



Module 0: Introduction

Part 2: Basics of Data Management in R

Ian McCarthy | Emory University
Econ 470 & HLTH 470

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()

Common to use `filter()` to check or remove missing values

```
starwars %>%  
  filter(is.na(height)) %>% 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 Arvel C...    NA    NA brown      fair        brown          NA male mascul..  
## 2 Finn          NA    NA black      dark        dark          NA male mascul..  
## 3 Rey           NA    NA brown      light       hazel          NA fema... femini..  
## 4 Poe Dam...    NA    NA brown      light       brown          NA male mascul..  
## 5 BB8           NA    NA none       none        black          NA none  mascul..  
## # ... 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  masculi...  
## 2 IG-88         200   140 none       metal      red          15 none  masculi...  
## 3 Luke Sky...   172    77 blond      fair       blue         19 male  masculi...  
## 4 Leia Org...   150    49 brown      light      brown         19 fema... feminin...  
## 5 Wedge An...   170    77 brown      fair       hazel        21 male  masculi...  
## # ... 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) %>% summarise(across(where(is.numeric), mean, na.rm=TRUE))
```

```
## # A tibble: 5 × 4  
##   species  height  mass birth_year  
##   <chr>    <dbl> <dbl>      <dbl>  
## 1 Aleena      79     15        NaN  
## 2 Besalisk   198    102        NaN  
## 3 Cerean     198     82         92  
## 4 Chagrian   196    NaN        NaN  
## 5 Clawdite   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 functions

`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 functions

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.

Other summaries

- Summary statistics can get cumbersome with `summarize()`
- Quick summary of selected variables: `sumtable()` from the `vtable` package

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 -0.456 -2.89  -0.221  
## 2 2009-01-02  1.36   0.482 -6.97
```

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      -0.456  
## 2 2009-01-01 Y      -2.89  
## 3 2009-01-01 Z      -0.221  
## 4 2009-01-02 X       1.36  
## 5 2009-01-02 Y       0.482  
## 6 2009-01-02 Z      -6.97
```

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 -0.456 -2.89  -0.221  
## 2 2009-01-02  1.36   0.482 -6.97
```

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  99.04880  
## 2 2016    1  2 103.95492  
## 3 2016    1  3  99.72489  
## 4 2016    1  4 100.43369
```

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 99.04880  
## 2 2016-1-2 103.95492  
## 3 2016-1-3 99.72489  
## 4 2016-1-4 100.43369
```

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  99.0
## 2 2016-1-2 104.
## 3 2016-1-3  99.7
## 4 2016-1-4 100.
```

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  99.0
## 2 2016-01-02 104.
## 3 2016-01-03  99.7
## 4 2016-01-04 100.
```

Other tidyr functions

- `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](#)

Access the data

First step is to download the raw data that we'll be using, or work in AWS:

- 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(),  
                             ...  
                           ))  
  }  
}
```

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.",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"))
}
```

1) Contract/enrollment info

```
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"))  
}
```


All together now

Now let's move to AWS and try some of this together.

1. Clone repository
2. Create your own repository
3. Remove remote, `git remote rm origin`
4. Set new remote `git remote add origin NEW_URL` (where NEW_URL is your own repository)
5. `git push origin HEAD:master`
6. `git push --set-upstream origin master`

All together now

Next, you'll need to make some changes to get things to work:

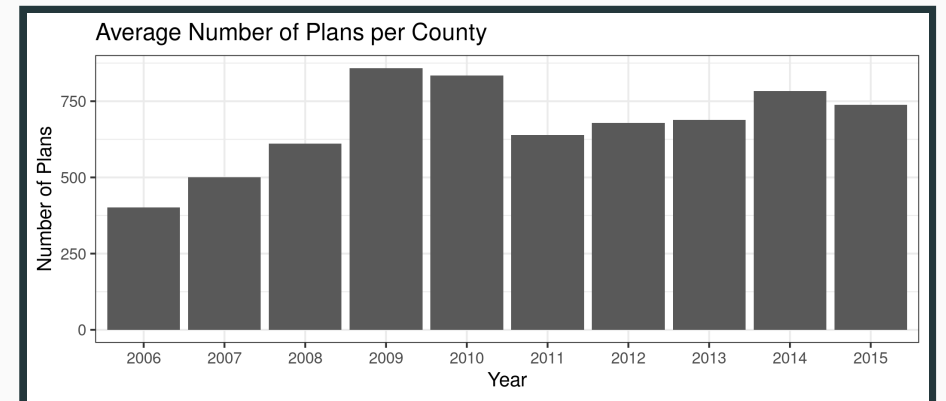
1. Create new `paths.R` script with:

```
path.code="R_code"  
path.data.ma="/data/medicare-advantage"  
path.data.ffs="/data/medicare-ffs"
```

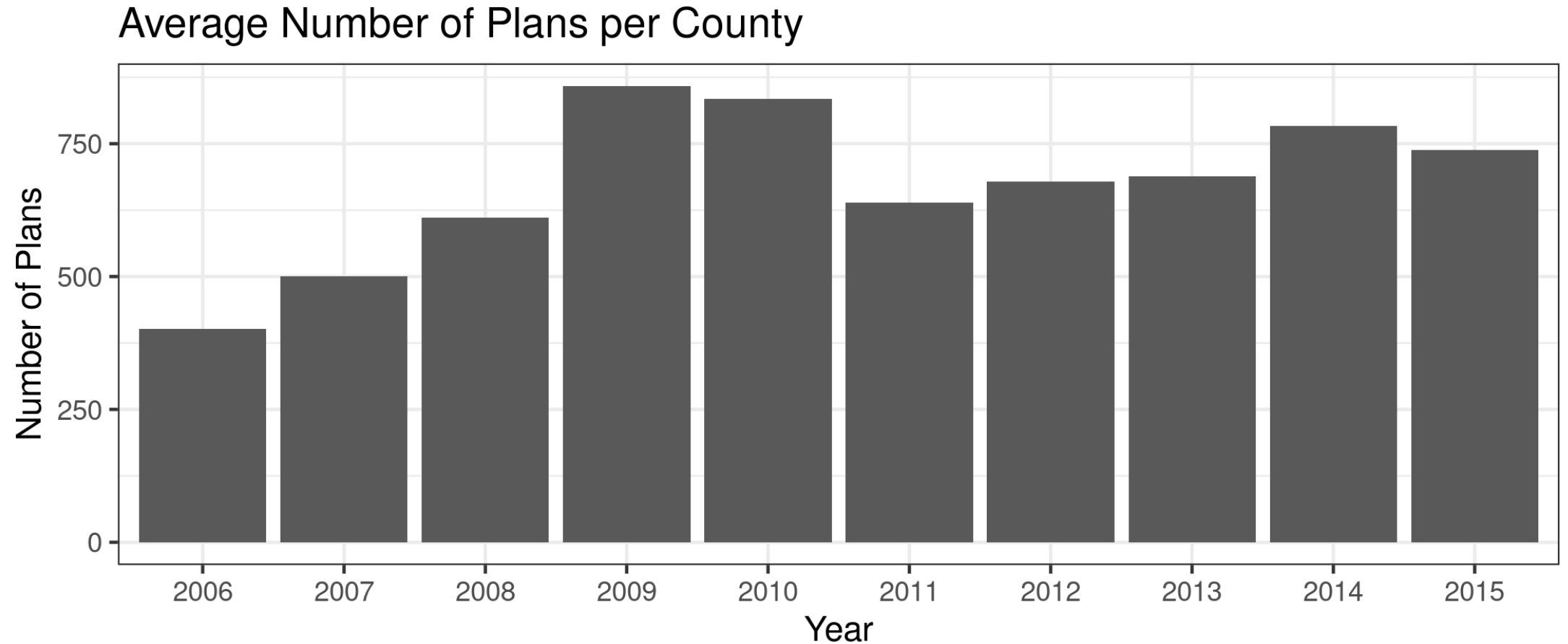
1. Update file paths
2. Remove "path.data.final" and rely on `here()` package

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()
```

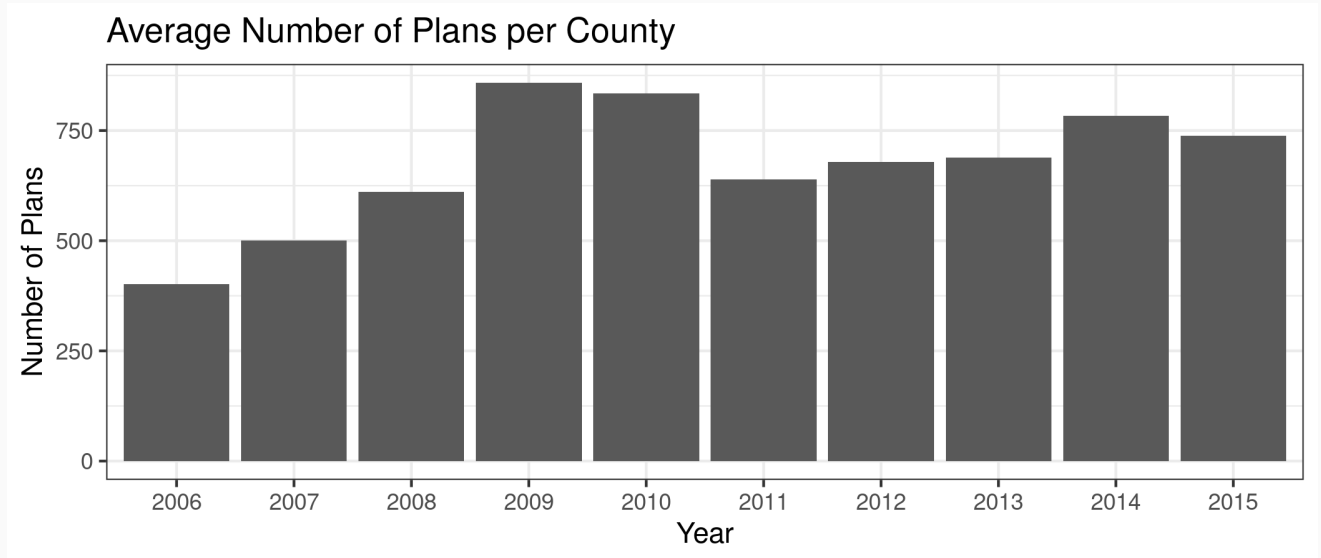


Number of plans available



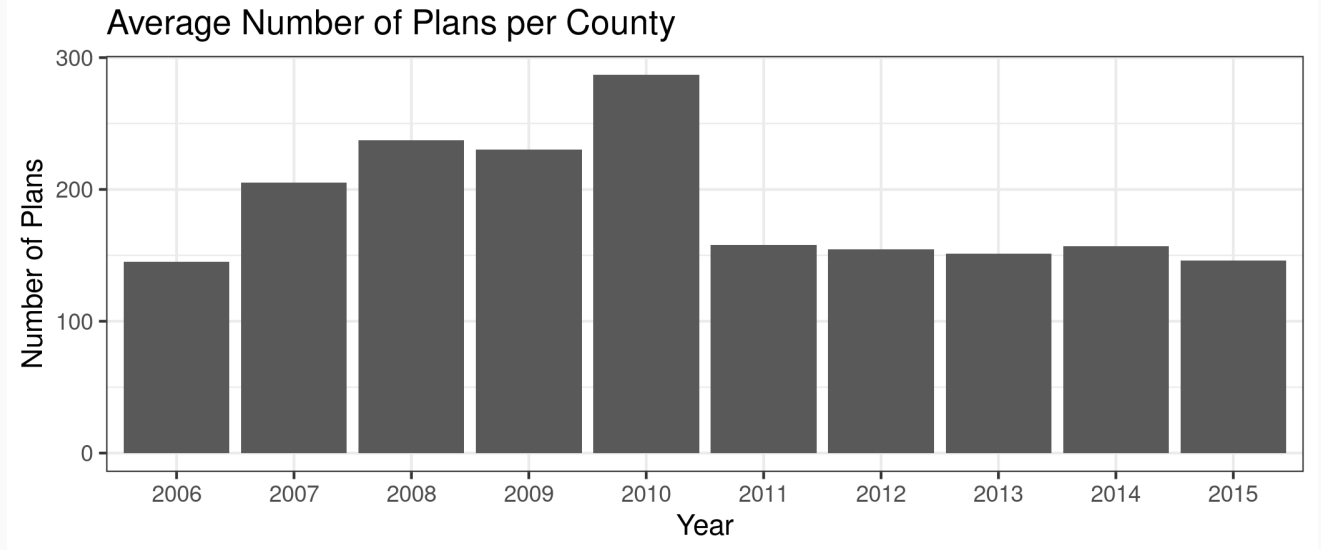
Number of plans available

```
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") %>%  
  scale_y_continuous(labels=comma) %>%  
  theme_bw()
```



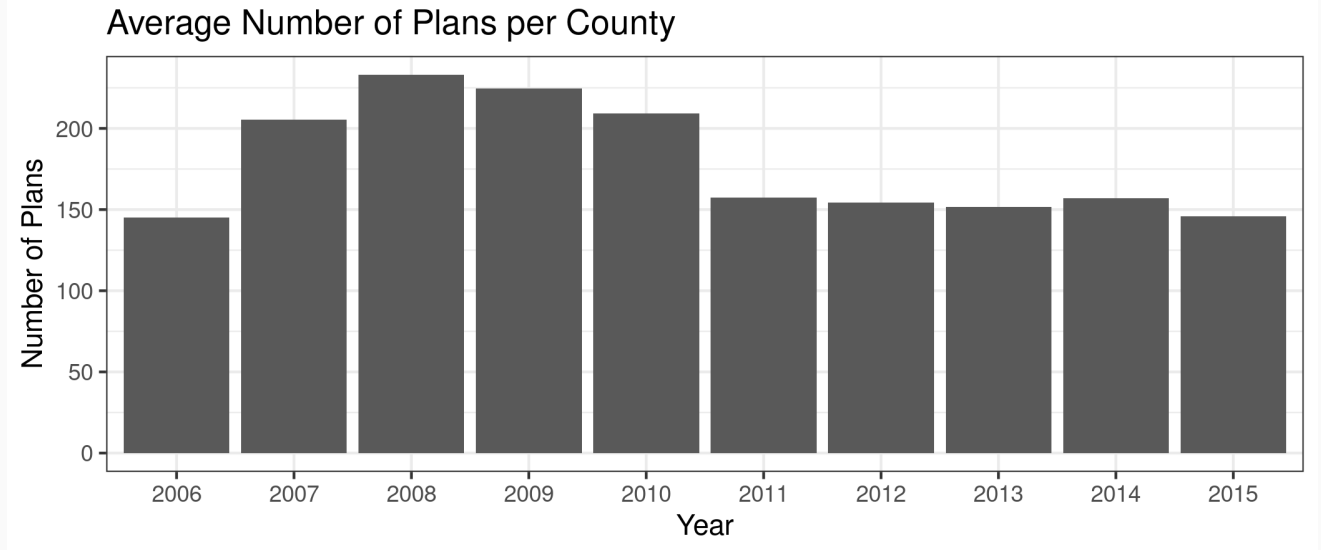
Number of plans available

```
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") %>%  
  scale_y_continuous(labels=comma) %>%  
  theme_bw()
```



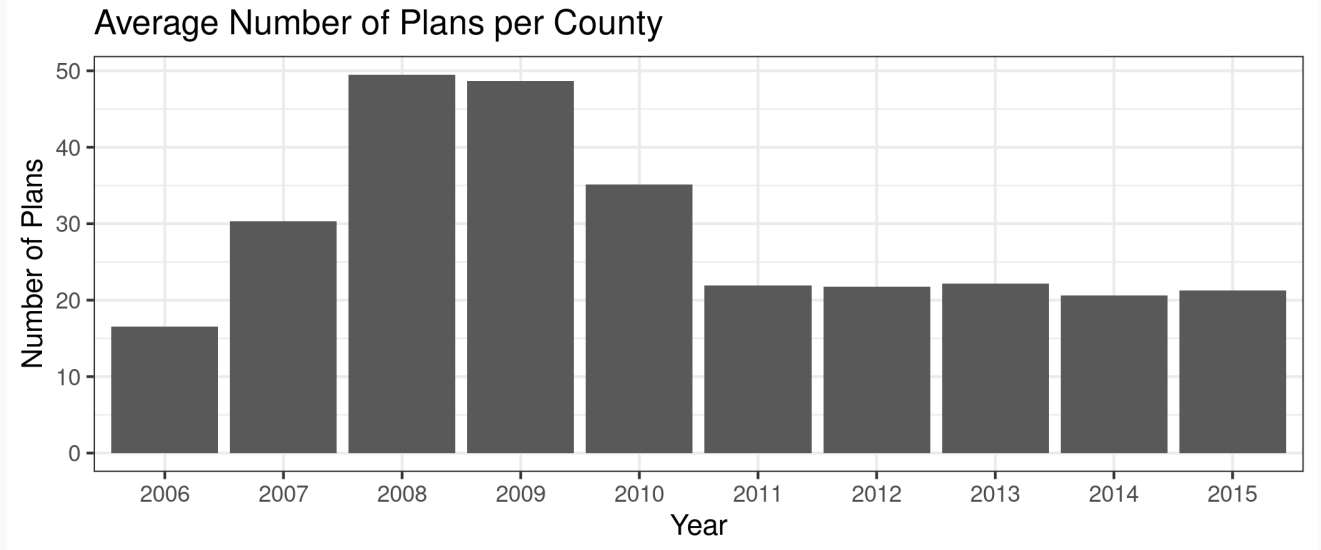
Number of plans available

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



Number of plans available

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



Interactive plot

