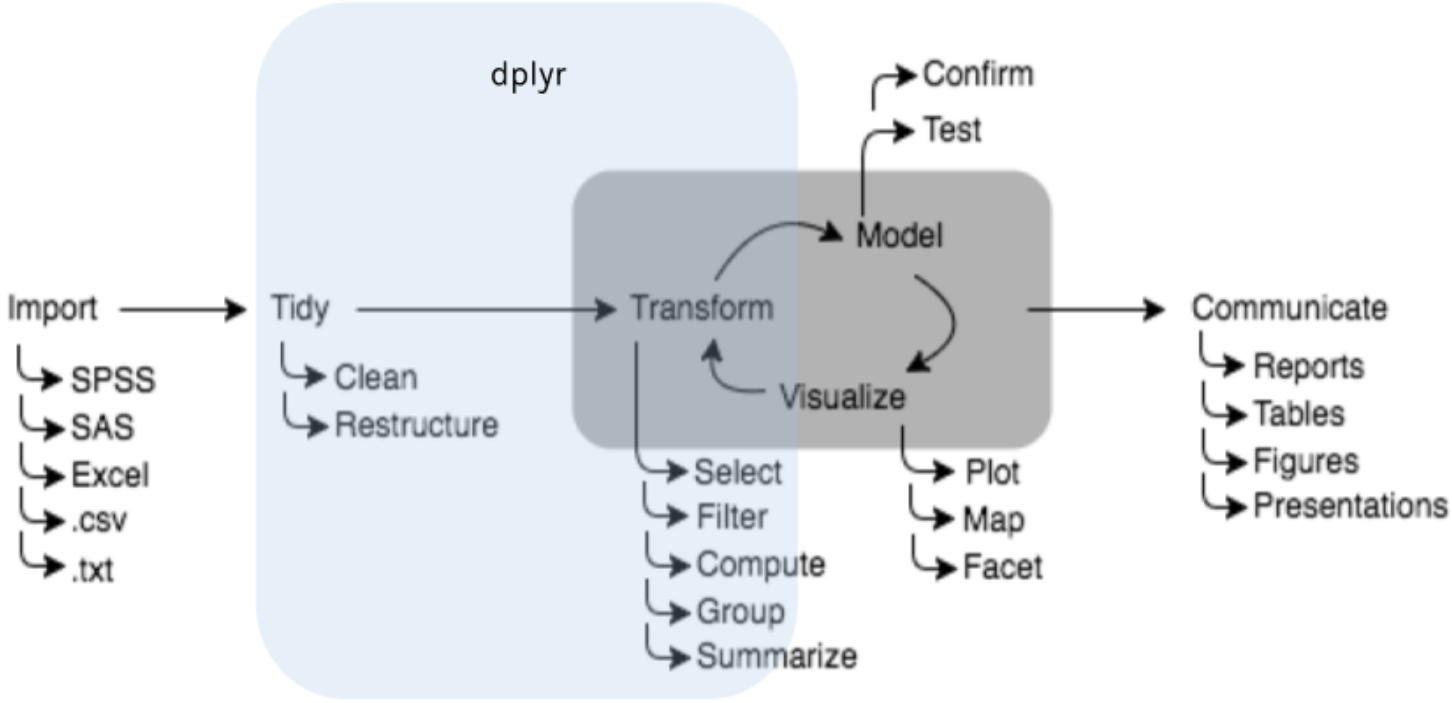

Figure Adapted from R for Data Science by Ethan Young

Agenda

- Introduction to dplyr & data
- Subsetting columns & filtering rows
- Ordering data
- Aggregating data
- Reshaping data

Slicing & Dicing: Base R vs dplyr

dplyr is part of tidyverse and contains functions for working with data.

In base R, you may have done subsetting using indices or functions like subset().

```
data[x, y]

subset(data, subset = ..., select = ...)
```

This will always work.

Slicing & Dicing: Base R vs dplyr

dplyr is part of tidyverse and contains functions for working with data.

In base R, you may have done subsetting using indices or functions like subset().

```
data[x, y]

subset(data, subset = ..., select = ...)
```

This will always work. But...

- variable/row position can change

Slicing & Dicing: Base R vs dplyr

dplyr is part of tidyverse and contains functions for working with data.

In base R, you may have done subsetting using indices or functions like subset().

```
data[x, y]

subset(data, subset = ..., select = ...)
```

This will always work. But...

- variable/row position can change
- base R isn't always human readable

Slicing & Dicing: Base R vs dplyr

dplyr is part of tidyverse and contains functions for working with data.

In base R, you may have done subsetting using indices or functions like subset().

```
data[x, y]

subset(data, subset = ..., select = ...)
```

This will always work. But...

- variable/row position can change
- base R isn't always human readable
- multiple functions require nesting

Readability of multi-step functions

Example: Want to make a cake

Readability of multi-step functions

Example: Want to make a cake

Base R

- Nested functions
- Step 1 on the inside

```
eat(frost(bake(cake)))
```

Readability of multi-step functions

Example: Want to make a cake.

Base R

- Or could create intermediate objects for each step

```
a <- bake(cake)
b <- frost(a)
eat(b)
```

Readability of multi-step functions

Example: Want to make a cake.

Base R

- Or could create intermediate objects for each step

```
a <- bake(cake)
b <- frost(a)
eat(b)
```

But end up with lots of objects in environment, and naming things is hard.

Readability of multi-step functions

Example: Want to make a cake.

dplyr

- String together functions with a pipe
- Step 1 is first

```
cake %>%
  bake() %>%
  frost() %>%
  eat()
```

Important dplyr functions

Verb based data manipulation:

- **select()** select variables
- **filter()** select rows
- **arrange()** order rows by variable
- **mutate()** create new variable
- **summarize()** aggregate the data
- **group_by()** group rows by variable

American Time Use Survey (ATUS)

Nationally representative survey from the Bureau of Labor Statistics on how Americans spend their time.

Subset of ATUS from the Minnesota Population Center (IPUMS)

- Years 2010, 2012, & 2013
- ~37,000 respondents
- Demographics
- Diary questions

ATUS Questions

Looking at the codebook, what variables would you be interested in exploring?

First pick a year (2010, 2012, 2013).

Then, pick 4 variables you want to explore further:

- Two categorical variables (demographics)
- Two continuous variables (activity variables)

To RStudio!

Open a new script in R for today's activities

- File -> New File -> R script

Let's start with some metadata at the top of your script:

```
##########################################
## Dplyr Workshop Notes
## March 8, 2019
##########################################
```

Installing dplyr

If you've never used dplyr, you will need to install the package.

```
install.packages("dplyr")
```

Then load it so the functions are available to R:

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Load the data into R

Remember to set the working directory

```
setwd("~/Desktop/Data Wrangling in R")
```

Load in the ATUS data

```
load("ATUSdata_subset.Rdata")
```

```
View(atus)
```

Select variables of interest

Take a subset of columns with **select()**

```
select(<data>, <unquoted var name>, <unquoted var name>, ... )
```

Select variables of interest

Take a subset of columns with **select()**

```
select(<data>, <unquoted var name>, <unquoted var name>, ... )
```

Select only year, region, and age:

```
atus.sub <- select(atus, YEAR, REGION, AGE)
head(atus.sub, n=3)
```

```
##   YEAR    REGION AGE
## 1 2010 Northeast  30
## 2 2010     South  27
## 3 2010   Midwest  20
```

Select variables of interest

Select variables within a range:

```
atus.sub <- select(atus, YEAR:RACE)
head(atus.sub)
```

```
##   YEAR    REGION HH_CHILD AGE    SEX              RACE
## 1 2010 Northeast       Yes  30 Female      Black only
## 2 2010     South       Yes  27 Female      White only
## 3 2010   Midwest       Yes  20   Male      White only
## 4 2010      West        No  69 Female Asian-Hawaiian
## 5 2010   Midwest        No  31 Female      White only
## 6 2010     South        No  63 Female      White only
```

Select variables of interest

Specialized helper functions can make selection easier:

- **starts_with()** select based on prefix

Select variables of interest

Specialized helper functions can make selection easier:

- **starts_with()** select based on prefix
- **ends_with()** select based on suffix

Select variables of interest

Specialized helper functions can make selection easier:

- **starts_with()** select based on prefix
- **ends_with()** select based on suffix
- **num_range()** select based on prefix & numeric range

Select variables of interest

Specialized helper functions can make selection easier:

- **starts_with()** select based on prefix
- **ends_with()** select based on suffix
- **num_range()** select based on prefix & numeric range
- **contains()** matches a string within the variable

Select variables of interest

Specialized helper functions can make selection easier:

- **starts_with()** select based on prefix
- **ends_with()** select based on suffix
- **num_range()** select based on prefix & numeric range
- **contains()** matches a string within the variable
- **matches()** more general matching using regular expressions

Select variables of interest

Specialized helper functions can make selection easier:

- **starts_with()** select based on prefix
- **ends_with()** select based on suffix
- **num_range()** select based on prefix & numeric range
- **contains()** matches a string within the variable
- **matches()** more general matching using regular expressions
- **one_of()** selects columns from a group of names

Select variables of interest

Select all the activity variables (start with "ACT_")

```
atus.act <- select(atus, starts_with("ACT"))
head(atus.act)
```

```
##   ACT_CAREHH ACT_FOOD ACT_HHACT ACT_PCARE ACT_PURCH ACT_SOCIA
## 1          0       50         0       405         0
## 2          0       92       205       775        45        15
## 3         58       40        60       695         5        36
## 4          0       80       250       810         0        27
## 5          0       80        20       590        10        63
## 6          0      110        15       570         0        25
##   ACT_VOL ACT_WORK
## 1       0      910
## 2       0        0
## 3       0        0
## 4       0        0
## 5       0        0
## 6       0        0
```

Filter relevant rows

Take a subset of the rows in a data set based on a criterion:

```
filter(<data>, <logical test> )
```

Filter relevant rows

Take a subset of the rows in a data set based on a criterion:

```
filter(<data>, <logical test> )
```

Keep only data for the Midwest

```
atus.midwest <- filter(atus.sub, REGION == "Midwest")
summary(atus.midwest)
```

```
##     YEAR              REGION           AGE
##   2010:3265   Northeast:   0   Min.   :15.00
##   2012:2948   Midwest  :8977   1st Qu.:33.00
##   2013:2764   South    :   0   Median :46.00
##               West     :   0   Mean   :47.38
##                                3rd Qu.:61.00
##                                Max.   :85.00
```

Filter relevant rows

Use the operators you would in base R:

- == is equal to
- >, >=, <, <= greater than/less than
- **!=** is not equal to
- **is.na()** is missing
- **between()** within range of numbers (inclusive)

Try it

First, create a subset of the atus data that contains the column YEAR plus the columns you are interested in using **select()**.

- Call this new data "atus.subset1".

Then, filter the rows of atus.subset1 to contain only the data for the year you selected using **filter()**.

- Call this new data object "atus.subset2".

Doing multiple things at the same time

Instead of nesting or creating intermediate objects like we did above, you can "chain" together multiple commands with pipes (%>%) in dplyr to do it in one step.

```
atus.subset <- atus %>%
  select(YEAR, REGION, AGE) %>%
  filter(YEAR == 2013)

summary(atus.subset)
```

```
##    YEAR                REGION          AGE
##  2010:    0   Northeast:1918   Min.   :15.00
##  2012:    0   Midwest  :2764   1st Qu.:34.00
##  2013:11385   South    :4179   Median :48.00
##               West     :2524   Mean   :48.28
##                                 3rd Qu.:62.00
##                                 Max.   :85.00
```

TIP: keyboard short for %>% is `cntl/cmd + shift + m`

Try it

Now, re-create your subset in a data object called “atus.subset” in a single command by chaining together the select and filter commands.

Arrange: Sorting data by variables

Ordering or sorting a dataset by specific variables can be done with **arrange()**; use **desc()** around variable name for descending.

```
#Sort by age asending
atus.subset.ar <- arrange(atus.subset, AGE)
head(atus.subset.ar, n=2)
```

```
##   YEAR  REGION AGE
## 1 2013 Midwest  15
## 2 2013   South  15
```

```
#age descending
atus.subset.ar <- arrange(atus.subset, desc(AGE))
head(atus.subset.ar, n=2)
```

```
##   YEAR  REGION AGE
## 1 2013 Midwest  85
## 2 2013   South  85
```

Mutate: Creating new variables

You are likely familiar with how to create a new variable in base R:

```
atus$EARNMONTH <- atus$EARNWEEK*4
summary(atus$EARNMONTH)
```

```
##     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##        0    1638    2944    3609    4769   11538   17351
```

Mutate: Creating new variables

You are likely familiar with how to create a new variable in base R:

```
atus$EARNMONTH <- atus$EARNWEEK*4
summary(atus$EARNMONTH)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##       0    1638    2944    3609    4769   11538   17351
```

mutate() is an alternate way to do this in dplyr using a function. This is especially useful when creating a variable within a chain.

```
atus <- mutate(atus, EARNMONTH = EARNWEEK*4)
summary(atus$EARNMONTH)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##       0    1638    2944    3609    4769   11538   17351
```

Try it

The "ACT_" variables in the atus data currently capture the number of **minutes** spent performing each activity.

For the activity variables in your "atus.subset":

1. Use mutate to create two new variables for the activities in **hours** (divide by 60)
2. Sort the dataset by one of these variables.

Summarizing data

Create specific summaries of the data with **summarize()**.

```
summarize(<data>, <summary_name> = <summary_function>,
          <summary_name> = <summary_function>, ...)
```

This will return a data frame with one row, and one column for each summary function.

Summarizing data

Create specific summaries of the data with **summarize()**.

```
summarize(<data>, <summary_name> = <summary_function>,
          <summary_name> = <summary_function>, ...)
```

This will return a data frame with one row, and one column for each summary function.

For example, let's say we want to take the average and sd of age

```
summarize(atus, mean_age = mean(AGE), sd_age = sd(AGE))
```

```
##   mean_age   sd_age
## 1 47.64892 17.82435
```

Summarizing data

Any function that returns a single value can be used in summarize().

Some useful ones include:

- **mean()**
- **median()**
- **sd()**
- **min()**, **max()**
- **cor()**
- **n()**
- **n_distinct()**

Try it

Use **summarize()** to calculate the mean and standard deviation for each of your activity variables

Grouping data

Often you may want to return summaries of the data grouped by categorical variables.

For example, instead of taking the mean of age across the entire dataset, want means of age for each region.

```
atus %>%
  group_by(REGION) %>%
  summarize(mean_age = mean(AGE),
            sd_age = sd(AGE),
            count = n())
```

```
## # A tibble: 4 x 4
##   REGION    mean_age sd_age count
##   <fct>        <dbl>  <dbl> <int>
## 1 Northeast     48.8   17.8  6277
## 2 Midwest       47.4   17.9  8977
## 3 South         48.0   17.8 13634
## 4 West          46.4   17.7  8200
```

Try it

Now try taking the mean and standard deviation you calculated earlier grouped by one of your demographic variables.

- Add in count (**n()**) in your summarize() command to return the count of respondents in each level of your demographic variable.

Next, try grouping by both demographic variables.

Challenge

Starting with the atus dataset, create a dataset that contains summary data by sex and whether there are children under 18 in the house. The data should contain:

- average hours worked that day
- average hours spent socializing
- correlation between hours spent working & socializing
- counts of respondents in each group

Before you start, consider:

1. What functions will you need?
2. In what order should you chain them?

Try it: Challenge & Solution

One possible solution:

```
#first, specify the data
atus %>%
  #then use mutate() to create new varibles for hours
  #(rather than minutes) worked
  mutate(ACT_WORK_HR = ACT_WORK/60,
         ACT_SOCIAL_HR = ACT_SOCIAL/60) %>%
  #the group by sex and whether they have children
  group_by(SEX, HH_CHILD) %>%
  #finally, create averages for the hours worked, socialized,
  #their correlation, and counts
  summarize(Avg_HRwork = mean(ACT_WORK_HR),
            AvgHRsocial = mean(ACT_SOCIAL_HR),
            CorWorkSocial = cor(ACT_WORK_HR, ACT_SOCIAL_HR),
            Count = n())
```

Try it: Challenge & Solution

```
## # A tibble: 4 x 6
## # Groups:   SEX [2]
##   SEX    HH_CHILD Avg_HRwork AvgHRsocial CorWorkSocial Count
##   <fct>  <fct>          <dbl>       <dbl>          <dbl> <int>
## 1 Male   No              2.78        6.02         -0.531  9439
## 2 Male   Yes             3.72        4.30         -0.492  6992
## 3 Female No              1.96        5.58         -0.456 11022
## 4 Female Yes             2.31        3.75         -0.370  9635
```

Reshaping data: Long versus wide

Sometimes our dataset is not in the format we need

Different parts of our analysis or plotting may require different "shapes" of the same data

Reshaping data: Long versus wide

Sometimes our dataset is not in the format we need

Different parts of our analysis or plotting may require different "shapes" of the same data

```
atus %>%
  select(EDUC, REGION) %>%
  group_by(EDUC, REGION) %>%
  summarize(count = n())
```

```
## # A tibble: 68 x 3
## # Groups:   EDUC [17]
##    EDUC                          REGION    count
##    <fct>                         <fct>     <int>
##  1 Less than 1st grade           Northeast     8
##  2 Less than 1st grade           Midwest      11
##  3 Less than 1st grade           South        23
##  4 Less than 1st grade           West         27
##  5 1st, 2nd, 3rd, or 4th grade Northeast    34
##  6 1st, 2nd, 3rd, or 4th grade Midwest      14
```

Reshaping data: Long vs wide

Long
Observations in multiple rows, with index in a column (more rows)

ID	Time	Measure
1	1	4.5
1	2	6.7
1	3	2.3
2	1	5.4
2	2	7.0
2	3	1.5

Wide
Observations in multiple columns within same row (more columns)

ID	Measure_Time1	Measure_Time2	Measure_Time3
1	4.5	6.7	2.3
2	5.4	7.0	1.5

Many options for reshaping

	tidyr	reshape2	Base R
Wide to Long	gather()	melt()	reshape(..., direction="long")
Long to Wide	spread()	dcast()	reshape(..., direction="wide")
Pros	Easy for simple datasets	Works well for many types of data	Can handle anything in single function
Cons	Limited: hard to use for complex data	Logic differs between functions	Hard to use; confusing arguments

Using reshape2

We're going to focus on reshape2 - IMO, balances functionality with ease of use.

First, let's install and load the reshape2 package.

```
install.packages("reshape2")
library(reshape2)
```

Long data

Save our summary table as a new (long) data frame

```
atus.long <- atus %>%
  select(EDUC, REGION) %>%
  group_by(EDUC, REGION) %>%
  summarize(count = n())

head(atus.long)
```

```
## # A tibble: 6 x 3
## # Groups:   EDUC [2]
##   EDUC                          REGION    count
##   <fct>                         <fct>     <int>
## 1 Less than 1st grade           Northeast     8
## 2 Less than 1st grade           Midwest      11
## 3 Less than 1st grade           South        23
## 4 Less than 1st grade           West         27
## 5 1st, 2nd, 3rd, or 4th grade Northeast    34
## 6 1st, 2nd, 3rd, or 4th grade Midwest      14
```

Going from long to wide: dcast()

dcast() takes in a formula:

ID variables $\sim$ **Measured Variables**, where

- **ID variables** are the variables you want to remain in rows
- **Measured Variables** are the variables for which you want separate columns

On either side of the equation, you can specify multiple ID or Measured variables with $+$

Going from long to wide: dcast()

dcast() takes in a formula:

ID variables $\sim$ **Measured Variables**, where

- **ID variables** are the variables you want to remain in rows
- **Measured Variables** are the variables for which you want separate columns

On either side of the equation, you can specify multiple ID or Measured variables with $+$

- Indicate which values should be in the columns set up by **Measured Variables** with `value.var = "variable name"`

ATUS long to wide

Reshape so that there are separate columns for each region

```
head(atus.long)
```

```
## # A tibble: 6 x 3
## # Groups:   EDUC [2]
##   EDUC                          REGION    count
##   <fct>                         <fct>     <int>
## 1 Less than 1st grade           Northeast     8
## 2 Less than 1st grade           Midwest      11
## 3 Less than 1st grade           South        23
## 4 Less than 1st grade           West         27
## 5 1st, 2nd, 3rd, or 4th grade Northeast    34
## 6 1st, 2nd, 3rd, or 4th grade Midwest      14
```

EDUC will be the **ID variable**, REGION is our **Measured variable**, and count is our value variable.

ATUS long to wide

```
atus.wide <- dcast(atus.long,
                   EDUC ~ REGION,
                   value.var = "count")

head(atus.wide)
```

```
##                              EDUC Northeast Midwest South West
## 1          Less than 1st grade          8      11    23   27
## 2 1st, 2nd, 3rd, or 4th grade         34      14   102   91
## 3              5th or 6th grade         50      49   190  172
## 4              7th or 8th grade        132     164   370  141
## 5                   9th grade          157     203   442  215
## 6                  10th grade          204     268   496  235
```

Tagging on with dplyr

But why make new objects when you can add on with pipes? :)

```
atus %>%
  select(EDUC, REGION) %>%
  group_by(EDUC, REGION) %>%
  summarize(count = n()) %>%
  dcast(EDUC ~ REGION, value.var = "count") %>%
  head()
```

```
##                                EDUC Northeast Midwest South West
## 1                Less than 1st grade         8      11    23   27
## 2 1st, 2nd, 3rd, or 4th grade        34      14   102   91
## 3                   5th or 6th grade        50      49   190  172
## 4                   7th or 8th grade       132     164   370  141
## 5                         9th grade       157     203   442  215
## 6                        10th grade       204     268   496  235
```

Try it

What columns would the following dataset contain? Guess, then try!

```
atus %>%
  select(EDUC, REGION, SEX) %>%
  group_by(EDUC, REGION, SEX) %>%
  summarize(count = n()) %>%
  dcast(EDUC ~ REGION + SEX, value.var = "count")
```

How will this one differ?

```
atus %>%
  select(EDUC, REGION, SEX) %>%
  group_by(EDUC, REGION, SEX) %>%
  summarize(count = n()) %>%
  dcast(EDUC + REGION ~ SEX, value.var = "count")
```

Try it: solutions

What columns would the following dataset contain?

```
atus %>%
  select(EDUC, REGION, SEX) %>%
  group_by(EDUC, REGION, SEX) %>%
  summarize(count = n()) %>%
  dcast(EDUC ~ REGION + SEX, value.var = "count") %>%
  head()
```

```
##                          EDUC Northeast_Male Northeast_Female Midwes
## 1          Less than 1st grade              4                4
## 2 1st, 2nd, 3rd, or 4th grade             15               19
## 3             5th or 6th grade             21               29
## 4             7th or 8th grade             56               76
## 5                 9th grade                70               87
## 6                10th grade                89              115
##   Midwest_Female South_Male South_Female West_Male West_Female
## 1              3          12            11        10           17
## 2              6          55            47        40           51
## 3             22          92            98        66          106
## 4             86         149           221        57           84
## 5            108         224           218       105          110
```

Try it: solutions

How will this one differ?

```
atus %>%
  select(EDUC, REGION, SEX) %>%
  group_by(EDUC, REGION, SEX) %>%
  summarize(count = n()) %>%
  dcast(EDUC + REGION ~ SEX, value.var = "count") %>%
  head()
```

```
##                                 EDUC    REGION Male Female
## 1          Less than 1st grade Northeast    4      4
## 2          Less than 1st grade   Midwest    8      3
## 3          Less than 1st grade     South   12     11
## 4          Less than 1st grade      West   10     17
## 5 1st, 2nd, 3rd, or 4th grade Northeast   15     19
## 6 1st, 2nd, 3rd, or 4th grade   Midwest    8      6
```

Going from wide to long: melt()

Reshape columns for education back into rows:

```
head(atus.wide)
```

```
##                                EDUC Northeast Midwest South West
## 1                Less than 1st grade         8      11    23   27
## 2 1st, 2nd, 3rd, or 4th grade         34      14   102   91
## 3                   5th or 6th grade         50      49   190  172
## 4                   7th or 8th grade        132     164   370  141
## 5                         9th grade        157     203   442  215
## 6                        10th grade        204     268   496  235
```

You can go the other way: melt()

To go from wide to long, use **melt()**.

It's a much easier function than **dcast()**.

- Takes a vector of **id variables**
- It assumes the rest should be collapsed into:
  - a categorical **variable** that contains column names, and
  - a **value** that contains the cell contents

Example wide to long

```
atus.long.again <- melt(atus.wide, id.vars = "EDUC")
head(atus.long.again)
```

```
##                                EDUC  variable value
## 1          Less than 1st grade Northeast      8
## 2 1st, 2nd, 3rd, or 4th grade Northeast     34
## 3              5th or 6th grade Northeast     50
## 4              7th or 8th grade Northeast    132
## 5                   9th grade Northeast    157
## 6                  10th grade Northeast    204
```

Going from wide to long: melt()

Can also specify the names of the newly created variables:

```
atus.long.again <- melt(atus.wide, id.vars = "EDUC", variab
head(atus.long.again)
```

```
##                                EDUC    REGION count
## 1          Less than 1st grade Northeast      8
## 2 1st, 2nd, 3rd, or 4th grade Northeast     34
## 3             5th or 6th grade Northeast     50
## 4             7th or 8th grade Northeast    132
## 5                   9th grade Northeast    157
## 6                  10th grade Northeast    204
```

Questions?

slides & data: https://z.umn.edu/latisdwR

Contact us

- latisresearch@umn.edu
- hofelich@umn.edu