CSSS 508, Lecture 3

Manipulating and Summarizing Data

Michael Pearce
(based on slides from Chuck Lanfear)

October 13, 2022

A Quick Note

The last two lectures have been jam-packed with content! This makes it hard to answer individual-level questions as we get started with R and RStudio.

Luckily, today's lecture has **less content** and **more time for questions!** To keep things moving along,

- Please try to **limit very-individual questions while I'm running through slides** (e.g., "I'm getting this error message, why is that?"). I'm happy to answer these while we're working on exercises, during a break/lab/office hour, or before/after class.
- Please **interrupt me with questions about content during lecture!** If something isn't clear on the slides, it's best to discuss it right away!

Topics

Last time, we learned about,

1. Useful coding tips: packages, directories, and saving data
2. Advanced data manipulation tools
3. Basics of ggplot: layers and aesthetics
4. Advanced ggplot tools

Today, we will cover,

1. Subsetting data
2. Modifying data
3. Summarizing data
4. Joining (merging) data

Death to Spreadsheets

You may be familiar with tools like Excel or Google Sheets, which let you manipulate data in a spreadsheet using functions. Spreadsheets are *not reproducible*: It's hard to know how someone changed the raw data!

Today we'll talk more about `dplyr`: an R package that does just about anything Excel can, but more *transparently*, *reproducibly*, and *safely*.

Don't be the next sad research assistant who makes headlines with an Excel error (Reinhart & Rogoff, 2010)

Subsetting data

- `filter()`
- `distinct()`
- `select()`
- `pull()`

Reminder: Pipes (%>%)

In `dplyr`, we use the **pipe** operator (`%>%`) to make code readable. A keyboard shortcut to create pipes is `Ctrl+Shift+M` (Windows) or `Command+Shift+M` (OS).

Pipes take the object on the *left* and apply to it the function on the *right*:

```
library(dplyr)
library(gapminder)
gapminder %>% filter(country == "Canada") %>% head(2)
```

```
## # A tibble: 2 × 6
##   country continent  year lifeExp      pop gdpPercap
##   <fct>   <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Canada  Americas   1952    68.8 14785584    11367.
## 2 Canada  Americas   1957    70.0 17010154    12490.
```

In words, we (1) take the gapminder data, (2) filter to observations where the country is "Canada", and (3) display the first two observations.

Subsetting rows

We often get *big* datasets, but only want to use a portion of them. We can subset portions of our data using the `filter()` function.

Last week we used the `filter()` command to subset data:

```
Canada <- gapminder %>% filter(country == "Canada")
```

We now have the object `Canada` saved in our environment, which contains all observations of the gapminder data from the country Canada.

[1] Reminder: `==` is an operator that tests for equality.

Another Operator: `%in%`

What if I want to subset data from multiple countries at once?! Use the `%in%` operator!

We can use `%in%` like `==` but for matching *any element* in the vector on its right[1].

```
former_yugoslavia <- c("Bosnia and Herzegovina", "Croatia",
                       "Montenegro", "Serbia", "Slovenia")
Yugoslavia <- gapminder %>% filter(country %in% former_yugoslavia)
head(Yugoslavia, 4)
```

```
## # A tibble: 4 × 6
##   country                  continent  year lifeExp     pop gdpPercap
##   <fct>                    <fct>     <int>   <dbl>   <int>     <dbl>
## 1 Bosnia and Herzegovina Europe     1952    53.8 2791000      974.
## 2 Bosnia and Herzegovina Europe     1957    58.4 3076000     1354.
## 3 Bosnia and Herzegovina Europe     1962    61.9 3349000     1710.
## 4 Bosnia and Herzegovina Europe     1967    64.8 3585000     2172.
```

[1] Reminder: The `c()` function is how we make **vectors** in R.

Finding unique values

You may want to find the unique combinations of variables in a dataset. This can be performed using the function `distinct()`.

For example, if we want to see what unique combinations of "continent" and "year" are in gapminder, run:

```
gapminder %>% distinct(continent, year) %>% head(6)
```

```
## # A tibble: 6 × 2
##   continent  year
##   <fct>     <int>
## 1 Asia       1952
## 2 Asia       1957
## 3 Asia       1962
## 4 Asia       1967
## 5 Asia       1972
## 6 Asia       1977
```

`distinct()` drops variables!

By default, `distinct()` drops all unused variables. If you don't want to drop the others, use `distinct(.keep_all=TRUE)`:

```
gapminder %>% distinct(continent, year, .keep_all=TRUE) %>% head(6)
```

```
## # A tibble: 6 × 6
##   country     continent  year lifeExp      pop gdpPercap
##   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Afghanistan Asia       1952    28.8  8425333      779.
## 2 Afghanistan Asia       1957    30.3  9240934      821.
## 3 Afghanistan Asia       1962    32.0 10267083      853.
## 4 Afghanistan Asia       1967    34.0 11537966      836.
## 5 Afghanistan Asia       1972    36.1 13079460      740.
## 6 Afghanistan Asia       1977    38.4 14880372      786.
```

Keeping or dropping variables

What if we want to subset *variables* (as opposed to observations)? Use the `select()` function!

If we want to view the variables "country", "year", and "pop" (in that order), we can do so using the following code:

```
Yugoslavia %>% select(country, year, pop) %>% head(4)
```

```
## # A tibble: 4 × 3
##   country                  year     pop
##   <fct>                   <int>   <int>
## 1 Bosnia and Herzegovina   1952 2791000
## 2 Bosnia and Herzegovina   1957 3076000
## 3 Bosnia and Herzegovina   1962 3349000
## 4 Bosnia and Herzegovina   1967 3585000
```

Dropping columns

Alternatively, we can use `select()` to drop variables using a `-` sign:

```
Yugoslavia %>% select(-continent, -pop, -lifeExp) %>% head(4)
```

```
## # A tibble: 4 × 3
##   country                  year gdpPercap
##   <fct>                   <int>     <dbl>
## 1 Bosnia and Herzegovina  1952      974.
## 2 Bosnia and Herzegovina  1957     1354.
## 3 Bosnia and Herzegovina  1962     1710.
## 4 Bosnia and Herzegovina  1967     2172.
```

Now, we're showing all variables *except* "continent", "pop", and "lifeExp".

Extracting a single column?

Sometimes you want to extract a single column from a data frame as a *vector* (or single value). `pull()` *pulls* a column of a data frame out as a vector.

```
gapminder %>% pull(lifeExp) %>% head(4)
```

```
## [1] 28.801 30.332 31.997 34.020
```

```
gapminder %>% select(lifeExp) %>% head(4)
```

```
## # A tibble: 4 × 1
##   lifeExp
##     <dbl>
## 1    28.8
## 2    30.3
## 3    32.0
## 4    34.0
```

Note the difference between these two operations: The second yields only one column but is still a data frame.

In-Line `pull()`

`pull()` is particularly useful when you want to use a vector-only command in a `dplyr` chain of functions (say, in an in-line expression).

This in-line code...

```
The average life expectancy in Afghanistan from 1952 to 2007 was `r gapminder %>% filter(country=="Afghanistan") %>% pull(lifeExp) %>% mean() %>% round(1)` years.
```

... will produce this output:

The average life expectancy in Afghanistan from 1952 to 2007 was 37.5 years.

NOTE: `mean()` can only take a *vector* input, not a data frame. So this won't work with `select(lifeExp)` instead of `pull(lifeExp)`.

Check Your Understanding:

With a neighbor or two, write one line of code to answer each of the following questions:

1. Create an object that includes all rows in gapminder from the continents Asia and Oceania
2. Remove the variables "lifeExp" and "gdpPercap" from your subsetted data.
3. Display the distinct combinations of "country" and "continent" from your subsetted data, but do not drop the remaining variables!

My Solution

1: Filter Asia and Oceania:

```
Asia_and_Oceania <- gapminder %>%
  filter(continent %in% c("Asia", "Oceania"))
head(Asia_and_Oceania,4)
```

```
## # A tibble: 4 × 6
##   country     continent  year lifeExp      pop gdpPercap
##   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Afghanistan Asia       1952    28.8  8425333      779.
## 2 Afghanistan Asia       1957    30.3  9240934      821.
## 3 Afghanistan Asia       1962    32.0 10267083      853.
## 4 Afghanistan Asia       1967    34.0 11537966      836.
```

My Solution

2: Drop "lifeExp" and "gdpPercap":

```
Asia_and_Oceania <- Asia_and_Oceania %>% select(-lifeExp,-gdpPercap)
head(Asia_and_Oceania,4)
```

```
## # A tibble: 4 × 4
##   country     continent  year      pop
##   <fct>       <fct>     <int>    <int>
## 1 Afghanistan Asia       1952  8425333
## 2 Afghanistan Asia       1957  9240934
## 3 Afghanistan Asia       1962 10267083
## 4 Afghanistan Asia       1967 11537966
```

My Solution

3: Display distinct "country" and "continent" without dropping:

```
Asia_and_Oceania %>%
  distinct(country,continent,.keep_all=TRUE) %>%
  head(4)
```

```
## # A tibble: 4 × 4
##   country     continent  year      pop
##   <fct>       <fct>     <int>    <int>
## 1 Afghanistan Asia       1952  8425333
## 2 Australia   Oceania    1952  8691212
## 3 Bahrain     Asia       1952   120447
## 4 Bangladesh  Asia       1952 46886859
```

Modifying data

- `arrange()`
- `rename()`
- `mutate()`
- `recode()`

Sorting data by rows

Sometimes we want to sort data by row, in either ascending (low to high) or descending (high to low) order. We can do that with `arrange()`.

`arrange` uses ascending order by default. Arrange by descending order using the function `desc`.

For example, we can sort Yugoslavia first by year and then by population:

```
Yugoslavia %>% arrange(year, desc(pop)) %>% head(6)
```

```
## # A tibble: 6 × 6
##   country                  continent  year lifeExp     pop gdpPercap
##   <fct>                    <fct>     <int>   <dbl>   <int>     <dbl>
## 1 Serbia                   Europe     1952    58.0 6860147     3581.
## 2 Croatia                  Europe     1952    61.2 3882229     3119.
## 3 Bosnia and Herzegovina   Europe     1952    53.8 2791000      974.
## 4 Slovenia                 Europe     1952    65.6 1489518     4215.
## 5 Montenegro               Europe     1952    59.2  413834     2648.
## 6 Serbia                   Europe     1957    61.7 7271135     4981.
```

Rename variables

You may receive data with unintuitive variable names. You can change them using `rename()`.

```
Yugoslavia %>% select(country,year,lifeExp) %>%
  rename(Life_Expectancy = lifeExp) %>%
    head(4)
```

```
## # A tibble: 4 × 3
##   country                  year Life_Expectancy
##   <fct>                   <int>           <dbl>
## 1 Bosnia and Herzegovina  1952            53.8
## 2 Bosnia and Herzegovina  1957            58.4
## 3 Bosnia and Herzegovina  1962            61.9
## 4 Bosnia and Herzegovina  1967            64.8
```

(NOTE: I did *not* re-save the object Yugoslavia, so the name change is *not* permanent!)

Column Naming Practices

- *Good* column names are self-describing. Don't use inscrutable abbreviations to save typing, since RStudio can autocomplete.
- *Valid* column names can contain upper or lowercase letters, numbers, periods, and underscores. They must start with a letter or period and not be a special reserved word (e.g. `TRUE`, `if`).
- Names are case-sensitive: `Year` and `year` are not the same thing!
- You can include spaces if you put backticks around the name.

Column Name with Space Example

```
library(pander)
Yugoslavia %>% filter(country == "Serbia") %>%
    select(year, lifeExp) %>%
    rename(Year = year, `Life Expectancy` = lifeExp) %>%
    head(5) %>%
    pander(style = "rmarkdown", caption = "Serbian life expectancy")
```

Year	Life Expectancy
1952	58
1957	61.69
1962	64.53
1967	66.91
1972	68.7

Table: Serbian life expectancy

Create new columns

You can add new columns to a data frame using `mutate()`.

For example, we could add a new variable that provides the population in millions:

```
Yugoslavia %>% select(country, year, pop) %>%
    mutate(pop_million = pop / 1000000) %>%
    head(5)
```

```
## # A tibble: 5 × 4
##   country                  year     pop pop_million
##   <fct>                   <int>   <int>       <dbl>
## 1 Bosnia and Herzegovina   1952 2791000        2.79
## 2 Bosnia and Herzegovina   1957 3076000        3.08
## 3 Bosnia and Herzegovina   1962 3349000        3.35
## 4 Bosnia and Herzegovina   1967 3585000        3.58
## 5 Bosnia and Herzegovina   1972 3819000        3.82
```

Recoding variables

We've renamed *variables*, but what about variable *values*?

We can use the function `recode()` inside `mutate()`, which allows us to change specific values to others. This is best for categorical data.

```
Yugoslavia %>%
  mutate(country = recode(country,
                          `Bosnia and Herzegovina`="B and H",
                          Montenegro="M")) %>%
  distinct(country, .keep_all=TRUE)
```

```
## # A tibble: 5 × 6
##   country  continent  year lifeExp     pop gdpPercap
##   <fct>    <fct>     <int>   <dbl>   <int>     <dbl>
## 1 B and H  Europe     1952    53.8 2791000      974.
## 2 Croatia  Europe     1952    61.2 3882229     3119.
## 3 M        Europe     1952    59.2  413834     2648.
## 4 Serbia   Europe     1952    58.0 6860147     3581.
## 5 Slovenia Europe     1952    65.6 1489518     4215.
```

Check Your Understanding:

Try to answer the following questions on your own, then share your solutions with a neighbor

1. Sort the gapminder data by population in ascending order and print the first 5 rows. What's the country/year with the smallest population?

2. Filter the gapminder data to the countries "United States and "United Kingdom". Then, recode the country values to "US" and "UK", respectively. Print the unique combinations of country and continent.

My Solution

1: Sort by population

```
gapminder %>% arrange (pop) %>%  head(5)
```

```
## # A tibble: 5 × 6
##   country                continent  year lifeExp   pop gdpPercap
##   <fct>                  <fct>     <int>   <dbl> <int>     <dbl>
## 1 Sao Tome and Principe Africa     1952    46.5 60011      880.
## 2 Sao Tome and Principe Africa     1957    48.9 61325      861.
## 3 Djibouti              Africa     1952    34.8 63149     2670.
## 4 Sao Tome and Principe Africa     1962    51.9 65345     1072.
## 5 Sao Tome and Principe Africa     1967    54.4 70787     1385.
```

My Solution

2: Display US and UK

```
gapminder %>%
  filter(country %in% c("United States","United Kingdom")) %>%
  mutate(country = recode(country,
                          "United States" = "US",
                          "United Kingdom"="UK")) %>%
  distinct(country,continent)
```

```
## # A tibble: 2 × 2
##   country continent
##   <fct>   <fct>
## 1 UK      Europe
## 2 US      Americas
```

Summarizing data

1. `summarize()`
2. `group_by()`

Summarizing data

`summarize()` takes your column(s) of data and computes something using every row:

- Count how many rows there are
- Calculate the mean
- Compute the sum
- Obtain a minimum or maximum value

You can use any function inside `summarize()` that aggregates *multiple values* into a *single value* (like `sd()`, `mean()`, or `max()`).

`summarize()` Example

For the year 1982, let's get the number of observations, total population, mean life expectancy, and range of life expectancy for former Yugoslavian countries.

```
Yugoslavia %>% filter(year == 1982) %>%
    summarize(n_obs          = n(),
              total_pop      = sum(pop),
              mean_life_exp  = mean(lifeExp),
              range_life_exp = max(lifeExp) - min(lifeExp))
```

```
## # A tibble: 1 × 4
##   n_obs total_pop mean_life_exp range_life_exp
##   <int>     <int>         <dbl>          <dbl>
## 1     5  20042685          71.3           3.94
```

These new variables are calculated using *all of the rows* in `Yugoslavia`

Summarizing data by groups

What if we want to summarize data by category? For example, what's the population of countries in former Yugoslavia by year?

We can do this by applying the function `group_by()` and then `summarize()`.

Functions after `group_by()` are computed *within each group* as defined by variables given, rather than over all rows at once.

`group_by()` Example

```
Yugoslavia %>%
  group_by(year) %>%
  summarize(num_countries     = n_distinct(country),
            total_pop         = sum(pop)) %>%
  head(5)
```

```
## # A tibble: 5 × 3
##    year num_countries total_pop
##   <int>         <int>     <int>
## 1  1952             5  15436728
## 2  1957             5  16314276
## 3  1962             5  17099107
## 4  1967             5  17878535
## 5  1972             5  18579786
```

Because we did `group_by()` with `year` then used `summarize()`, we get *one row per value of* *`year`*!

Each value of year is its own **group**!

Check Your Understanding:

Try to answer the following questions on your own, then share your solutions with a neighbor.

1. Calculate the mean GDP for Canada, United States, and Mexico between 2000 and 2010. *HINT: Filter the data based on countries and years, then group the data by country, and then summarize.*

2. Plot the mean GDP for each country during the decade 2000-2010 using `ggplot`. Try to make sure the axis labels and limits look nice, and add a title.

My Solution

1: Mean GDP by Country in North America, 2000-2010.

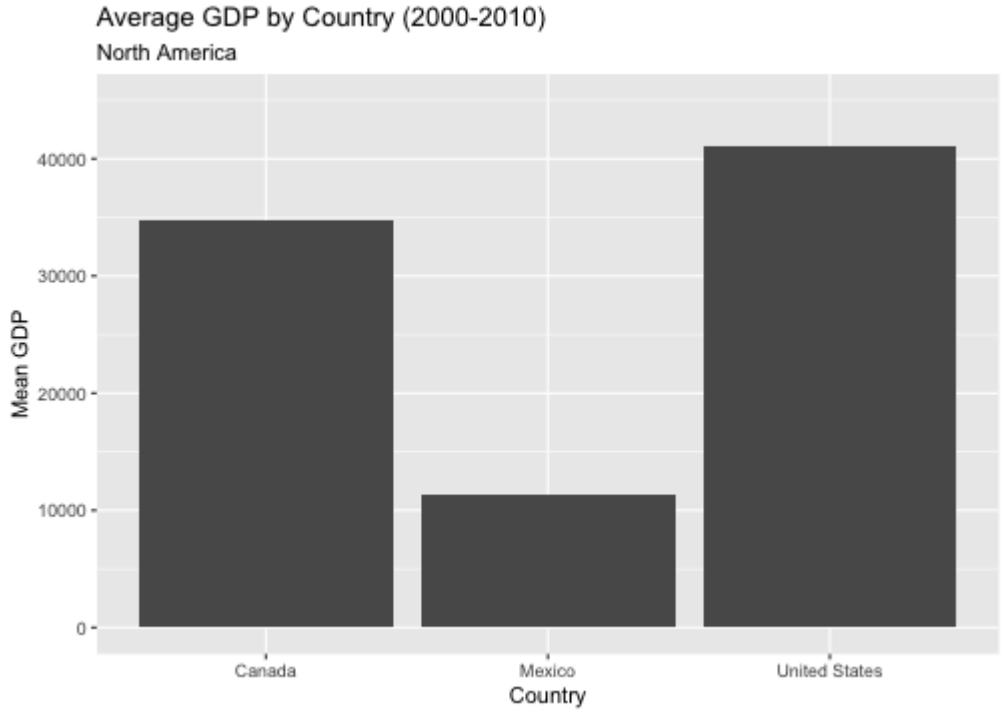
```
meanGDP_2000s <- gapminder %>%
  filter(country %in% c("Canada","United States","Mexico"),
         year > 2000 & year <= 2010) %>%
  group_by(country) %>%
  summarize(meanGDP = mean(gdpPercap))
meanGDP_2000s
```

```
## # A tibble: 3 × 2
##   country       meanGDP
##   <fct>           <dbl>
## 1 Canada         34824.
## 2 Mexico         11360.
## 3 United States  41024.
```

My Solution

2: Plot the data

```
library(ggplot2)
ggplot(meanGDP_2000s,aes(country,meanGDP)) +geom_col() +
  xlab("Country") + ylab("Mean GDP") + ylim(c(0,45000)) +
  ggtitle("Average GDP by Country (2000-2010)",subtitle = "North America")
```



Joining (Merging) Data

1. `left_join()`
2. `full_join()`

When Do We Need to Join Tables?

In practice, you may need to collect data from different sources. Merging those datasets can be tricky!

For example, imagine you would like to study county-level patterns with respect to age and grocery spending. However, you can only find,

- County level age data from the US Census, and
- County level grocery spending data from the US Department of Agriculture

Solution: Join the datasets!

Joining in Concept

When merging dataframes `A` and `B`, we think about:

- Which **rows** to keep from each data frame?
  - *Example: One row for each county*
- Which **columns** to keep from each data frame?
  - *Example: County name, mean age, mean grocery spending*
- Which variables determine whether rows **match**?
  - *Example: County name*

Join Types: Rows and columns kept

We'll focus on two types of joins:[1]...

- `A %>% left_join(B)`: keep all rows from `A`, matched with `B` wherever possible (`NA` when not), keep columns from both `A` and `B`
- `A %>% full_join(B)`: keep all rows from both `A` and `B`, matched wherever possible (`NA` when not), keep columns from both `A` and `B`

[1] Other types include `right_join`, `inner_join`, `semi_join`, and `anti_join`, but we won't study those here.

Matching Criteria

We say rows *match* when they have some columns containing the same value. We list these in a `by =` argument to the join.

Matching Behavior:

- `by = c("var1", "var2")`: Match on identical values of `var1` and `var2` in both `A` and `B`.
- `by = c("A_var1" = "B_var1", "A_var2" = "B_var2")`: Match identical values of `A_var1` variable in `A` to `B_var1` variable in `B`, and `A_var2` variable in `A` to `B_var2` variable in `B`.
- If you don't include `by`: Match using all variables in `A` and `B` that have identical names.

Note: If there are multiple matches, you'll get *one row for each possible combination*.

`nycflights13` Data

We'll use data in the `nycflights13` package. Install and load it:

```
# install.packages("nycflights13") # Uncomment to run
library(nycflights13)
```

It includes five data frames, some of which contain missing data (`NA`):

- `flights`: flights leaving JFK, LGA, or EWR in 2013
- `airlines`: airline abbreviations
- `airports`: airport metadata
- `planes`: airplane metadata
- `weather`: hourly weather data for JFK, LGA, and EWR

Note these are *separate data frames*, each needing to be *loaded separately*:

```
data(flights)
data(airlines)
data(airports)
data(planes)
data(weather)
```

Join Example #1

The "flights" data has carrier abbreviations, but not full names. That information is in the "airlines" data. Let's join them together!

```
flights %>% left_join(airlines, by = "carrier") %>%
  select(flight,origin,dest,carrier,name) %>%
  head(5)
```

```
## # A tibble: 5 × 5
##   flight origin dest  carrier name
##    <int> <chr>  <chr> <chr>   <chr>
## 1   1545 EWR    IAH   UA      United Air Lines Inc.
## 2   1714 LGA    IAH   UA      United Air Lines Inc.
## 3   1141 JFK    MIA   AA      American Airlines Inc.
## 4    725 JFK    BQN   B6      JetBlue Airways
## 5    461 LGA    ATL   DL      Delta Air Lines Inc.
```

We now have one row per flight, with both carrier abbreviations and full names!

Join Example #1 (cont.)

Which airlines had the most flights to Seattle?

```
flights %>% left_join(airlines, by = "carrier") %>%
  filter(dest == "SEA") %>%
  group_by(name) %>%
  summarize(num_flights = n())
```

```
## # A tibble: 5 × 2
##   name                  num_flights
##   <chr>                       <int>
## 1 Alaska Airlines Inc.          714
## 2 American Airlines Inc.        365
## 3 Delta Air Lines Inc.         1213
## 4 JetBlue Airways               514
## 5 United Air Lines Inc.        1117
```

Join Example #2

The "flights" data doesn't has plane manufacturer information, but "manufacturer" does. Both have the variable "tailnum". Let's join them together!

```
flights %>% left_join(planes, by = "tailnum") %>%
  select(flight,origin,dest,tailnum,manufacturer) %>%
  head(5)
```

```
## # A tibble: 5 × 5
##   flight origin dest  tailnum manufacturer
##    <int> <chr>  <chr> <chr>   <chr>
## 1   1545 EWR    IAH   N14228  BOEING
## 2   1714 LGA    IAH   N24211  BOEING
## 3   1141 JFK    MIA   N619AA  BOEING
## 4    725 JFK    BQN   N804JB  AIRBUS
## 5    461 LGA    ATL   N668DN  BOEING
```

We now have one row per flight, with plane manufacturer names!

Join Example #2 (cont.)

How many flights from JFK to Seattle were made by each manufacturer?

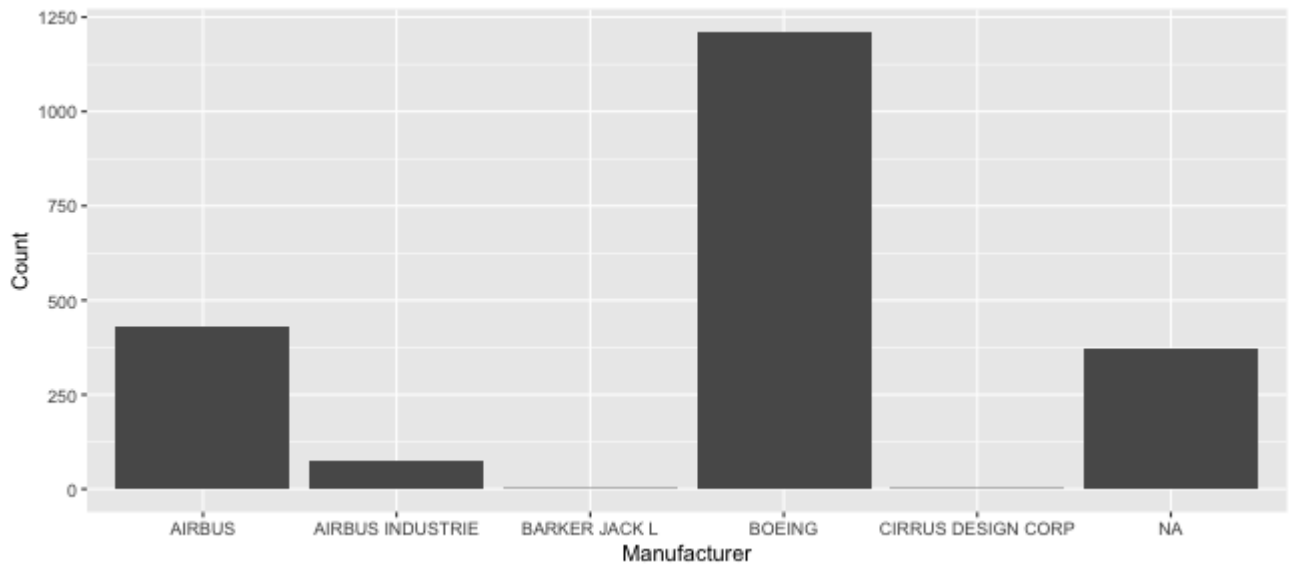
```
flights %>% left_join(planes, by = "tailnum") %>%
  filter(origin == "JFK",dest == "SEA") %>%
  group_by(manufacturer) %>%
  summarize(count_flights = n())
```

```
## # A tibble: 6 × 2
##   manufacturer       count_flights
##   <chr>                      <int>
## 1 AIRBUS                       430
## 2 AIRBUS INDUSTRIE              75
## 3 BARKER JACK L                  2
## 4 BOEING                      1211
## 5 CIRRUS DESIGN CORP             2
## 6 <NA>                         372
```

Join Example #2 (cont.)

Let's plot who manufactures the planes that flew from JFK to Seattle:

```
JFK_Seattle_manufacturers <- flights %>% left_join(planes, by = "tailnum") %>%
  filter(origin == "JFK",dest == "SEA") %>%
  group_by(manufacturer) %>% summarize(count_manufacturer = n())
ggplot(JFK_Seattle_manufacturers,aes(manufacturer,count_manufacturer))+
  geom_col()+xlab("Manufacturer")+ylab("Count")
```



Tinkering Suggestions

Some possible questions to investigate:

- What are the names of the most common destination airports?
- Which airlines fly from NYC to your home city?
- What is the distribution of departure times for flights leaving NYC over a 24 hour period?
    - Are especially late or early arrivals departures to some regions or for some airlines?

Warning: `flights` has 336776 rows, so if you do a sloppy join, you can end up with **many** matches per observation and have the data *explode* in size.

Homework 3

Pick something to look at in the `nycflights13` data and write up a .Rmd file showing your investigation. Upload both the .Rmd file and the .html file to Canvas.

You must:

1. Use each of the following functions at least once: `mutate()`, `summarize()`, `group_by()`, and `left_join()`.
2. Include at least one nicely formatted plot (`ggplot2`) and at least one nicely formatted table (`pander`). In plots and tables, use "nice" variable names (try out spaces!) and rounded values (<= 3 digits).
3. Briefly write down your question, observations from your analyses, and a summary of your work in words.

This time, *include all your code in your output document* (`echo=TRUE`), using comments and line breaks separating commands so that it is clear to a peer what you are doing (or trying to do!).