



Module 0: Introduction

Ian McCarthy | Emory University
Econ 470 & HLTH 470

Table of contents

1. Motivation
2. Syllabus highlights
3. Software installation
4. Starting with Git and RStudio
5. Tidy Data
6. Real World
7. Medicare Advantage

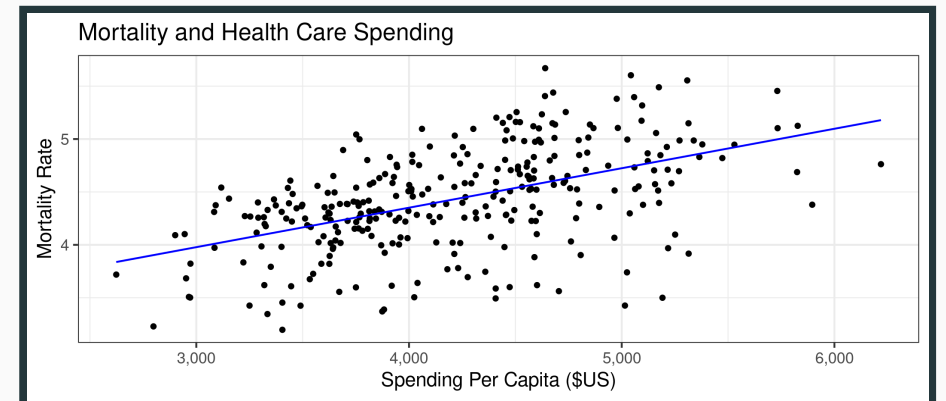
Motivation

Motivating question

Does health care spending improve health?

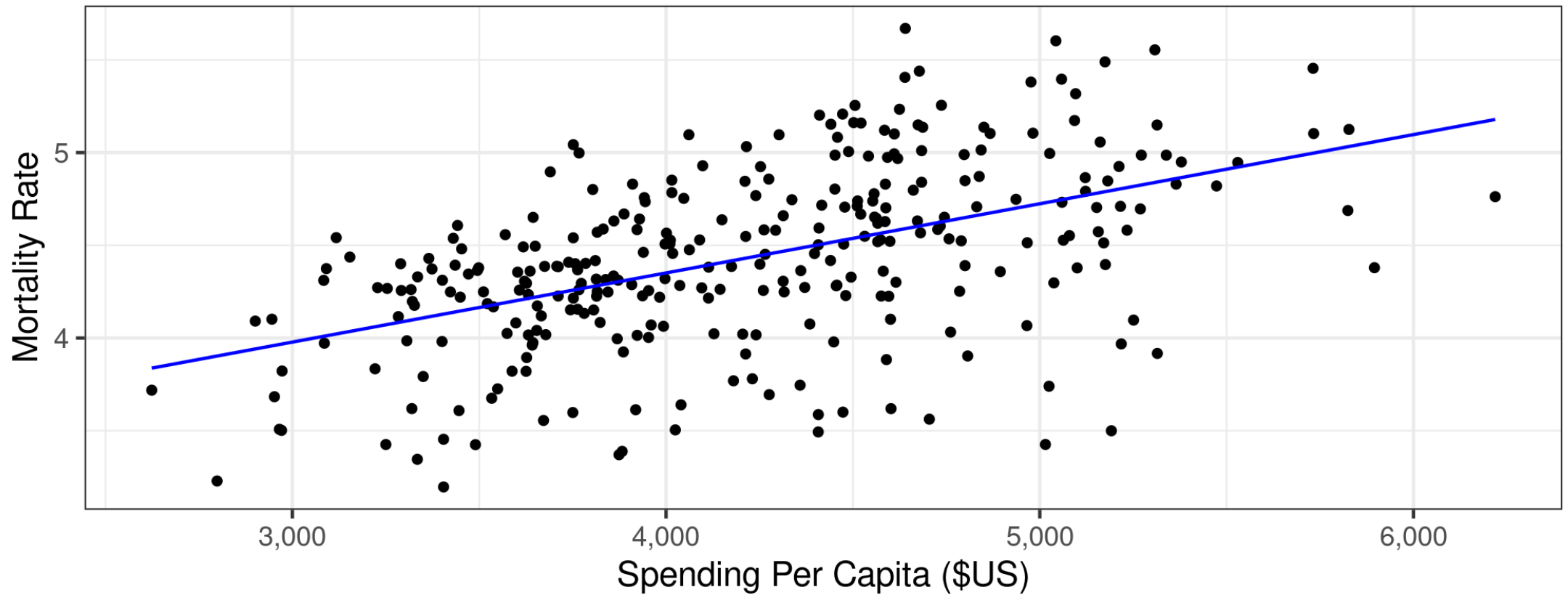
Spending and Health

```
ggplot(data = (dartmouth.data %>% filter(Year=2015)),  
       mapping = aes(x = Expenditures, y = Total_Mortality)) +  
  geom_point(size = 1) + theme_bw() + scale_x_continuous(label = comma) +  
  geom_smooth(method="lm", se=FALSE, color="blue", size=1/2) +  
  labs(x = "Spending Per Capita ($US)",  
       y = "Mortality Rate",  
       title = "Mortality and Health Care Spending")
```



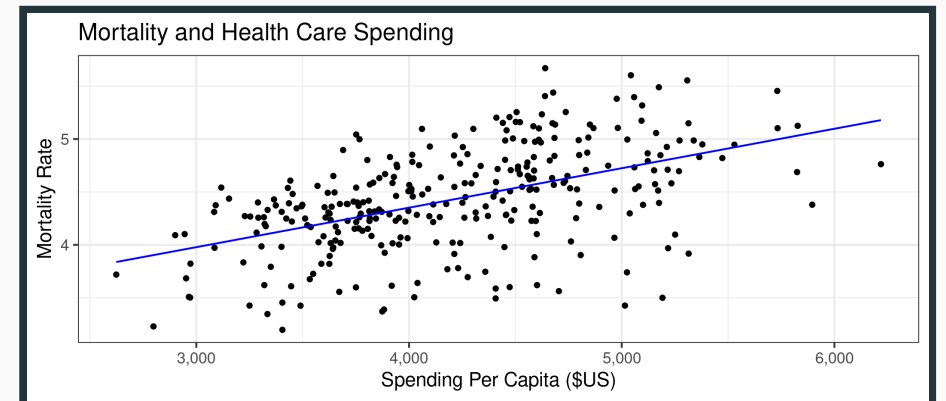
Spending and Health

Mortality and Health Care Spending



Spending and Health

- Does medical spending make us sicker?
- What else might explain this relationship?



Goals of this course.

1. Understand and implement selected methods for causal inference
2. Along the way...data management and version control with real data
3. Summarize, visualize, and explain research results

Syllabus highlights

(Read the full document [here.](#))

Why this course?

1. Major problems that need solutions
2. Need good, convincing empirical work for policy
3. Working with data is hard, particularly health care data
4. Your work should be transparent and reproducible

Structure

- Very applied in nature
- *Methods* for causal inference
 - Selection on observables (regression, re-weighting, matching, propensity scores)
 - Instrumental variables
 - Regression discontinuity
 - Difference-in-differences

Structure

- *Substantive* areas
 - Hospital pricing, policy, and competition
 - Cigarette taxes and demand
 - Medicare Advantage and quality disclosure
 - Medicaid expansion and health insurance

Structure

- *Datasets* from the real world
 - Hospital Cost Report Information System (HCRIS)
 - Centers for Disease Control (CDC)
 - Medicare Advantage data
 - Behavioral Risk Factor Surveillance System (BRFSS), Medicaid, Health Insurance Exchanges

Assignments

- Homework (x5)
- Research project
- Participation

Grading

Component	Weight
5 × homework assignments (11% each)	55%
Research project	40%
Participation	5%

Software Installation

Software Installation

1. Download **R**
2. Download **RStudio**
3. Download **Git**
4. Create an account on **GitHub**

For help and troubleshooting with Git and GitHub, take a look at Jenny Bryan's <http://happygitwithr.com>.

Checklist

☑ Do you have the most recent version of R?

```
version$version.string
```

```
## [1] "R version 4.1.2 (2021-11-01)"
```

☑ Do you have the most recent version of RStudio? (The **preview version** is fine.)

```
RStudio.Version()$version
```

```
## Requires an interactive session but should return something like "[1] '1.4.1717'"
```

☑ Have you updated all of your R packages?

```
update.packages(ask = FALSE, checkBuilt = TRUE)
```

Checklist

- Open up the **shell**
- Windows users, make sure that you installed a Bash-compatible version of the shell. If you installed **Git for Windows**, then you should be good to go.

Checklist

☑ Which version of Git have you installed?

```
git --version
```

☑ Did you introduce yourself to Git? (Substitute in your details.)

```
git config --global user.name 'Ian McCarthy'  
git config --global user.email 'ian.mccarthy@emory.edu'  
git config --global --list
```

☑ Did you register an account in GitHub?

Alternative setup...

Just use the cloud!

- We have our own virtual computer via AWS
- This computer has all the space you need and the data are already available
- Downside: the computer will be "on" during designated times only
- Details of this are on Canvas

Practice with Git and RStudio

Before next class (see <http://happygitwithr.com>)

1. Download **R**
2. Download **RStudio**
3. Download **Git**
4. Create an account on **GitHub**
5. Connect RStudio to Git and GitHub
6. Start/clone/fork a repository for this class

Setting things up

Now we're going to clone a GitHub repository (repo) using RStudio. The video below is from Grant McDermott's class.



Some common mistakes for windows users

- Windows folders are *not* files...there is no content without a file. You can't commit or push changes without content.
- Let RStudio/GitHub create the directory (main folder) for you.
- If you're working across devices on your own repo, be sure to pull before starting and push afterward.
- Avoid spaces in file names. Avoid them at all costs. *DO NOT PUT SPACES IN YOUR FILE NAMES.*

| *"A space in a file name is a space in your soul."*

Ideal workflow

Until you are a Git(Hub) expert...

1. Start project on GitHub (fork from another repo if needed)
2. Clone to desktop with RStudio
3. See <http://happygitwithr.com> for instructions on linking your local repo with a new upstream remote

Working with AWS

Let's do this live!

Tidy Data

The tidyverse

- Suite of packages collectively known as the tidyverse
- Different from `base` R in many ways
- The tidyverse with pipes¹ is more intuitive to me

¹ We'll talk about pipes very soon!

What is Tidy data?

Resources:

- Paper: *Tidy Data* (Hadley Wickham, 2014 JSS)
- Vignette: *Tidy data* (from the `tidyr` package)

Essentially:

1. Variables are columns
2. Observations are rows
3. Variables and observations make a table

Intro to Tidy data

Let's load the tidyverse package and check the output:

```
library(tidyverse)
```

Comes with lots of other packages like `ggplot2`, `tibble`, `dplyr`, etc.

Pipes: %>%

- The pipe operator is denoted `%>%` and is automatically loaded with the tidyverse.
- Pipes are awesome!

These next two lines of code do exactly the same thing.

```
mpg %>% filter(manufacturer="audi") %>% group_by(model) %>% summarise(hwy_mean = mean(hwy))  
summarise(group_by(filter(mpg, manufacturer="audi"), model), hwy_mean = mean(hwy))
```

The first line reads from left to right and from data to operation. The `Base R` version (line 2) works in the opposite order.

Pipes: %>%

Helps to break the pipes over several lines

```
mpg %>%  
  filter(manufacturer="audi") %>%  
  group_by(model) %>%  
  summarise(hwy_mean = mean(hwy))
```

```
## # A tibble: 3 × 2  
##   model      hwy_mean  
##   <chr>      <dbl>  
## 1 a4          28.3  
## 2 a4 quattro  25.8  
## 3 a6 quattro  24
```

The dplyr package:

1. `filter()`: Find or exclude certain rows
2. `arrange()`: Sort your observations
3. `select()`: Select specific variables
4. `mutate()`: Create new variables
5. `summarise()`: Collapse multiple rows into a single summary value

1) dplyr::filter()

Multiple filters separated by commas:

```
starwars %>%  
  filter(  
    species = "Human",  
    height ≥ 190  
  ) %>% head(5)
```

```
## # A tibble: 4 × 14  
##   name      height  mass hair_color skin_color eye_color birth_year sex  gender  
##   <chr>      <int> <dbl> <chr>      <chr>      <chr>      <dbl> <chr> <chr>  
## 1 Darth Va...    202   136 none       white      yellow      41.9 male  masculi...  
## 2 Qui-Gon ...    193    89 brown      fair       blue        92   male  masculi...  
## 3 Dooku         193    80 white      fair       brown      102   male  masculi...  
## 4 Bail Pre...    191   NA black      tan        brown       67   male  masculi...  
## # ... with 5 more variables: homeworld <chr>, species <chr>, films <list>,  
## #   vehicles <list>, starships <list>
```

1) dplyr::filter()

To remove missing observations, use `filter(!is.na(height))`. Also try the `drop_na()` function from `tidyr`.

2) dplyr::arrange()

Arrange in ascending order:

```
starwars %>%  
  arrange(birth_year) %>% head(5)
```

```
## # A tibble: 5 × 14  
##   name      height  mass hair_color skin_color eye_color birth_year sex  gender  
##   <chr>      <int> <dbl> <chr>      <chr>      <chr>      <dbl> <chr> <chr>  
## 1 Wicket S...    88    20 brown      brown      brown          8 male masculin  
## 2 IG-88        200   140 none       metal      red          15 none masculin  
## 3 Luke Sky...   172    77 blond      fair       blue          19 male masculin  
## 4 Leia Org...   150    49 brown      light      brown          19 fema... feminin  
## 5 Wedge An...   170    77 brown      fair       hazel          21 male masculin  
## # ... with 5 more variables: homeworld <chr>, species <chr>, films <list>,  
## #   vehicles <list>, starships <list>
```

2) dplyr::arrange()

Arrange descending order using `arrange(desc())`:

```
starwars %>%  
  arrange(desc(birth_year)) %>% head(5)
```

```
## # A tibble: 5 × 14  
##   name      height  mass hair_color skin_color eye_color birth_year sex   gender  
##   <chr>    <int> <dbl> <chr>      <chr>      <chr>      <dbl> <chr> <chr>  
## 1 Yoda        66    17 white      green      brown        896 male  masculi...  
## 2 Jabba ...   175  1358 <NA>      green-tan,... orange        600 herma... masculi...  
## 3 Chewba...   228   112 brown     unknown    blue         200 male  masculi...  
## 4 C-3PO       167    75 <NA>      gold       yellow        112 none  masculi...  
## 5 Dooku       193    80 white     fair       brown         102 male  masculi...  
## # ... with 5 more variables: homeworld <chr>, species <chr>, films <list>,  
## #   vehicles <list>, starships <list>
```

3) dplyr::select()

- Use commas to select multiple columns
- Use "first:last" for consecutive columns
- Deselect a column with "-"

3) dplyr::select()

```
starwars %>%  
  select(name:skin_color, species, -height) %>% head(5)
```

```
## # A tibble: 5 × 5  
##   name          mass hair_color skin_color species  
##   <chr>      <dbl> <chr>      <chr>      <chr>  
## 1 Luke Skywalker    77 blond      fair      Human  
## 2 C-3PO             75 <NA>      gold      Droid  
## 3 R2-D2             32 <NA>      white, blue Droid  
## 4 Darth Vader      136 none      white      Human  
## 5 Leia Organa       49 brown     light      Human
```


3) dplyr::select()

Rename within `select()`:

```
starwars %>%  
  select(alias=name, crib=homeworld, sex=gender) %>% head(5)
```

```
## # A tibble: 5 × 3  
##   alias      crib      sex  
##   <chr>      <chr>    <chr>  
## 1 Luke Skywalker Tatooine masculine  
## 2 C-3PO      Tatooine masculine  
## 3 R2-D2      Naboo     masculine  
## 4 Darth Vader Tatooine masculine  
## 5 Leia Organa Alderaan  feminine
```

3) dplyr::select()

Use `select(contains(PATTERN))` to find rows that contain some strings of interest

```
starwars %>%  
  select(name, contains("color")) %>% head(5)
```

```
## # A tibble: 5 × 4  
##   name          hair_color skin_color eye_color  
##   <chr>         <chr>      <chr>    <chr>  
## 1 Luke Skywalker blond      fair     blue  
## 2 C-3PO         <NA>      gold     yellow  
## 3 R2-D2         <NA>      white, blue red  
## 4 Darth Vader   none      white     yellow  
## 5 Leia Organa   brown     light     brown
```

Also look into the `stringr` package.

4) dplyr::mutate()

Create new variables with `mutate()`

```
starwars %>%  
  select(name, birth_year) %>%  
  mutate(dog_years = birth_year * 7) %>%  
  mutate(comment = paste0(name, " is ", dog_years, " in dog years. ")) %>% head(5)
```

```
## # A tibble: 5 × 4  
##   name          birth_year dog_years comment  
##   <chr>          <dbl>     <dbl> <chr>  
## 1 Luke Skywalker      19        133 Luke Skywalker is 133 in dog years.  
## 2 C-3PO              112        784 C-3PO is 784 in dog years.  
## 3 R2-D2              33        231 R2-D2 is 231 in dog years.  
## 4 Darth Vader       41.9       293.3 Darth Vader is 293.3 in dog years.  
## 5 Leia Organa        19        133 Leia Organa is 133 in dog years.
```

4) dplyr::mutate()

`mutate()` processes in order, so you can put dependent mutates one after another:

```
starwars %>%  
  select(name, birth_year) %>%  
  mutate(dog_years = birth_year * 7, ## Separate with a comma  
         comment = paste0(name, " is ", dog_years, " in dog years. ")) %>% head(5)
```

```
## # A tibble: 5 × 4  
##   name          birth_year dog_years comment  
##   <chr>          <dbl>     <dbl> <chr>  
## 1 Luke Skywalker    19         133 Luke Skywalker is 133 in dog years.  
## 2 C-3PO             112         784 C-3PO is 784 in dog years.  
## 3 R2-D2             33         231 R2-D2 is 231 in dog years.  
## 4 Darth Vader      41.9        293.3 Darth Vader is 293.3 in dog years.  
## 5 Leia Organa       19         133 Leia Organa is 133 in dog years.
```

4) dplyr::mutate()

Other handy ways to use `mutate()`:

```
starwars %>%  
  select(name, height) %>%  
  filter(name %in% c("Luke Skywalker", "Anakin Skywalker")) %>%  
  mutate(tall1 = height > 180) %>%  
  mutate(tall2 = ifelse(height > 180, "Tall", "Short")) ## Same effect, but can choose lab
```

```
## # A tibble: 2 × 4  
##   name          height tall1 tall2  
##   <chr>         <int> <lgl> <chr>  
## 1 Luke Skywalker    172 FALSE Short  
## 2 Anakin Skywalker    188 TRUE  Tall
```

4) dplyr::mutate()

Note the "scoped" variants of `mutate()` that work on a subset of variables:

- `mutate_all()` affects every variable
- `mutate_at()` affects named or selected variables
- `mutate_if()` affects variables that meet some criteria (e.g. are numeric)

These have since been replaced with `across()`. So you would use `mutate(across())` instead.

5) dplyr::summarise() with group_by()

```
starwars %>%  
  group_by(species, gender) %>%  
  summarise(mean_height = mean(height, na.rm = T)) %>% head(5)
```

```
## # A tibble: 5 × 3  
## # Groups:   species [5]  
##   species  gender  mean_height  
##   <chr>    <chr>      <dbl>  
## 1 Aleena   masculine      79  
## 2 Besalisk masculine    198  
## 3 Cerean   masculine    198  
## 4 Chagrian masculine    196  
## 5 Clawdite feminine    168
```

Note: `na.rm = T` is usually a good idea, otherwise your summary will be `NA` too.

5) dplyr::summarise()

```
starwars %>% group_by(species, gender) %>% summarise(across(where(is.numeric), mean, na.rm
```

```
## # A tibble: 5 × 5  
## # Groups:   species [5]  
##   species  gender    height  mass birth_year  
##   <chr>    <chr>    <dbl> <dbl>    <dbl>  
## 1 Aleena  masculine    79     15      NaN  
## 2 Besalisk masculine   198    102      NaN  
## 3 Cerean  masculine   198     82      92  
## 4 Chagrian masculine   196    NaN      NaN  
## 5 Clawdite feminine   168     55      NaN
```


Joining operations

Central feature of the `dplyr` package involves merging data from multiple tables with **join operations**.

- `inner_join(df1, df2)`
- `left_join(df1, df2)`
- `right_join(df1, df2)`
- `full_join(df1, df2)`
- `semi_join(df1, df2)`
- `anti_join(df1, df2)`

Joining operations

- For some simple examples, we'll need some data sets that come bundled with the `nycflights13` package.
- Load it now and then inspect these data frames in your own console.

```
library(nycflights13)  
flights  
planes
```

Left join

Let's perform a **left join** on the flights and planes datasets.

```
left_join(flights, planes) %>%  
  select(year, month, day, dep_time, arr_time, carrier, flight, tailnum, type, model) %>%
```

```
## # A tibble: 5 × 10
```

```
##   year month   day dep_time arr_time carrier flight tailnum type  model  
##   <int> <int> <int>   <int>   <int> <chr>   <int> <chr>   <chr> <chr>  
## 1  2013     1     1     517     830 UA      1545 N14228 <NA> <NA>  
## 2  2013     1     1     533     850 UA      1714 N24211 <NA> <NA>  
## 3  2013     1     1     542     923 AA      1141 N619AA <NA> <NA>  
## 4  2013     1     1     544    1004 B6       725 N804JB <NA> <NA>  
## 5  2013     1     1     554     812 DL       461 N668DN <NA> <NA>
```

Left join

`dplyr` guessed about which columns to join on (i.e. columns that share the same name). It also told us its choices:

```
## Joining, by = c("year", "tailnum")
```

Problem: the variable "year" does not have a consistent meaning across our joining datasets!

- *year of flight* versus *year of construction*

Left join

Luckily, there's an easy way to avoid this problem.

- See if you can figure it out before turning to the next slide.
- Try `?dplyr::join`.

Left join

Let's be more explicit with the `by =` argument:

```
left_join(  
  flights,  
  planes %>% rename(year_built = year), ## Not necessary w/ below line, but helpful  
  by = "tailnum" ## Be specific about the joining column  
) %>%  
select(year, month, day, dep_time, arr_time, carrier, flight, tailnum, year_built, type,  
head(5) ## Just to save vertical space on the slide
```

```
## # A tibble: 5 × 11
```

```
##   year month   day dep_time arr_time carrier flight tailnum year_built type  
##   <int> <int> <int>   <int>   <int> <chr>   <int> <chr>      <int> <chr>  
## 1  2013     1     1     517     830 UA      1545 N14228     1999 Fixed w...  
## 2  2013     1     1     533     850 UA      1714 N24211     1998 Fixed w...  
## 3  2013     1     1     542     923 AA      1141 N619AA     1990 Fixed w...  
## 4  2013     1     1     544    1004 B6       725 N804JB     2012 Fixed w...
```

Be specific

What happens if we again specify the join column but don't rename the ambiguous "year"?

```
left_join(flights, planes, ## Not renaming "year" to "year_built" this time
  by = "tailnum") %>%
  select(contains("year"), month, day, dep_time, arr_time, carrier, flight, tailnum, type,
  head(5)
```

```
## # A tibble: 5 × 11
```

##	year.x	year.y	month	day	dep_time	arr_time	carrier	flight	tailnum	type	model
##	<int>	<int>	<int>	<int>	<int>	<int>	<chr>	<int>	<chr>	<chr>	<chr>
## 1	2013	1999	1	1	517	830	UA	1545	N14228	Fixe...	737-...
## 2	2013	1998	1	1	533	850	UA	1714	N24211	Fixe...	737-...
## 3	2013	1990	1	1	542	923	AA	1141	N619AA	Fixe...	757-...
## 4	2013	2012	1	1	544	1004	B6	725	N804JB	Fixe...	A320...
## 5	2013	1991	1	1	554	812	DL	461	N668DN	Fixe...	757-...

Other dplyr goodies

`pull()`: Extract a column from a data frame as a vector or scalar.

- e.g. `starwars %>% filter(gender="female") %>% pull(height)`

`count()` and `distinct()`: Number and isolate unique observations.

- e.g. `starwars %>% count(species)`, Or `starwars %>% distinct(species)`
- You could also use a combination of `mutate()`, `group_by()`, and `n()`, e.g. `starwars %>% group_by(species) %>% mutate(num = n())`
- Built-in combination using `add_count()`.

Other dplyr goodies

There is also a whole class of **window functions** for getting leads and lags, ranking, creating cumulative aggregates, etc.

See `vignette("window-functions")` for more.

Some dplyr tips

- Any `group_by()` statement stays until `ungroup()`
- Look out for `plyr` package. Do not use `plyr` and `dplyr` together. Just don't do it.



tidyr

Key tidyr verbs

1. `pivot_wider()` and `pivot_longer()` to reshape data between wide and long format
2. `separate()`: Split one column into multiple columns
3. `unite()`: Combine multiple columns into one

1) tidyr::pivot_longer()

```
stocks <- tibble(  
  time = as.Date('2009-01-01') + 0:1,  
  X = rnorm(2, 0, 1),  
  Y = rnorm(2, 0, 2),  
  Z = rnorm(2, 0, 4)  
)  
stocks
```

```
## # A tibble: 2 × 4  
##   time           X       Y       Z  
##   <date>       <dbl> <dbl> <dbl>  
## 1 2009-01-01  0.192 -1.62  -0.780  
## 2 2009-01-02 -0.907 -0.332  6.00
```

1) tidyr::pivot_longer()

```
tidy_stocks <- stocks %>%  
  pivot_longer(cols=c("X","Y","Z"),  
               names_to="stock", values_to="price")  
tidy_stocks
```

```
## # A tibble: 6 × 3  
##   time      stock price  
##   <date>    <chr> <dbl>  
## 1 2009-01-01 X      0.192  
## 2 2009-01-01 Y     -1.62  
## 3 2009-01-01 Z     -0.780  
## 4 2009-01-02 X     -0.907  
## 5 2009-01-02 Y     -0.332  
## 6 2009-01-02 Z      6.00
```

1) tidyr::pivot_longer()

Aside: Remembering the syntax

There's a long-running joke about no-one being able to remember Stata's "reshape" command. ([Exhibit A](#).)

It's easy to see this happening with `pivot_wider()` and `pivot_longer()` too.

1) tidyr::pivot_wider()

```
tidy_stocks %>%  
  pivot_wider(values_from="price", names_from="stock")
```

```
## # A tibble: 2 × 4  
##   time          X      Y      Z  
##   <date>      <dbl> <dbl> <dbl>  
## 1 2009-01-01  0.192 -1.62  -0.780  
## 2 2009-01-02 -0.907 -0.332  6.00
```


2) tidyr::separate()

```
economists <- tibble(  
  name = c("Abhijit Banerjee", "Esther Duflo", "Michael Kremer")  
)  
economists
```

```
## # A tibble: 3 × 1  
##   name  
##   <chr>  
## 1 Abhijit Banerjee  
## 2 Esther Duflo  
## 3 Michael Kremer
```

2) tidyr::separate()

```
economists %>% separate(name, c("first_name", "last_name"))
```

```
## # A tibble: 3 × 2
##   first_name last_name
##   <chr>      <chr>
## 1 Abhijit    Banerjee
## 2 Esther    Duflo
## 3 Michael    Kremer
```

Should also specify the separation character with `separate(... , sep=" ")`.

3) tidyr::separate_rows()

A related function is `separate_rows()` for splitting into new rows

```
jobs <- tibble(  
  name = c("Jack", "Jill"),  
  occupation = c("Homemaker", "Philosopher, Philanthropist, Troublemaker")  
)  
jobs
```

```
## # A tibble: 2 × 2  
##   name occupation  
##   <chr> <chr>  
## 1 Jack  Homemaker  
## 2 Jill  Philosopher, Philanthropist, Troublemaker
```

3) tidyr::separate_rows()

```
## Now split out Jill's various occupations into different rows  
jobs %>% separate_rows(occupation)
```

```
## # A tibble: 4 × 2  
##   name  occupation  
##   <chr> <chr>  
## 1 Jack  Homemaker  
## 2 Jill  Philosopher  
## 3 Jill  Philanthropist  
## 4 Jill  Troublemaker
```

4) tidyr::unite()

```
gdp <- data.frame(  
  yr = rep(2016, times = 4),  
  mnth = rep(1, times = 4),  
  dy = 1:4,  
  gdp = rnorm(4, mean = 100, sd = 2)  
)  
gdp
```

```
##      yr mnth dy      gdp  
## 1 2016     1  1  98.72684  
## 2 2016     1  2  98.91063  
## 3 2016     1  3  99.23657  
## 4 2016     1  4 105.45198
```

4) tidyr::unite()

```
## Combine "yr", "mnth", and "dy" into one "date" column  
gdp %>% unite(date, c("yr", "mnth", "dy"), sep = "-")
```

```
##      date      gdp  
## 1 2016-1-1 98.72684  
## 2 2016-1-2 98.91063  
## 3 2016-1-3 99.23657  
## 4 2016-1-4 105.45198
```

4) tidyr::unite()

`unite()` automatically creates character variable:

```
gdp_u ← gdp %>% unite(date, c("yr", "mnth", "dy"), sep = "-") %>% as_tibble()
gdp_u
```

```
## # A tibble: 4 × 2
##   date      gdp
##   <chr>    <dbl>
## 1 2016-1-1  98.7
## 2 2016-1-2  98.9
## 3 2016-1-3  99.2
## 4 2016-1-4 105.
```

Use `mutate()` with `lubridate` date functions to change the variable type.

4) tidyr::unite()

```
library(lubridate)  
gdp_u %>% mutate(date = ymd(date))
```

```
## # A tibble: 4 × 2  
##   date      gdp  
##   <date>    <dbl>  
## 1 2016-01-01  98.7  
## 2 2016-01-02  98.9  
## 3 2016-01-03  99.2  
## 4 2016-01-04 105.
```


Other tidyr goodies

- `drop_na()` to drop missing values among specified columns
- `fill()` to impute missing values from past/future values
- `replace_na()` to replace missing values with known value

Summary

dplyr

1. `filter()`
2. `arrange()`
3. `select()`
4. `mutate()`
5. `summarise()`

tidyr

1. `pivot_longer()`
2. `pivot_wider()`
3. `separate()`
4. `unite()`

Other useful items include: pipes (`%>%`), grouping (`group_by()`), joining functions (`left_join()`, `inner_join`, etc.).

Real World

Practice data versus the real world



Advice 1: Be patient and careful in your coding



Advice 2: Comment, comment, comment

You don't want to end up like this guy...



Medicare Advantage

Medicare Advantage

Let's work with the [Medicare Advantage GitHub repository](#)

Download the data

First step is to download the raw data that we'll be using.

- Monthly Enrollment
- Plan Characteristics
- Service Areas

Lots more out there, but this is enough for now.

1) Contract/enrollment info

```
for (y in 2006:2015) {  
  monthlist=get(paste0("monthlist_",y))  
  for (m in monthlist) {  
  
    ## Basic contract/plan information  
    ma.path=paste0(" ... CPSC_Contract_Info_",y,"_",m,".csv")  
    contract.info=read_csv(ma.path,  
                           skip=1,  
                           col_names = c("contractid","planid","org_type","plan_type",  
                                           "partd","snp","eghp","org_name","org_marketing_na",  
                                           "plan_name","parent_org","contract_date"),  
                           col_types = cols(  
                             contractid = col_character(),  
                             planid = col_double(),  
                             org_type = col_character(),  
                             plan_type = col_character(),  
                             partd = col_character(),  
                             snp = col_character(),
```

1) Contract/enrollment info

```
## Clean the contract level data  
contract.info = contract.info %>%  
  group_by(contractid, planid) %>%  
  mutate(id_count=row_number())  
  
contract.info = contract.info %>%  
  filter(id_count=1) %>%  
  select(-id_count)
```

1) Contract/enrollment info

```
## Enrollments per plan
```

```
ma.path=paste0(" ... CPSC_Enrollment_Info_",y,"_",m,".csv")
```

```
enroll.info=read_csv(ma.path,  
  skip=1,  
  col_names = c("contractid","planid","ssa","fips","state","county"  
  col_types = cols(  
    contractid = col_character(),  
    planid = col_double(),  
    ssa = col_double(),  
    fips = col_double(),  
    state = col_character(),  
    county = col_character(),  
    enrollment = col_double()  
  ),na="✖")
```

1) Contract/enrollment info

```
## Merge contract info with enrollment info
plan.data = contract.info %>%
  left_join(enroll.info, by=c("contractid", "planid")) %>%
  mutate(month=as.numeric(m),year=y)

assign(paste0("plan.data.",y,".",m),plan.data)
}
```

1) Contract/enrollment info

```
## Append monthly enrollment info for each year  
if (y==2006) {  
  plan.month=rbind(plan.data.07, plan.data.08, plan.data.09, plan.data.10,  
                   plan.data.11, plan.data.12)  
}  
else {  
  plan.month=rbind(plan.data.01, plan.data.02, plan.data.03, plan.data.04,  
                   plan.data.05, plan.data.06, plan.data.07, plan.data.08,  
                   plan.data.09, plan.data.10, plan.data.11, plan.data.12)  
}
```

1) Contract/enrollment info

```
## Fill in missing fips codes (by state and county)
```

```
plan.month = plan.month %>%  
  group_by(state, county) %>%  
  fill(fips)
```

```
## Fill in missing plan characteristics by contract and plan id
```

```
plan.month = plan.month %>%  
  group_by(contractid, planid) %>%  
  fill(plan_type, partd, snp, eghp, plan_name)
```

```
## Fill in missing contract characteristics by contractid
```

```
plan.month = plan.month %>%  
  group_by(contractid) %>%  
  fill(org_type, org_name, org_marketing_name, parent_org)
```

1) Contract/enrollment info

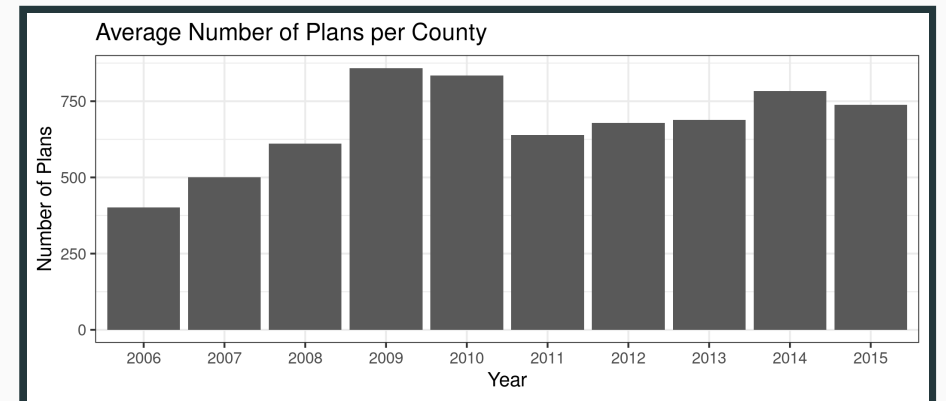
```
## Collapse from monthly data to yearly
plan.year = plan.month %>%
  group_by(contractid, planid, fips) %>%
  arrange(contractid, planid, fips, month) %>%
  summarize(avg_enrollment=mean(enrollment), sd_enrollment=sd(enrollment),
            min_enrollment=min(enrollment), max_enrollment=max(enrollment),
            first_enrollment=first(enrollment), last_enrollment=last(enrollment),
            state=last(state), county=last(county), org_type=last(org_type),
            plan_type=last(plan_type), partd=last(partd), snp=last(snp),
            eghp=last(eghp), org_name=last(org_name), org_marketing_name=last(org_marketing_name),
            plan_name=last(plan_name), parent_org=last(parent_org), contract_date=last(contract_date),
            year=last(year))

write_rds(plan.year, paste0(path.data.final, "/ma_data_", y, ".rds"))
}

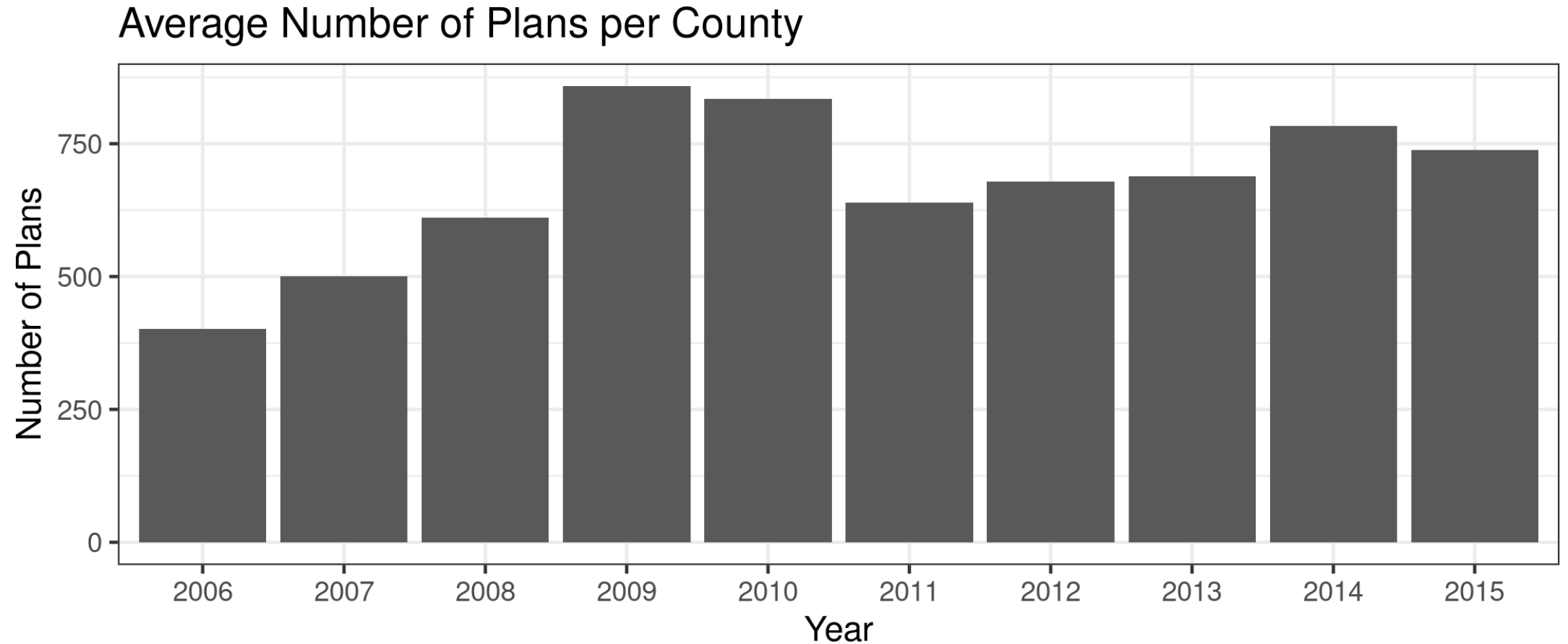
full.ma.data <- readRDS(paste0(path.data.final, "/ma_data_2006.rds"))
for (y in 2007:2015) {
  full.ma.data <- rbind(full.ma.data, paste0(path.data.final, "/ma_data_", y, ".rds"))
}
```


Number of plans available

```
full.ma.data %>% group_by(fips, year) %>% select(fips, year) %>% summarize(plan_count=n())  
  ggplot(aes(x=as.factor(year),y=plan_count)) +  
  stat_summary(fun.y="mean", geom="bar") +  
  labs(  
    x="Year",  
    y="Number of Plans",  
    title="Average Number of Plans per County"  
  ) + scale_y_continuous(labels=comma) +  
  theme_bw()
```



Number of plans available



Number of plans available

```
full.ma.data %>%  
  group_by(fips, year) %>%  
  select(fips, year) %>%  
  summarize(plan_count=n()) %>%  
  ggplot(aes(x=as.factor(year),y=plan_count)) +  
  stat_summary(fun.y="mean", geom="bar") +  
  labs(  
    x="Year",  
    y="Number of Plans",  
    title="Average Number of Plans per County"  
  ) + scale_y_continuous(labels=comma) +  
  theme_bw()
```

Number of plans available

```
full.ma.data %>%  
  filter(snp="No" & eghp="No") %>%  
  group_by(fips, year) %>%  
  select(fips, year) %>%  
  summarize(plan_count=n()) %>%  
  ggplot(aes(x=as.factor(year),y=plan_count)) +  
  stat_summary(fun.y="mean", geom="bar") +  
  labs(  
    x="Year",  
    y="Number of Plans",  
    title="Average Number of Plans per County"  
  ) + scale_y_continuous(labels=comma) +  
  theme_bw()
```

Number of plans available

```
full.ma.data %>%  
  filter(snp="No" & eghp="No") %>%  
  filter(planid < 800 | planid ≥ 900) %>%  
  filter(!is.na(planid)) %>%  
  group_by(fips, year) %>%  
  select(fips, year) %>%  
  summarize(plan_count=n()) %>%  
  ggplot(aes(x=as.factor(year),y=plan_count)) +  
  stat_summary(fun.y="mean", geom="bar") +  
  labs(  
    x="Year",  
    y="Number of Plans",  
    title="Average Number of Plans per County"  
  ) + scale_y_continuous(labels=comma) +  
  theme_bw()
```

Number of plans available

```
full.ma.data %>%  
  filter(snp="No" & eghp="No") %>%  
  filter(planid < 800 | planid ≥ 900) %>%  
  filter(!is.na(planid)) %>%  
  inner_join(service.area %>%  
    select(contractid, fips, year),  
    by=c("contractid", "fips", "year")) %>%  
  group_by(fips, year) %>%  
  select(fips, year) %>%  
  summarize(plan_count=n()) %>%  
  ggplot(aes(x=as.factor(year),y=plan_count)) +  
  stat_summary(fun.y="mean", geom="bar") +  
  labs(  
    x="Year",  
    y="Number of Plans",  
    title="Average Number of Plans per County"  
  ) + scale_y_continuous(labels=comma) +  
  theme_bw()
```

Interactive plot

