Subsetting Data in R

Reminder

Refresh the website and get the latest version of the labs and slides! We are constantly making improvements.

Recap

- Use `<-` to save (assign) values to objects
- Use `c()` to **combine** vectors
- `length()`, `class()`, and `str()` tell you information about an object
- The sequence `seq()` function helps you create numeric vectors (`from`,`to`, `by`, and `length.out` arguments)
- The repeat `rep()` function helps you create vectors with the `each` and `times` arguments
- Reproducible science makes everyone's life easier!
- `readr`has helpful functions like `read_csv()` that can help you import data into R

 Cheatsheet

Overview

In this module, we will show you how to:

1. Look at your data in different ways
2. Create a data frame and a tibble
3. Create new variables/make rownames a column
4. Rename columns of a data frame
5. Subset rows of a data frame
6. Subset columns of a data frame
7. Add/remove new columns to a data frame
8. Order the columns of a data frame
9. Order the rows of a data frame

Setup

We will largely focus on the `dplyr` package which is part of the `tidyverse`.

Some resources on how to use `dplyr`:

- https://dplyr.tidyverse.org/
- https://cran.r-project.org/web/packages/dplyr/vignettes/dplyr.html
- https://www.opencasestudies.org/

Why dplyr?

hadley commented on May 26, 2016 — Member

The d is for dataframes, the plyr is to evoke pliers. Pronounce however you like.

👍 1

The `dplyr` package is one of the most helpful packages for altering your data to get it into a form that is useful for creating visualizations, summarizing, or more deeply analyzing.

So you can imagine using pliers on your data.

Loading in dplyr and tidyverse

See this website for a list of the packages included in the `tidyverse`: https://www.tidyverse.org/packages/

```
library(tidyverse) # loads dplyr and other packages!
```

Getting data to work with

We will use a dataset from a project we worked on called Open Case Studies.

See https://www.opencasestudies.org/.

We have added the specific data to our website.

Getting data to work with

We will load data from one of the case studies about opioid shipments.

See https://www.opencasestudies.org/ocs-bp-opioid-rural-urban/ about this data.

Import the data

We will work with data called `annualDosage` (number of shipments (count) of either oxycodone or hydrocodone pills (DOSAGE_UNIT)).

```
annualDosage <-
  read_csv("https://jhudatascience.org/intro_to_r/data/annualDosage.csv")
```

```
Rows: 27758 Columns: 6
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
chr (3): BUYER_COUNTY, BUYER_STATE, countyfips
dbl (3): year, count, DOSAGE_UNIT

ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message
```

Checking the data `dim()`

The `dim()`, `nrow()`, and `ncol()` functions are good options to check the dimensions of your data before moving forward.

```
dim(annualDosage) # rows, columns
```

```
[1] 27758     6
```

```
nrow(annualDosage) # number of rows
```

```
[1] 27758
```

```
ncol(annualDosage) # number of columns
```

```
[1] 6
```

Checking the data: `glimpse()`

In addition to `head()` and `tail()`, the `glimpse()`function of the `dplyr` package is another great function to look at your data.

```
glimpse(annualDosage)
```

```
Rows: 27,758
Columns: 6
$ BUYER_COUNTY <chr> "ABBEVILLE", "ABBEVILLE", "ABBEVILLE", "ABBEVILLE", "ABBE…
$ BUYER_STATE  <chr> "SC", "SC", "SC", "SC", "SC", "SC", "SC", "SC", "SC", "LA…
$ year         <dbl> 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 200…
$ count        <dbl> 877, 908, 871, 930, 1197, 1327, 1509, 1572, 1558, 5802, 5…
$ DOSAGE_UNIT  <dbl> 363620, 402940, 424590, 467230, 539280, 566560, 589010, 5…
$ countyfips   <chr> "45001", "45001", "45001", "45001", "45001", "45001", "45…
```

Checking your data: `slice_sample()`

What if you want to see the middle of your data? You can use the `slice_sample()` function of the `dplyr` package to see a **random** set of rows. You can specify the number of rows with the `n` argument.

```
slice_sample(annualDosage, n = 2)
```

```
# A tibble: 2 × 6
  BUYER_COUNTY BUYER_STATE  year count DOSAGE_UNIT countyfips
  <chr>        <chr>       <dbl> <dbl>       <dbl> <chr>
1 NICOLLET     MN           2014  1218      327780 27103
2 WASHINGTON   ID           2009   762      400020 16087
```

```
slice_sample(annualDosage, n = 2)
```

```
# A tibble: 2 × 6
  BUYER_COUNTY BUYER_STATE  year count DOSAGE_UNIT countyfips
  <chr>        <chr>       <dbl> <dbl>       <dbl> <chr>
1 MONO         CA           2008   523      227475 06051
2 HENRY        KY           2014  1039      567530 21103
```

Data frames and tibbles

Data frames

An older version of data in tables is called a data frame. The mtcars dataset is an example of this.

```
class(mtcars)
```

```
[1] "data.frame"
```

```
head(mtcars)
```

```
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

tibble

Tibbles are a **fancier** version of data frames:

- We don't have to use head to see a preview of it
- We see the dimensions
- We see the data types for each column

`annualDosage`

```
# A tibble: 27,758 × 6
   BUYER_COUNTY BUYER_STATE  year count DOSAGE_UNIT countyfips
   <chr>        <chr>       <dbl> <dbl>       <dbl> <chr>
 1 ABBEVILLE    SC           2006   877      363620 45001
 2 ABBEVILLE    SC           2007   908      402940 45001
 3 ABBEVILLE    SC           2008   871      424590 45001
 4 ABBEVILLE    SC           2009   930      467230 45001
 5 ABBEVILLE    SC           2010  1197      539280 45001
 6 ABBEVILLE    SC           2011  1327      566560 45001
 7 ABBEVILLE    SC           2012  1509      589010 45001
 8 ABBEVILLE    SC           2013  1572      596420 45001
 9 ABBEVILLE    SC           2014  1558      641350 45001
10 ACADIA       LA           2006  5802     1969720 22001
# ℹ 27,748 more rows
```

Creating a `tibble`

If we wanted to create a `tibble` ("fancy" data frame), we can using the `tibble()` function on a data frame.

```
tbl_mtcars <- tibble(mtcars)
tbl_mtcars
```

```
# A tibble: 32 × 11
     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1  21       6  160    110  3.9   2.62  16.5     0     1     4     4
 2  21       6  160    110  3.9   2.88  17.0     0     1     4     4
 3  22.8     4  108     93  3.85  2.32  18.6     1     1     4     1
 4  21.4     6  258    110  3.08  3.22  19.4     1     0     3     1
 5  18.7     8  360    175  3.15  3.44  17.0     0     0     3     2
 6  18.1     6  225    105  2.76  3.46  20.2     1     0     3     1
 7  14.3     8  360    245  3.21  3.57  15.8     0     0     3     4
 8  24.4     4  147.    62  3.69  3.19  20       1     0     4     2
 9  22.8     4  141.    95  3.92  3.15  22.9     1     0     4     2
10  19.2     6  168.   123  3.92  3.44  18.3     1     0     4     4
# … 22 more rows
```

Note don't necessarily need to use `head()` with tibbles, as they conveniently print a portion of the data.

Summary of tibbles and data frames

We generally recommend using tibbles, but you are likely to run into lots of data frames with your work.

Most functions work for both so you don't need to worry about it much!

It can be helpful to convert data frames to tibbles though just to see more about the data more easily. The `tibble()` function helps us do that.

Data frames vs tibbles - watch out for rownames

Note that this conversion can remove row names - which some data frames have. For example, `mtcars` (part of R) has row names. In this case we would want to make the rownames a new column first before making into a tibble.

```
head(mtcars, n = 2)
```

```
               mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4       21   6  160 110  3.9 2.620 16.46  0  1    4    4
Mazda RX4 Wag   21   6  160 110  3.9 2.875 17.02  0  1    4    4
```

```
head(tibble(mtcars), n = 2)
```

```
# A tibble: 2 × 11
    mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1    21     6   160   110   3.9  2.62  16.5     0     1     4     4
2    21     6   160   110   3.9  2.88  17.0     0     1     4     4
```

rownames_to_column function

There is a function that specifically helps you do that.

```
head(rownames_to_column(mtcars), n = 2)
```

```
        rowname mpg cyl disp  hp drat    wt  qsec vs am gear carb
1     Mazda RX4  21   6  160 110  3.9 2.620 16.46  0  1    4    4
2 Mazda RX4 Wag  21   6  160 110  3.9 2.875 17.02  0  1    4    4
```

```
head(tibble(rownames_to_column(mtcars)), n = 2)
```

```
# A tibble: 2 × 12
  rowname        mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
  <chr>        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Mazda RX4       21     6   160   110   3.9  2.62  16.5     0     1     4     4
2 Mazda RX4 W…    21     6   160   110   3.9  2.88  17.0     0     1     4     4
```

Data for now

Let's stick with the tibble annualDosage data for our next lesson

```
head(annualDosage)
```

```
# A tibble: 6 × 6
  BUYER_COUNTY BUYER_STATE  year count DOSAGE_UNIT countyfips
  <chr>        <chr>       <dbl> <dbl>       <dbl> <chr>
1 ABBEVILLE    SC           2006   877      363620 45001
2 ABBEVILLE    SC           2007   908      402940 45001
3 ABBEVILLE    SC           2008   871      424590 45001
4 ABBEVILLE    SC           2009   930      467230 45001
5 ABBEVILLE    SC           2010  1197      539280 45001
6 ABBEVILLE    SC           2011  1327      566560 45001
```

Renaming Columns

rename function

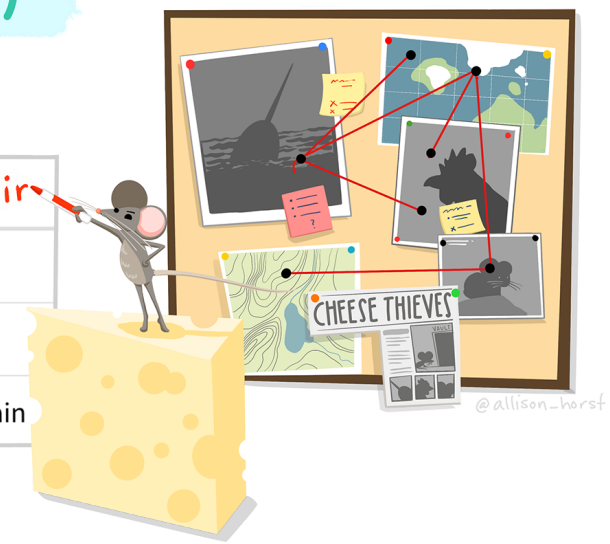
dplyr::rename()

RENAME COLUMNS*

df %>% rename(lair = site)

~~species~~ nemesis	status	~~site~~ lair
narwhal	unknown	ocean
chicken	active	coop
pika	active	mountain

*See **rename_with()** to rename using a function.



"Artwork by @allison_horst". https://allisonhorst.com/

Renaming Columns of a data frame or tibble

To rename columns in `dplyr`, you can use the `rename` function.

For example, let's rename BUYER_COUNTY to County. Notice the new name is listed **first**, similar to how a new object is assigned on the left!

```
# general format! not code!
{data you are creating or changing} <- rename({data you are using},
                                              {New Name} = {Old name})
```

```
renamed_annualDosage<- rename(annualDosage, County = BUYER_COUNTY)
head(renamed_annualDosage)
```

```
# A tibble: 6 × 6
  County    BUYER_STATE  year count DOSAGE_UNIT countyfips
  <chr>     <chr>       <dbl> <dbl>       <dbl> <chr>
1 ABBEVILLE SC           2006   877      363620 45001
2 ABBEVILLE SC           2007   908      402940 45001
3 ABBEVILLE SC           2008   871      424590 45001
4 ABBEVILLE SC           2009   930      467230 45001
5 ABBEVILLE SC           2010  1197      539280 45001
6 ABBEVILLE SC           2011  1327      566560 45001
```

Take Care with Column Names

When you can, avoid spaces, special punctuation, or numbers in column names, as these require special treatment to refer to them.

See https://jhudatascience.org/intro_to_r/resources/quotes_vs_backticks.html for more guidance.

```
 # this will cause an error
renamed_annualDosage <- rename(annualDosage, County! = BUYER_COUNTY)

# this will work
renamed_annualDosage <- rename(annualDosage, `County!` = BUYER_COUNTY)
head(renamed_annualDosage, 2)

# A tibble: 2 × 6
  `County!` BUYER_STATE  year count DOSAGE_UNIT countyfips
  <chr>     <chr>       <dbl> <dbl>       <dbl> <chr>
1 ABBEVILLE SC           2006   877      363620 45001
2 ABBEVILLE SC           2007   908      402940 45001
```

Unusual Column Names

It’s best to avoid unusual column names where possible, as things get tricky later.

We just showed the use of ` backticks` . You may see people use quotes as well.

Other atypical column names are those with:

- spaces
- number without characters
- number starting the name
- other punctuation marks (besides “_” or “.” and not at the beginning)

A solution!

Rename tricky column names so that you don’t have to deal with them later!

Be careful about copy pasting code!

Curly quotes will not work!

```
# this will cause an error!
renamed_annualDosage <- rename(annualDosage, ‘County!’ = BUYER_COUNTY)

# this will work!
renamed_annualDosage <- rename(annualDosage, 'County!' = BUYER_COUNTY)
```

Also true for double quotes

```
 # this will cause an error!
renamed_annualDosage <- rename(annualDosage, “County!” = BUYER_COUNTY)

# this will work!
renamed_annualDosage <- rename(annualDosage, "County!" = BUYER_COUNTY)
```

Rename multiple columns

A comma can separate different column names to change.

```
renamed_annualDosage <- rename(annualDosage,
                               County = BUYER_COUNTY,
                               State = BUYER_STATE)
head(renamed_annualDosage, 3)
```

```
# A tibble: 3 × 6
  County    State  year count DOSAGE_UNIT countyfips
  <chr>     <chr> <dbl> <dbl>       <dbl> <chr>
1 ABBEVILLE SC     2006   877      363620 45001
2 ABBEVILLE SC     2007   908      402940 45001
3 ABBEVILLE SC     2008   871      424590 45001
```

Renaming all columns of a data frame: dplyr

To rename all columns you use the `rename_with()`. In this case we will use `toupper()` to make all letters upper case. Could also use `tolower()` function.

```
annualDosage_upper <- rename_with(annualDosage, toupper)
head(annualDosage_upper, 3)
```

```
# A tibble: 3 × 6
  BUYER_COUNTY BUYER_STATE  YEAR COUNT DOSAGE_UNIT COUNTYFIPS
  <chr>        <chr>       <dbl> <dbl>       <dbl> <chr>
1 ABBEVILLE    SC           2006   877      363620 45001
2 ABBEVILLE    SC           2007   908      402940 45001
3 ABBEVILLE    SC           2008   871      424590 45001
```

```
annualDosage_lower<- rename_with(annualDosage, tolower)
head(annualDosage_lower, 3)
```

```
# A tibble: 3 × 6
  buyer_county buyer_state  year count dosage_unit countyfips
  <chr>        <chr>       <dbl> <dbl>       <dbl> <chr>
1 ABBEVILLE    SC           2006   877      363620 45001
2 ABBEVILLE    SC           2007   908      402940 45001
3 ABBEVILLE    SC           2008   871      424590 45001
```

janitor package

If you need to do lots of naming fixes - look into the janitor package!

```
#install.packages("janitor")
library(janitor)
```

janitor `clean_names`

The `clean_names` function can intuit what fixes you might need. Here it makes everything consistent.

```
head(annualDosage, 2)
```

```
# A tibble: 2 × 6
  BUYER_COUNTY BUYER_STATE  year count DOSAGE_UNIT countyfips
  <chr>        <chr>       <dbl> <dbl>       <dbl> <chr>
1 ABBEVILLE    SC           2006   877      363620 45001
2 ABBEVILLE    SC           2007   908      402940 45001
```

```
clean_AD <- clean_names(annualDosage)
head(clean_AD, 2)
```

```
# A tibble: 2 × 6
  buyer_county buyer_state  year count dosage_unit countyfips
  <chr>        <chr>       <dbl> <dbl>       <dbl> <chr>
1 ABBEVILLE    SC           2006   877      363620 45001
2 ABBEVILLE    SC           2007   908      402940 45001
```

more of clean_names

`clean_names` can also get rid of spaces and replace them with _.

```
test <- tibble(`col 1` = c(1,2,3), `col 2` = c(2,3,4))
test
```

```
# A tibble: 3 × 2
  `col 1` `col 2`
    <dbl>   <dbl>
1       1       2
2       2       3
3       3       4
```

```
clean_names(test)
```

```
# A tibble: 3 × 2
  col_1 col_2
  <dbl> <dbl>
1     1     2
2     2     3
3     3     4
```

GUT CHECK: Which of the following would work well with a column called `counties_of_US_with_population_over_10,000`?

A. Renaming it using `rename` function to something simpler like `counties_over_10thous`.

B. Keeping it as is and use backticks around the column name when you use it.

C. Keeping it as is and use quotes around the column name when you use it.

Summary

- data frames are simpler version of a data table
- tibbles are fancier `tidyverse` version
- tibbles are made with `tibble()`
- if your original data has rownames, you need to use `rownames_to_column` before converting to tibble
- the `rename()` function of `dplyr` can help you rename columns
- avoid using punctuation (except underscores), spaces, and numbers (to start or alone) in column names
- if you must do a nonstandard column name - typically use backticks around it. See https://jhudatascience.org/intro_to_r/resources/quotes_vs_backticks.html.
- avoid copy and pasting code from other sources - quotation marks will change!
- check out `janitor` if you need to make lots of column name changes

Lab Part 1

- Class Website
- Lab
- Day 3 Cheatsheet
- Posit's `dplyr` Cheatsheet # Subsetting Columns

Let's get our data again

This time lets also make it a smaller subset so it is easier for us to see the full dataset as we work through examples.

```
#read_csv("https://jhudatascience.org/intro_to_r/data/annualDosage.csv")
set.seed(1234)
AD <-slice_sample(annualDosage, n = 30)
```

Subset columns of a data frame - `tidyverse` way:

To grab a vector version (or "pull" out) the `year` column the `tidyverse` way we can use the `pull` function:

```
pull(AD, year)
```

```
 [1] 2006 2014 2013 2012 2009 2010 2010 2010 2009 2012 2010 2006 2007 2010 2011
[16] 2008 2009 2010 2007 2008 2006 2007 2013 2012 2011 2006 2013 2008 2013 2010
```

Subset columns of a data frame: dplyr

The `select` command from `dplyr` allows you to subset (still a `tibble`!)

```
select(AD, year)
```

```
# A tibble: 30 × 1
    year
   <dbl>
 1  2006
 2  2014
 3  2013
 4  2012
 5  2009
 6  2010
 7  2010
 8  2010
 9  2009
10  2012
# … 20 more rows
```

GUT CHECK: What function would be useful for getting a vector version of a column?

A. `pull()`

B. `select()`

Select multiple columns

We can use `select` to select for multiple columns.

```
select(AD, year, BUYER_COUNTY)
```

```
# A tibble: 30 × 2
    year BUYER_COUNTY
   <dbl> <chr>
 1  2006 EASTLAND
 2  2014 FALLS CHURCH CITY
 3  2013 DOUGLAS
 4  2012 FAYETTE
 5  2009 STAFFORD
 6  2010 GILES
 7  2010 ANDREW
 8  2010 LUNENBURG
 9  2009 HAWKINS
10  2012 ATCHISON
# ℹ 20 more rows
```

Subset columns of a data frame: dplyr

Note that if you want the values (not a `tibble`), use `pull` - as it pulls out the data:

```
pull(AD, year)
```

```
 [1] 2006 2014 2013 2012 2009 2010 2010 2010 2009 2012 2010 2006 2007 2010 2011
[16] 2008 2009 2010 2007 2008 2006 2007 2013 2012 2011 2006 2013 2008 2013 2010
```

```
# pull with select works too!
```

```
pull(select(AD, year))
```

```
 [1] 2006 2014 2013 2012 2009 2010 2010 2010 2009 2012 2010 2006 2007 2010 2011
[16] 2008 2009 2010 2007 2008 2006 2007 2013 2012 2011 2006 2013 2008 2013 2010
```

Select columns of a data frame: dplyr

The `select` command from `dplyr` allows you to subset columns matching patterns:

```
head(AD, 2)
```

```
# A tibble: 2 × 6
  BUYER_COUNTY      BUYER_STATE  year count DOSAGE_UNIT countyfips
  <chr>             <chr>       <dbl> <dbl>       <dbl> <chr>
1 EASTLAND          TX           2006  1980      723690 48133
2 FALLS CHURCH CITY VA           2014  1340      531270 51610
```

```
select(AD, starts_with("B"))
```

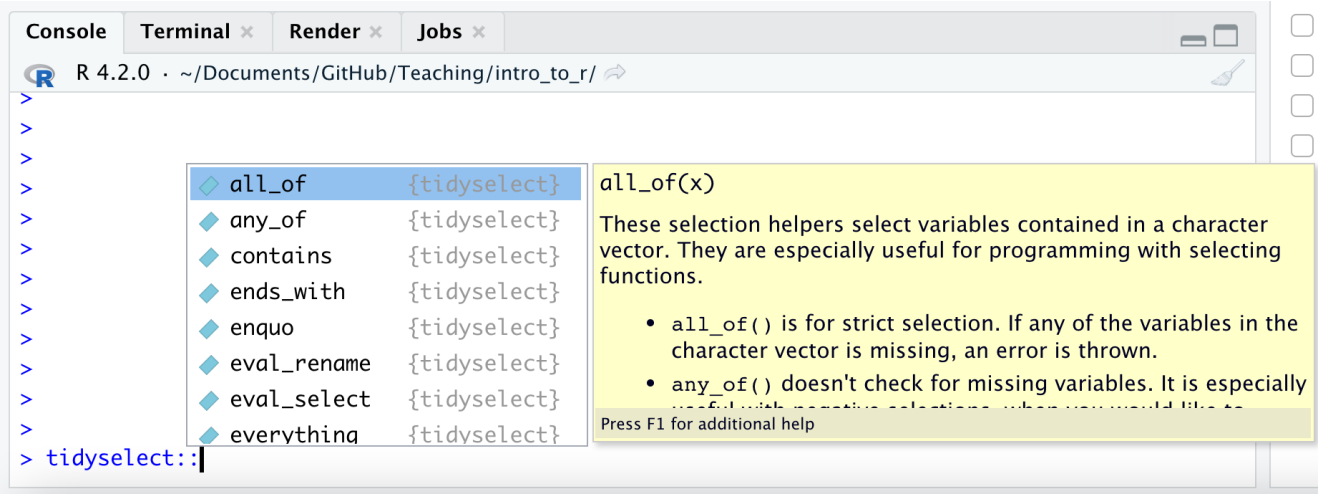
```
# A tibble: 30 × 2
   BUYER_COUNTY      BUYER_STATE
   <chr>             <chr>
 1 EASTLAND          TX
 2 FALLS CHURCH CITY VA
 3 DOUGLAS           CO
 4 FAYETTE           GA
 5 STAFFORD          VA
 6 GILES             TN
 7 ANDREW            MO
 8 LUNENBURG         VA
 9 HAWKINS           TN
10 ATCHISON          MO
# ℹ 20 more rows
```

See the Select "helpers"

Here are a few:

```
last_col()
starts_with()
ends_with()
contains() # like searching
```

Type `tidyselect::` in the **console** and see what RStudio suggests:



Combining tidyselect helpers with regular selection

```
head(AD, 2)
```

```
# A tibble: 2 × 6
  BUYER_COUNTY      BUYER_STATE  year count DOSAGE_UNIT countyfips
  <chr>             <chr>       <dbl> <dbl>       <dbl> <chr>
1 EASTLAND          TX           2006  1980      723690 48133
2 FALLS CHURCH CITY VA           2014  1340      531270 51610
```

```
select(AD, starts_with("B"), year)
```

```
# A tibble: 30 × 3
   BUYER_COUNTY      BUYER_STATE  year
   <chr>             <chr>       <dbl>
 1 EASTLAND          TX           2006
 2 FALLS CHURCH CITY VA           2014
 3 DOUGLAS           CO           2013
 4 FAYETTE           GA           2012
 5 STAFFORD          VA           2009
 6 GILES             TN           2010
 7 ANDREW            MO           2010
 8 LUNENBURG         VA           2010
 9 HAWKINS           TN           2009
10 ATCHISON          MO           2012
# ℹ 20 more rows
```

Multiple tidyselect functions

Follows OR logic.

```
select(AD, starts_with("B"), ends_with("r"))
```

```
# A tibble: 30 × 3
   BUYER_COUNTY      BUYER_STATE  year
   <chr>             <chr>       <dbl>
 1 EASTLAND          TX           2006
 2 FALLS CHURCH CITY VA           2014
 3 DOUGLAS           CO           2013
 4 FAYETTE           GA           2012
 5 STAFFORD          VA           2009
 6 GILES             TN           2010
 7 ANDREW            MO           2010
 8 LUNENBURG         VA           2010
 9 HAWKINS           TN           2009
10 ATCHISON          MO           2012
# ℹ 20 more rows
```

The `where()` function can help select columns of a specific class

`is.character()` and `is.numeric()` are often the most helpful

```
head(AD, 2)
```

```
# A tibble: 2 × 6
  BUYER_COUNTY      BUYER_STATE  year count DOSAGE_UNIT countyfips
  <chr>             <chr>       <dbl> <dbl>       <dbl> <chr>
1 EASTLAND          TX           2006  1980      723690 48133
2 FALLS CHURCH CITY VA           2014  1340      531270 51610
```

```
select(AD, where(is.numeric))
```

```
# A tibble: 30 × 3
    year count DOSAGE_UNIT
   <dbl> <dbl>       <dbl>
 1  2006  1980      723690
 2  2014  1340      531270
 3  2013 20961     6277640
 4  2012 12978     3701320
 5  2009  7921     2904600
 6  2010  4210     2074530
 7  2010  1167      315180
 8  2010   763      246130
 9  2009  7148     3420480
10  2012   601      204700
# ℹ 20 more rows
```

Subsetting Rows

Subset rows of a data frame: dplyr

The command in `dplyr` for subsetting rows is `filter`.

```
filter(AD, count > 10000)
```

```
# A tibble: 6 × 6
  BUYER_COUNTY BUYER_STATE  year count DOSAGE_UNIT countyfips
  <chr>        <chr>       <dbl> <dbl>       <dbl> <chr>
1 DOUGLAS      CO           2013 20961     6277640 08035
2 FAYETTE      GA           2012 12978     3701320 13113
3 BUTTE        CA           2006 20443    13802710 06007
4 PUEBLO       CO           2008 23932     9386790 08101
5 ROGERS       OK           2008 11150     4499860 40131
6 CABELL       WV           2013 18481     7806600 54011
```

Subset rows of a data frame: dplyr

You can have multiple logical conditions using the following:

- `==` : equals to
- `!=`: not equal to (`!` : not/negation)
- `>` / `<`: greater than / less than
- `>=` or `<=`: greater than or equal to / less than or equal to
- `&` : AND
- `|` : OR

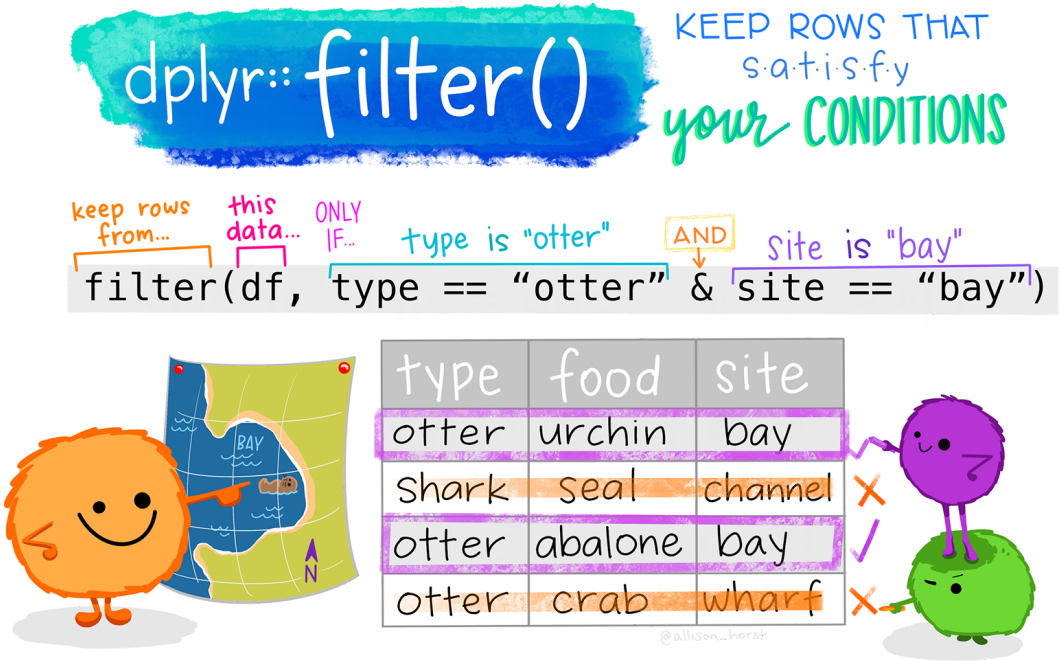
Common error for filter

If you try to filter for a column that does not exist it will not work:

- misspelled column name
- column that was already removed

Remember that filter works by keeping rows with values that match the conditions specified for particular columns.

`filter` function



"Artwork by @allison_horst". https://allisonhorst.com/

Subset rows of a data frame: dplyr

You can filter by two conditions using & or commas (must meet both conditions):

```
filter(AD, count > 10000, year == 2012)
```

```
filter(AD, count > 10000 & year == 2012) # same result
```

```
# A tibble: 1 × 6
  BUYER_COUNTY BUYER_STATE  year count DOSAGE_UNIT countyfips
  <chr>        <chr>       <dbl> <dbl>       <dbl> <chr>
1 FAYETTE      GA           2012 12978     3701320 13113
```

Subset rows of a data frame: dplyr

If you want OR statements (meaning the data can meet either condition does not need to meet both), you need to use `|` between conditions:

```
filter(AD, count > 10000 | year == 2012)
```

```
# A tibble: 8 × 6
  BUYER_COUNTY BUYER_STATE  year count DOSAGE_UNIT countyfips
  <chr>        <chr>       <dbl> <dbl>       <dbl> <chr>
1 DOUGLAS      CO           2013 20961     6277640 08035
2 FAYETTE      GA           2012 12978     3701320 13113
3 ATCHISON     MO           2012   601      204700 29005
4 BUTTE        CA           2006 20443    13802710 06007
5 PUEBLO       CO           2008 23932     9386790 08101
6 LEE          KY           2012  1654     1196860 21129
7 ROGERS       OK           2008 11150     4499860 40131
8 CABELL       WV           2013 18481     7806600 54011
```

Subset rows of a data frame: dplyr

The `%in%` operator can be used find values from a pre-made list (using `c()`) for a **single column** at a time.

```
filter(AD,  BUYER_STATE %in% c("CO","NM","GA"))
```

```
# A tibble: 4 × 6
  BUYER_COUNTY BUYER_STATE  year count DOSAGE_UNIT countyfips
  <chr>        <chr>       <dbl> <dbl>       <dbl> <chr>
1 DOUGLAS      CO           2013 20961     6277640 08035
2 FAYETTE      GA           2012 12978     3701320 13113
3 LUMPKIN      GA           2007  2239      567260 13187
4 PUEBLO       CO           2008 23932     9386790 08101
```

```
filter(AD,  BUYER_STATE == "CO"| BUYER_STATE == "NM"| BUYER_STATE == "GA") #equivalent
```

```
# A tibble: 4 × 6
  BUYER_COUNTY BUYER_STATE  year count DOSAGE_UNIT countyfips
  <chr>        <chr>       <dbl> <dbl>       <dbl> <chr>
1 DOUGLAS      CO           2013 20961     6277640 08035
2 FAYETTE      GA           2012 12978     3701320 13113
3 LUMPKIN      GA           2007  2239      567260 13187
4 PUEBLO       CO           2008 23932     9386790 08101
```

Subset rows of a data frame: dplyr

The `%in%` operator can be used find values from a pre-made vector (using `c()`) for a **single column** at a time.

```
filter(AD, year %in% c(2012,2014), BUYER_STATE %in% c("GA","CO"))
```

```
# A tibble: 1 × 6
  BUYER_COUNTY BUYER_STATE  year count DOSAGE_UNIT countyfips
  <chr>        <chr>       <dbl> <dbl>       <dbl> <chr>
1 FAYETTE      GA           2012 12978     3701320 13113
```

Be careful with column names and `filter`

This will not work the way you might expect! Best to stick with nothing but the column name if it is a typical name.

```
filter(AD, "year" > 2014)
```

```
# A tibble: 30 × 6
   BUYER_COUNTY      BUYER_STATE  year count DOSAGE_UNIT countyfips
   <chr>             <chr>       <dbl> <dbl>       <dbl> <chr>
 1 EASTLAND          TX           2006  1980      723690 48133
 2 FALLS CHURCH CITY VA           2014  1340      531270 51610
 3 DOUGLAS           CO           2013 20961     6277640 08035
 4 FAYETTE           GA           2012 12978     3701320 13113
 5 STAFFORD          VA           2009  7921     2904600 51179
 6 GILES             TN           2010  4210     2074530 47055
 7 ANDREW            MO           2010  1167      315180 29003
 8 LUNENBURG         VA           2010   763      246130 51111
 9 HAWKINS           TN           2009  7148     3420480 47073
10 ATCHISON          MO           2012   601      204700 29005
# ℹ 20 more rows
```

Don’t use quotes for atypical names

Atypical names are those with punctuation, spaces, start with a number, or are just a number.

```
AD_rename <- rename(AD, `year!` = year)

filter(AD_rename, "year!" > 2013) # will not work correctly
```

```
# A tibble: 30 × 6
   BUYER_COUNTY      BUYER_STATE `year!` count DOSAGE_UNIT countyfips
   <chr>             <chr>         <dbl> <dbl>       <dbl> <chr>
 1 EASTLAND          TX             2006  1980      723690 48133
 2 FALLS CHURCH CITY VA             2014  1340      531270 51610
 3 DOUGLAS           CO             2013 20961     6277640 08035
 4 FAYETTE           GA             2012 12978     3701320 13113
 5 STAFFORD          VA             2009  7921     2904600 51179
 6 GILES             TN             2010  4210     2074530 47055
 7 ANDREW            MO             2010  1167      315180 29003
 8 LUNENBURG         VA             2010   763      246130 51111
 9 HAWKINS           TN             2009  7148     3420480 47073
10 ATCHISON          MO             2012   601      204700 29005
# ℹ 20 more rows
```

Be careful with column names and `filter`

Using backticks works!

```
filter(AD_rename, `year!` > 2013)
```

```
# A tibble: 1 × 6
  BUYER_COUNTY      BUYER_STATE `year!` count DOSAGE_UNIT countyfips
  <chr>             <chr>         <dbl> <dbl>       <dbl> <chr>
1 FALLS CHURCH CITY VA             2014  1340      531270 51610
```

Be careful with column names and `filter`

```
filter(AD, "BUYER_STATE" == "CO") # this will not work
```

```
# A tibble: 0 × 6
# ℹ 6 variables: BUYER_COUNTY <chr>, BUYER_STATE <chr>, year <dbl>,
#   count <dbl>, DOSAGE_UNIT <dbl>, countyfips <chr>
```

Be careful with column names and `filter`

```
filter(AD, BUYER_STATE == "CO")# this works!
```

```
# A tibble: 2 × 6
  BUYER_COUNTY BUYER_STATE  year count DOSAGE_UNIT countyfips
  <chr>        <chr>       <dbl> <dbl>       <dbl> <chr>
1 DOUGLAS      CO           2013 20961     6277640 08035
2 PUEBLO       CO           2008 23932     9386790 08101
```

`filter()` is tricky

Try not use anything special for the column names in `filter()`. This is why it is good to not use atypical column names. Then you can just use the column name!

Always good to check each step!

Did the filter work the way you expected? Did the dimensions change?

https://media.giphy.com/media/5b5OU7aUekfdSAER5I/giphy.gif

GUT CHECK: If we want to keep just rows that meet either or two conditions, what code should we use?

A. `filter()` with `|`

B. `filter()` with `&`

Summary

- `pull()` to get values out of a data frame/tibble
- `select()` is the `tidyverse` way to get a tibble with only certain columns
- you can `select()` based on patterns in the column names
- you can also `select()` based on column class with the `where()` function
- you can combine multiple tidyselect functions together like `select(starts_with("C"), ends_with("state"))`
- you can combine multiple patterns with the `c()` function like `select(starts_with(c("A", "C")))` (see extra slides at the end for more info!)
- `filter()` can be used to filter out rows based on logical conditions
- avoid using quotes when referring to column names with `filter()`

Summary Continued

- `==` is the same as equivalent to
- `&` means both conditions must be met to remain after `filter()`
- `|` means either conditions needs to be met to remain after `filter()`

Lab Part 2

- Class Website
- Lab
- Day 3 Cheatsheet
- Posit's `dplyr` Cheatsheet

Get the data

```
#read_csv("https://jhudatascience.org/intro_to_r/data/annualDosage.csv")
set.seed(1234)
AD <-slice_sample(annualDosage, n = 30)
```

Combining `filter` and `select`

You can combine `filter` and `select` to subset the rows and columns, respectively, of a data frame:

```
select(filter(AD, year > 2012), BUYER_STATE)
```

```
# A tibble: 5 × 1
  BUYER_STATE
  <chr>
1 VA
2 CO
3 TX
4 IA
5 WV
```

Nesting

In R, the common way to perform multiple operations is to wrap functions around each other in a "nested" form.

```
head(select(AD, year, BUYER_STATE), 2)
```

```
# A tibble: 2 × 2
   year BUYER_STATE
  <dbl> <chr>
1  2006 TX
2  2014 VA
```

Nesting can get confusing looking

```
select(filter(AD, year > 2000 & BUYER_STATE == "CO"), year, count)
```

```
# A tibble: 2 × 2
   year count
  <dbl> <dbl>
1  2013 20961
2  2008 23932
```

Assigning Temporary Objects

One can also create temporary objects and reassign them:

```
AD_CO <- filter(AD, year > 2000 & BUYER_STATE == "CO")
AD_CO <- select(AD_CO, year, count)

head(AD_CO)

# A tibble: 2 × 2
   year count
  <dbl> <dbl>
1  2013 20961
2  2008 23932
```

Using the `pipe` (comes with `dplyr`):

The pipe `%>%` makes this much more readable. It reads left side "pipes" into right side. RStudio `CMD/Ctrl + Shift + M` shortcut. Pipe `AD` into `filter`, then pipe that into `select`:

```
AD %>% filter(year > 2000 & BUYER_STATE == "CO") %>% select(year, count)
```

```
# A tibble: 2 × 2
   year count
  <dbl> <dbl>
1  2013 20961
2  2008 23932
```

Alternative Pipes

There are multiple ways to write a pipe and you might see these (they work the same!):

```
|>
```

```
%>%
```

Note that `%>%` is not the same as `%in%` (which helps you check for a value in a vector).

Adding/Removing Columns

Adding columns to a data frame: dplyr (`tidyverse` way)

The `mutate` function in `dplyr` allows you to add or modify columns of a data frame.

```
# General format - Not the code!
{data object to update} <- mutate({data to use},
                          {new variable name} = {new variable source})
```

```
AD <- mutate(AD, newcol = count * 2)
head(AD, 4)
```

```
# A tibble: 4 × 7
  BUYER_COUNTY      BUYER_STATE  year count DOSAGE_UNIT countyfips newcol
  <chr>             <chr>       <dbl> <dbl>       <dbl> <chr>       <dbl>
1 EASTLAND          TX           2006  1980      723690 48133        3960
2 FALLS CHURCH CITY VA           2014  1340      531270 51610        2680
3 DOUGLAS           CO           2013 20961     6277640 08035       41922
4 FAYETTE           GA           2012 12978     3701320 13113       25956
```

Use mutate to modify existing columns

The `mutate` function in `dplyr` allows you to add or modify columns of a data frame.

```
# General format - Not the code!
{data object to update} <- mutate({data to use},
                          {variable name to change} = {variable modification})
```

```
AD <- mutate(AD, newcol = newcol / 2)
head(AD, 4)
```

```
# A tibble: 4 × 7
  BUYER_COUNTY      BUYER_STATE  year count DOSAGE_UNIT countyfips newcol
  <chr>             <chr>       <dbl> <dbl>       <dbl> <chr>       <dbl>
1 EASTLAND          TX           2006  1980      723690 48133        1980
2 FALLS CHURCH CITY VA           2014  1340      531270 51610        1340
3 DOUGLAS           CO           2013 20961     6277640 08035       20961
4 FAYETTE           GA           2012 12978     3701320 13113       12978
```

You can pipe data into mutate

```
AD <- AD %>% mutate(newcol = newcol / 2)
head(AD,4)
```

```
# A tibble: 4 × 7
  BUYER_COUNTY      BUYER_STATE  year count DOSAGE_UNIT countyfips newcol
  <chr>             <chr>       <dbl> <dbl>       <dbl> <chr>       <dbl>
1 EASTLAND          TX           2006  1980      723690 48133        990
2 FALLS CHURCH CITY VA           2014  1340      531270 51610        670
3 DOUGLAS           CO           2013 20961     6277640 08035      10480.
4 FAYETTE           GA           2012 12978     3701320 13113       6489
```

`mutate` function

"Artwork by @allison_horst". https://allisonhorst.com/

Removing columns of a data frame: dplyr

The `NULL` method is still very common.

The `select` function can remove a column with exclamation mark (`!`) our using the minus sign (`-`):

```
select(AD, !newcol)
```

```
# A tibble: 6 × 6
  BUYER_COUNTY      BUYER_STATE  year count DOSAGE_UNIT countyfips
  <chr>             <chr>       <dbl> <dbl>       <dbl> <chr>
1 EASTLAND          TX           2006  1980      723690 48133
2 FALLS CHURCH CITY VA           2014  1340      531270 51610
3 DOUGLAS           CO           2013 20961     6277640 08035
4 FAYETTE           GA           2012 12978     3701320 13113
5 STAFFORD          VA           2009  7921     2904600 51179
6 GILES             TN           2010  4210     2074530 47055
```

Or, you can simply select the columns you want to keep, ignoring the ones you want to remove.

Removing columns in a data frame: dplyr

You can use `c()` to list the columns to remove.

Remove `newcol` and `drat`:

```
select(AD, !c(newcol, year))
```

```
# A tibble: 30 × 5
   BUYER_COUNTY      BUYER_STATE count DOSAGE_UNIT countyfips
   <chr>             <chr>       <dbl>       <dbl> <chr>
 1 EASTLAND          TX           1980      723690 48133
 2 FALLS CHURCH CITY VA           1340      531270 51610
 3 DOUGLAS           CO          20961     6277640 08035
 4 FAYETTE           GA          12978     3701320 13113
 5 STAFFORD          VA           7921     2904600 51179
 6 GILES             TN           4210     2074530 47055
 7 ANDREW            MO           1167      315180 29003
 8 LUNENBURG         VA            763      246130 51111
 9 HAWKINS           TN           7148     3420480 47073
10 ATCHISON          MO            601      204700 29005
# ℹ 20 more rows
```

Ordering columns

Ordering the columns of a data frame: dplyr

The `select` function can reorder columns.

```
head(AD, 2)
```

```
# A tibble: 2 × 7
  BUYER_COUNTY      BUYER_STATE  year count DOSAGE_UNIT countyfips newcol
  <chr>             <chr>       <dbl> <dbl>       <dbl> <chr>       <dbl>
1 EASTLAND          TX           2006  1980      723690 48133         990
2 FALLS CHURCH CITY VA           2014  1340      531270 51610         670
```

```
AD %>% select(year, count, BUYER_STATE, BUYER_COUNTY) %>%
head(2)
```

```
# A tibble: 2 × 4
   year count BUYER_STATE BUYER_COUNTY
  <dbl> <dbl> <chr>       <chr>
1  2006  1980 TX          EASTLAND
2  2014  1340 VA          FALLS CHURCH CITY
```

Ordering the columns of a data frame: dplyr

The `select` function can reorder columns. Put `newcol` first, then select the rest of columns:

```
select(AD, newcol, everything())
```

```
# A tibble: 3 × 7
  newcol BUYER_COUNTY      BUYER_STATE  year count DOSAGE_UNIT countyfips
   <dbl> <chr>             <chr>       <dbl> <dbl>       <dbl> <chr>
1   990  EASTLAND          TX           2006  1980      723690 48133
2   670  FALLS CHURCH CITY VA           2014  1340      531270 51610
3 10480. DOUGLAS           CO           2013 20961     6277640 08035
```

Ordering the columns of a data frame: dplyr

Put `year` at the end ("remove, everything, then add back in"):

```
select(AD, !year, everything(), year)
```

```
# A tibble: 3 × 7
  BUYER_COUNTY      BUYER_STATE count DOSAGE_UNIT countyfips newcol  year
  <chr>             <chr>       <dbl>       <dbl> <chr>       <dbl> <dbl>
1 EASTLAND          TX           1980      723690 48133         990   2006
2 FALLS CHURCH CITY VA           1340      531270 51610         670   2014
3 DOUGLAS           CO          20961     6277640 08035       10480.  2013
```

Ordering the column names of a data frame: alphabetically

Using the base R `order()` function.

```
order(colnames(AD))
```

```
[1] 1 2 4 6 5 7 3
```

```
AD %>% select(order(colnames(AD)))
```

```
# A tibble: 30 × 7
   BUYER_COUNTY      BUYER_STATE count countyfips DOSAGE_UNIT newcol  year
   <chr>             <chr>       <dbl> <chr>            <dbl>  <dbl> <dbl>
 1 EASTLAND          TX           1980 48133           723690   990   2006
 2 FALLS CHURCH CITY VA           1340 51610           531270   670   2014
 3 DOUGLAS           CO          20961 08035          6277640 10480.  2013
 4 FAYETTE           GA          12978 13113          3701320  6489   2012
 5 STAFFORD          VA           7921 51179          2904600  3960.  2009
 6 GILES             TN           4210 47055          2074530  2105   2010
 7 ANDREW            MO           1167 29003           315180   584.  2010
 8 LUNENBURG         VA            763 51111           246130   382.  2010
 9 HAWKINS           TN           7148 47073          3420480  3574   2009
10 ATCHISON          MO            601 29005           204700   300.  2012
# ℹ 20 more rows
```

Ordering the columns of a data frame: dplyr

In addition to `select` we can also use the `relocate()` function of dplyr to rearrange the columns for more complicated moves.

For example, let say we just wanted `year` to be before `BUYER_STATE`.

```
head(AD, 1)
```

```
# A tibble: 1 × 7
  BUYER_COUNTY BUYER_STATE  year count DOSAGE_UNIT countyfips newcol
  <chr>        <chr>       <dbl> <dbl>       <dbl> <chr>       <dbl>
1 EASTLAND     TX           2006  1980      723690 48133         990
```

```
tb_carb <- relocate(AD, year, .before = BUYER_STATE)
```

```
head(tb_carb, 1)
```

```
# A tibble: 1 × 7
  BUYER_COUNTY  year BUYER_STATE count DOSAGE_UNIT countyfips newcol
  <chr>        <dbl> <chr>       <dbl>       <dbl> <chr>       <dbl>
1 EASTLAND      2006 TX           1980      723690 48133         990
```

Ordering rows

Ordering the rows of a data frame: dplyr

The `arrange` function can reorder rows By default, `arrange` orders in increasing order:

```
arrange(AD, year)
```

```
# A tibble: 30 × 7
   BUYER_COUNTY     BUYER_STATE  year count DOSAGE_UNIT countyfips  newcol
   <chr>            <chr>       <dbl> <dbl>       <dbl> <chr>        <dbl>
 1 EASTLAND         TX           2006  1980      723690 48133        990
 2 BUTTE            CA           2006 20443    13802710 06007      10222.
 3 PHELPS           NE           2006   842      165100 31137        421
 4 BENTON           IN           2006   314      100370 18007        157
 5 NORTHWEST ARCTIC AK           2007     1         240 02188          0.5
 6 LUMPKIN          GA           2007  2239      567260 13187       1120.
 7 HANCOCK          IA           2007   393      131400 19081        196.
 8 OKEECHOBEE       FL           2008  5050     1980520 12093       2525
 9 PUEBLO           CO           2008 23932     9386790 08101      11966
10 ROGERS           OK           2008 11150     4499860 40131       5575
# ℹ 20 more rows
```

Ordering the rows of a data frame: dplyr

Use the `desc` to arrange the rows in descending order:

```
arrange(AD, desc(year))
```

```
# A tibble: 30 × 7
   BUYER_COUNTY      BUYER_STATE  year count DOSAGE_UNIT countyfips newcol
   <chr>             <chr>       <dbl> <dbl>       <dbl> <chr>       <dbl>
 1 FALLS CHURCH CITY VA           2014  1340      531270 51610         670
 2 DOUGLAS           CO           2013 20961     6277640 08035       10480.
 3 BROWN             TX           2013  4336     2992140 48049        2168
 4 WINNEBAGO         IA           2013  1297      397720 19189         648.
 5 CABELL            WV           2013 18481     7806600 54011        9240.
 6 FAYETTE           GA           2012 12978     3701320 13113        6489
 7 ATCHISON          MO           2012   601      204700 29005         300.
 8 LEE               KY           2012  1654     1196860 21129         827
 9 SAINT HELENA      LA           2011   320      164300 22091         160
10 POLK              NE           2011   253       73600 31143         126.
# ℹ 20 more rows
```

Ordering the rows of a data frame: dplyr

You can combine increasing and decreasing orderings:

```
arrange(AD, count, desc(year)) %>% head(n = 2)
```

```
# A tibble: 2 × 7
  BUYER_COUNTY     BUYER_STATE  year count DOSAGE_UNIT countyfips newcol
  <chr>            <chr>       <dbl> <dbl>       <dbl> <chr>       <dbl>
1 NORTHWEST ARCTIC AK           2007     1         240 02188         0.5
2 POLK             NE           2011   253       73600 31143       126.
```

```
arrange(AD, desc(year), count) %>% head(n = 2)
```

```
# A tibble: 2 × 7
  BUYER_COUNTY      BUYER_STATE  year count DOSAGE_UNIT countyfips newcol
  <chr>             <chr>       <dbl> <dbl>       <dbl> <chr>       <dbl>
1 FALLS CHURCH CITY VA           2014  1340      531270 51610         670
2 WINNEBAGO         IA           2013  1297      397720 19189         648.
```

Summary

- `select()` and `filter()` can be combined together
- you can do sequential steps in a few ways:
  1. nesting them inside one another using parentheses `()`
  2. creating intermediate data objects in between
  3. using pipes `%>%` (like “then” statements)
- `select()` and `relocate()` can be used to reorder columns
- `arrange()` can be used to reorder rows
- can remove rows with `filter()`
- can remove a column in a few ways:
  1. using `select()` with exclamation mark in front of column name(s)
  2. not selecting it (without exclamation mark)

Summary cont…

- `mutate()` can be used to create new variables or modify them

```
# General format - Not the code!
{data object to update} <- mutate({data to use},
                                  {new variable name} = {new variable source})

AD <- mutate(AD, newcol = count/2.2)
```

A note about base R:

The `$` operator is similar to `pull()`. This is the base R way to do this:

```
AD$year
```

```
 [1] 2006 2014 2013 2012 2009 2010 2010 2010 2009 2012 2010 2006 2007 2010 2011
[16] 2008 2009 2010 2007 2008 2006 2007 2013 2012 2011 2006 2013 2008 2013 2010
```

Although it is easier (for this one task), mixing and matching the `$` operator with tidyverse functions usually doesn't work. Therefore, we want to let you know about it in case you see it, but we suggest that you try working with the tidyverse way.

Adding new columns to a data frame: base R

You can add a new column (or modify an existing one) using the `$` operator instead of `mutate`.

Just want you to be aware of this as it is very common.

```
AD$newcol <- AD$count/2.2
head(AD,3)
```

```
# A tibble: 3 × 7
  BUYER_COUNTY      BUYER_STATE  year count DOSAGE_UNIT countyfips newcol
  <chr>             <chr>       <dbl> <dbl>       <dbl> <chr>       <dbl>
1 EASTLAND          TX           2006  1980      723690 48133        900 
2 FALLS CHURCH CITY VA           2014  1340      531270 51610        609.
3 DOUGLAS           CO           2013 20961     6277640 08035       9528.
```

Even though `$` is easier for creating new columns, `mutate` is really powerful, so it's worth getting used to.

Lab Part 3

- Class Website
- Lab
- Day 3 Cheatsheet
- Posit's `dplyr` Cheatsheet

Image by Gerd Altmann from Pixabay

Extra Slides

Multiple patterns with one tidyselect statement

Need to combine the patterns with the `c()` function.

```
# These do the same thing
select(AD, starts_with(c("B", "D")))
select(AD, starts_with("B"), starts_with("D"))
```

Nuances about `filter()`

```
test <- tibble(A = c(1,2,3,4), B = c(1,2,3,4))
test
```

```
# A tibble: 4 × 2
      A     B
  <dbl> <dbl>
1     1     1
2     2     2
3     3     3
4     4     4
```

```
# These are technically the same but >= is easier to read
# Separating can cause issues
filter(test,  B > 2 | B==2)
```

```
# A tibble: 3 × 2
      A     B
  <dbl> <dbl>
1     2     2
2     3     3
3     4     4
```

```
filter(test, B >= 2)
```

```
# A tibble: 3 × 2
      A     B
  <dbl> <dbl>
1     2     2
```

Order of operations for `filter()`

Order can matter. Think of individual statements separately first.

```
filter(test,  A>3 | B==2 & B>2) # A is greater than 3 or B is equal to 2 AND (th

# A tibble: 1 × 2
      A     B
  <dbl> <dbl>
1     4     4

filter(test,  A>3 & B>2 | B==2) # A is greater than 3 AND B is greater than 2 le

# A tibble: 2 × 2
      A     B
  <dbl> <dbl>
1     2     2
2     4     4
```

`which()` function

Instead of removing rows like filter, `which()` simply shows where the values occur if they pass a specific condition. We will see that this can be helpful later when we want to select and filter in more complicated ways.

```
which(select(AD, year) == 2014)
```

```
[1] 2
```

```
select(AD, year) == 2014 %>% head(10)
```

```
       year
 [1,] FALSE
 [2,]  TRUE
 [3,] FALSE
 [4,] FALSE
 [5,] FALSE
 [6,] FALSE
 [7,] FALSE
 [8,] FALSE
 [9,] FALSE
[10,] FALSE
[11,] FALSE
[12,] FALSE
[13,] FALSE
[14,] FALSE
[15,] FALSE
[16,] FALSE
[17,] FALSE
```

Remove a column in base R

```
AD$year <- NULL
```

Renaming Columns of a data frame: base R

We can use the `colnames` function to extract and/or directly reassign column names of `df`:

```
colnames(AD) # just prints
```

```
[1] "BUYER_COUNTY" "BUYER_STATE"  "year"         "count"        "DOSAGE_UNIT"
[6] "countyfips"   "newcol"
```

```
colnames(AD)[1:3] <- c("County", "State", "Year") # reassigns
head(AD)
```

```
# A tibble: 6 × 7
  County            State  Year count DOSAGE_UNIT countyfips newcol
  <chr>             <chr> <dbl> <dbl>       <dbl> <chr>       <dbl>
1 EASTLAND          TX     2006  1980      723690 48133         900
2 FALLS CHURCH CITY VA     2014  1340      531270 51610         609.
3 DOUGLAS           CO     2013 20961     6277640 08035        9528.
4 FAYETTE           GA     2012 12978     3701320 13113        5899.
5 STAFFORD          VA     2009  7921     2904600 51179        3600.
6 GILES             TN     2010  4210     2074530 47055        1914.
```

Subset rows of a data frame with indices:

Let's select **rows** 1 and 3 from `df` using brackets:

```
AD[ c(1, 3), ]
```

```
# A tibble: 2 × 7
  County   State  Year count DOSAGE_UNIT countyfips newcol
  <chr>    <chr> <dbl> <dbl>       <dbl> <chr>       <dbl>
1 EASTLAND TX     2006  1980      723690 48133         900
2 DOUGLAS  CO     2013 20961     6277640 08035        9528.
```

Subset columns of a data frame:

We can also subset a data frame using the bracket `[, ]` subsetting.

For data frames and matrices (2-dimensional objects), the brackets are `[rows, columns]` subsetting. We can grab the `x` column using the index of the column or the column name ("`year`")

```
AD[, 3]
```

```
# A tibble: 30 × 1
    Year
   <dbl>
 1  2006
 2  2014
 3  2013
 4  2012
 5  2009
 6  2010
 7  2010
 8  2010
 9  2009
10  2012
# ℹ 20 more rows
```

```
AD[, "count"]
```

```
# A tibble: 30 × 1
   count
   <dbl>
```

Subset columns of a data frame:

We can select multiple columns using multiple column names:

```
AD[, c("State", "count")]
```

```
# A tibble: 30 × 2
   State count
   <chr> <dbl>
 1 TX     1980
 2 VA     1340
 3 CO    20961
 4 GA    12978
 5 VA     7921
 6 TN     4210
 7 MO     1167
 8 VA      763
 9 TN     7148
10 MO      601
# ℹ 20 more rows
```