

Chapter 2

EXPLORATORY DATA ANALYSIS IN R

DATA ANALYSIS IN R

Outline

- Formulate your question
- Read in your data
- Check the packaging

Run `str()`

Look at the top and the bottom of your data

- Check your "n"s
- Validate with at least one external data source
- Try the easy solution first
- Challenge your solution
- Follow up questions

• *P.S: Dataset used in this chapter is the air pollution dataset from the U.S. Environmental Protection Agency (EPA) for the year 2019.*

Formulate your question

- A sharp question or hypothesis can serve as a dimension reduction tool that can eliminate variables that are not immediately relevant to the question.

- Example :

Are air pollution levels higher on the east coast than on the west coast?

But a more specific question might be

Are hourly ozone levels on average higher in New York City than they are in Los Angeles?

Which counties in the United States have the highest levels of ambient ozone pollution?

- Figure out what is the question you're really interested in and narrow it down to be as specific as possible (without becoming uninteresting).

Read in your data

- **The data are available from the EPA's Air Quality**
 - http://aqsdrl.epa.gov/aqsweb/aqstn1p/airdata/download_files.html
 - https://aqs.epa.gov/aqsweb/airdata/download_files.html
- **Simply download the zip file from the web site, unzip the archive, and put the resulting file in a directory called "data"**
- The dataset is a comma-separated value (CSV) file, where each row of the file contains one hourly measurement of ozone at some location in the country.
- Running the code below may take a few minutes. If it takes too long, you can read in a subset by specifying a value for the `n_max` argument to `read_csv()` that is greater than 0.

Read in your data

```
> library(readr)
> ozone <- read_csv("data/hourly_44201_2019.csv",
+ col_types = "cccccinnccccccnncnncccccc")
```

- The character string provided to the `col_types` argument specifies the class of each column in the dataset. Each letter represents the class of a column: “c” for character, “n” for numeric”, and “i” for integer.
- Just look quickly at the file to see what the column classes were.
- Rewrite the names of the columns to remove any spaces.

```
> names(ozone) <- make.names(names(ozone))
```

Check the packaging

- You have now an object in your workspace named `ozone`.
- Poke at that object:

Check the number of rows and columns.

```
> nrow(ozone)
[1] 9086926
> ncol(ozone)
[1] 24
```

- This dataset has relatively few columns, so you might be able to check the original text file to see if the number of columns printed out (24) here matches the number of columns you see in the original file.

Run str()

```
> str(ozone)
spc_tbl_ [9,086,926 × 24] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ State.Code      : chr [1:9086926] "01" "01" "01" "01" ...
 $ County.Code     : chr [1:9086926] "003" "003" "003" "003" ...
 $ Site.Num       : chr [1:9086926] "0010" "0010" "0010" "0010" ...
 $ Parameter.Code  : chr [1:9086926] "44201" "44201" "44201" "44201" ...
 $ POC            : int [1:9086926] 1 1 1 1 1 1 1 1 1 1 ...
 $ Latitude       : num [1:9086926] 30.5 30.5 30.5 30.5 30.5 ...
 $ Longitude      : num [1:9086926] -87.9 -87.9 -87.9 -87.9 -87.9 ...
 $ Datum         : chr [1:9086926] "NAD83" "NAD83" "NAD83" "NAD83" ...
 $ Parameter.Name  : chr [1:9086926] "Ozone" "Ozone" "Ozone" "Ozone" ...
 $ Date.Local     : chr [1:9086926] "2019-03-01" "2019-03-01" "2019-03-01" "2019-03-01" ...
 $ Time.Local     : chr [1:9086926] "01:00" "02:00" "03:00" "04:00" ...
 $ Date.GMT       : chr [1:9086926] "2019-03-01" "2019-03-01" "2019-03-01" "2019-03-01" ...
 $ Time.GMT       : chr [1:9086926] "07:00" "08:00" "09:00" "10:00" ...
 $ Sample.Measurement : num [1:9086926] 0.013 0.014 0.016 0.013 0.012 0.01 0.009 0.014 0.019 0.02
 ...
 $ Units.of.Measure : chr [1:9086926] "Parts per million" "Parts per million" "Parts per million"
 "Parts per million" ...
 $ MDL             : num [1:9086926] 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 ...
 $ Uncertainty     : num [1:9086926] NA NA NA NA NA NA NA NA NA NA ...
 $ Qualifier       : chr [1:9086926] NA NA NA NA ...
 $ Method.Type     : chr [1:9086926] "FEM" "FEM" "FEM" "FEM" ...
 $ Method.Code     : chr [1:9086926] "087" "087" "087" "087" ...
 $ Method.Name     : chr [1:9086926] "INSTRUMENTAL - ULTRA VIOLET ABSORPTION" "INSTRUMENTAL - UL
 TRA VIOLET ABSORPTION" "INSTRUMENTAL - ULTRA VIOLET ABSORPTION" "INSTRUMENTAL - ULTRA VIOLET ABSORP
 TION" ...
 $ State.Name      : chr [1:9086926] "Alabama" "Alabama" "Alabama" "Alabama" ...
 $ County.Name     : chr [1:9086926] "Baldwin" "Baldwin" "Baldwin" "Baldwin" ...
 $ Date.of.Last.Change: Date[1:9086926], format: "2019-07-03" "2019-07-03" "2019-07-03" ...
 - attr(*, "spec")=
 .. cols(
 .. `State Code` = col_character(),
 .. `County Code` = col_character(),
 .. `Site Num` = col_character(),
 .. `Parameter Code` = col_character(),
 .. POC = col_integer(),
 .. Latitude = col_number(),
 .. Longitude = col_number().
```

Look at the top and the bottom of your data

- **Look at the "beginning" and "end" of a dataset right after I check the packaging.**
 - This lets you know if the data were read in properly, things are properly formatted, and that everything is there.
 - If your data are time series data then make sure the dates at the beginning and end of the dataset match what you expect the beginning and ending time period to be.
 - Peek at the top and bottom of the data with the *head()* and *tail()* functions

Look at the top and the bottom of your data

- Make sure to check all the columns and verify that all of the data in each column looks the way it's supposed to look.
- Not a foolproof approach, because we're only looking at a few rows, but it's a decent start.

```
> head(ozone)
# A tibble: 6 x 24
  State.Code County.Code Site.Num Parameter.Code POC Latitude Longitude Datum Parameter.Name
  <chr>      <chr>      <chr>      <chr>      <int>   <dbl>   <dbl>   <chr>   <chr>
1 01         003        0010      44201         1    30.5   -87.9 NAD83 Ozone
2 01         003        0010      44201         1    30.5   -87.9 NAD83 Ozone
3 01         003        0010      44201         1    30.5   -87.9 NAD83 Ozone
4 01         003        0010      44201         1    30.5   -87.9 NAD83 Ozone
5 01         003        0010      44201         1    30.5   -87.9 NAD83 Ozone
6 01         003        0010      44201         1    30.5   -87.9 NAD83 Ozone
# i 15 more variables: Date.Local <chr>, Time.Local <chr>, Date.GMT <chr>, Time.GMT <chr>,
# Sample.Measurement <dbl>, Units.of.Measure <chr>, MDL <dbl>, Uncertainty <dbl>,
# Qualifier <chr>, Method.Type <chr>, Method.Code <chr>, Method.Name <chr>,
# State.Name <chr>, County.Name <chr>, Date.of.Last.Change <date>
> tail(ozone)
# A tibble: 6 x 24
  State.Code County.Code Site.Num Parameter.Code POC Latitude Longitude Datum Parameter.Name
  <chr>      <chr>      <chr>      <chr>      <int>   <dbl>   <dbl>   <chr>   <chr>
1 80         026        8012      44201         1    32.5   -115. WGS84 Ozone
2 80         026        8012      44201         1    32.5   -115. WGS84 Ozone
3 80         026        8012      44201         1    32.5   -115. WGS84 Ozone
4 80         026        8012      44201         1    32.5   -115. WGS84 Ozone
5 80         026        8012      44201         1    32.5   -115. WGS84 Ozone
6 80         026        8012      44201         1    32.5   -115. WGS84 Ozone
# i 15 more variables: Date.Local <chr>, Time.Local <chr>, Date.GMT <chr>, Time.GMT <chr>,
# Sample.Measurement <dbl>, Units.of.Measure <chr>, MDL <dbl>, Uncertainty <dbl>,
# Qualifier <chr>, Method.Type <chr>, Method.Code <chr>, Method.Name <chr>,
# State.Name <chr>, County.Name <chr>, Date.of.Last.Change <date>
>
# table(Season, Time.Local)
```

Check your "n"s

- **Count your observations**

- If you're expecting there to be 1,000 observations and it turns out there's only 20 $\Rightarrow$ something must have gone wrong somewhere

- **Identify some *landmarks* that can be used to check against your data.**

- If you are collecting data on people, such as in a survey or clinical trial, then you should know how many people there are in your study.

Check your "n"s

Example

- Assume that the dataset contains *hourly* data for the *entire country*. These will be our two landmarks for comparison.
 - Hourly ozone data that comes from monitors across the country: monitors should be monitoring continuously during the day so all hours should be represented.
 - Good representation of country of states.
- Take a look at the `Time.Local` variable to see what time measurements are recorded as being taken.

```
> table(ozone$Time.Local)
```

00:00	01:00	02:00	03:00	04:00	05:00	06:00	07:00	08:00	09:00	10:00	11:00	12:00
361076	361077	356237	359480	366309	379384	385409	382612	380056	377808	378259	379883	380606
13:00	14:00	15:00	16:00	17:00	18:00	19:00	20:00	21:00	22:00	23:00		
382935	384398	385956	386990	387499	387535	387729	387926	387190	380302	380270		

Check your "n"s

Example

➤ Good representation of country of states:

- Check the unique elements of the *State.Name* variable to see what's going on.
- This last bit of analysis made use of something we will discuss in the next section: external data. We knew that there are only 50 states in the U.S., so seeing 52 state names was an immediate trigger that something might be off.
- In this case, all was well, but validating your data with an external data source can be very useful.

```
> unique(ozone$State.Name)
[1] "Alabama"      "Alaska"        "Arizona"
[4] "Arkansas"     "California"    "Colorado"
[7] "Connecticut"  "Delaware"      "District Of Columbia"
[10] "Florida"      "Georgia"       "Hawaii"
[13] "Idaho"        "Illinois"      "Indiana"
[16] "Iowa"         "Kansas"        "Kentucky"
[19] "Louisiana"    "Maine"         "Maryland"
[22] "Massachusetts" "Michigan"      "Minnesota"
[25] "Mississippi"  "Missouri"      "Montana"
[28] "Nebraska"     "Nevada"        "New Hampshire"
[31] "New Jersey"   "New Mexico"    "New York"
[34] "North Carolina" "North Dakota" "Ohio"
[37] "Oklahoma"     "Oregon"        "Pennsylvania"
[40] "Rhode Island" "South Carolina" "South Dakota"
[43] "Tennessee"    "Texas"         "Utah"
[46] "Vermont"      "Virginia"      "Washington"
[49] "West Virginia" "Wisconsin"     "Wyoming"
[52] "Puerto Rico" "Country Of Mexico"
```

Validate with at least one external data source

- In the U.S. we have national ambient air quality standards, and for ozone, the current standard¹⁴ set in 2008 is that the "annual fourth-highest daily maximum 8-hr concentration, averaged over 3 years" should not exceed 0.075 parts per million (ppm).
 - ▶ The exact details of how to calculate this are not important for this analysis, but roughly speaking the 8-hour average concentration should not be too much higher than 0.075 ppm (it can be higher because of the way the standard is worded).
- Let's take a look at the hourly measurements of ozone.

```
> summary(ozone$Sample.Measurement)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
-0.00400	0.02100	0.03100	0.03077	0.04100	0.16000
- From the summary we can see that the maximum hourly concentration is quite high (0.16 ppm) but that in general, the bulk of the distribution is far below 0.075 . (75% are less than 0.041 ppm)

Validate with at least one external data source

- Check detail on the distribution by looking at deciles of the data.

```
> quantile(ozone$Sample.Measurement, seq(0,1,0.1))
```

0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-0.004	0.010	0.018	0.023	0.027	0.031	0.035	0.039	0.043	0.049	0.160

- The data are at least of the right order of magnitude (i.e. the units are correct)
- The range of the distribution is roughly what we'd expect, given the regulation around ambient pollution levels.

Try the easy solution first

- Our original question was:

Which countries in the United States have the highest levels of ambient ozone pollution?

- Detect the countries having the *highest* levels=> we need a list of countries that are ordered from highest to lowest with respect to their levels of ozone.
 - Let's Blindly take the average across the entire year for each county and then rank countries according to this metric.
 - To identify each country we will use a combination of the *State.Name* and the *County.Name* variables.

To do that we need `dyplr` package!

The data frame

- The data frame is a key data structure in statistics and in R.
- There is one observation per row.
- Each column represents a variable or measure or characteristic.

dplyr

- Developed by Hadley Wickham of Rstudio
- An optimized and distilled version of plyr package (also by Hadley)
- Does not provide any "new" functionality per se, but greatly simplifies existing functionality in R
- Provides a "grammar" (in particular, verbs) for data manipulation
- Is very fast, as many key operations are coded in C++

Dplyr Verbs/Grammar

- **select:** return a subset of the columns of a data frame
- **filter:** extract a subset of rows from a data frame based on logical conditions
- **arrange:** reorder rows of a data frame
- **rename:** rename variables in a data frame
- **mutate:** add new variables/columns or transform existing variables
- **summarise / summarize:** generate summary statistics of different variables in the data frame
- **%>%:** the "pipe" operator is used to connect multiple verb actions together into a pipeline

dplyr Properties

- The first argument is a data frame.
- The subsequent arguments describe what to do with it, and you can refer to columns in the data frame directly without using the \$ operator (just use the names).
- The result is a new data frame
- **Data frames must be properly formatted and annotated for this to all be useful**
 - ▶ There should be one observation per row, and each column should represent a feature or characteristic of that observation.

Installing the dplyr package

- **Install dplyr package from CRAN or from GitHub using the devtools package and the `install_github()` function.**

- **To install from CRAN, run**

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

- **To install from GitHub, run**

```
> install_github("hadley/dplyr")
```

- **After installing the package it is important that you load it into your R session with the `library()` function.**

```
> library(dplyr)
```

dplyr package!

- You may get some warnings when the package is loaded because there are functions in the dplyr package that have the same name as functions in other packages. For now you can ignore the warnings.
- **NOTE:** *If you ever run into a problem where R is getting confused over which function you mean to call, you can specify the full name of a function using the :: operator.*
 - ▶ The full name is `package_name::function_name` where package_name is the package name from which the function is defined followed by :: and then the function name.
 - ▶ For example, the filter function from the dplyr package has the full name `dplyr::filter`.
- Calling functions with their full name will resolve any confusion over which function was meant to be called.

dplyr package: select

- ***Let's select State.Name and the County.Name variables from the ozone dataframe***
 - ▶ The select() function can be used to select columns of a data frame that you want to focus on: you can use numerical indices or you can use the names directly.

```
>subset <- select(ozone, State.Name:County.Name)  
>head(subset) # A tibble: 6 x 2
```

	<chr>	<chr>
1	Alabama	Baldwin
2	Alabama	Baldwin
3	Alabama	Baldwin
4	Alabama	Baldwin
5	Alabama	Baldwin
6	Alabama	Baldwin

dplyr package: select

- You can also *omit* variables using the `select()` function by using the negative sign.

```
excludesubset <- select(ozone, -(Method.Name:County.Name))
```

- which indicates that we should include every variable *except* the variables through **Method.Name:County.Name**.
- **Note:** the **:** normally cannot be used with names or strings, but inside the `select()` function you can use it to specify a range of variable names.

dplyr package: match()

- **Match(): Return vector of the position of the columns**

```
i <- match("County.Name", names(ozone))  
i  
j <- match("State.Name", names(ozone))  
j  
head(ozone[, -(i:j)])
```

- **Select() function also allows a special syntax that allows you to specify variable names based on patterns.**

Example, if we want to keep every variable that ends with a "m"

```
subset <- select(ozone, ends_with("m"))  
str(subset)  
subset
```

if we want to keep every variable that starts with a "d"

```
subset <- select(ozone, starts_with("d"))  
str(subset)
```

dplyr package: filter()

- **filter()**: used to extract subsets of rows from a data frame.
- ▶ similar to the existing subset() function in R but is quite a bit faster

```
# select the rows of ozone where Sample.Measurement > 0.075
head(ozone$Sample.Measurement)
ozone.f <- filter(ozone, Sample.Measurement > 0.075)
str(ozone.f)
head(ozone.f)
```

```
# A tibble: 6 × 24
  State.Code County.Code Site.Num Parameter.Code POC Latitude Longitude Datum Parameter.Name
  <chr>      <chr>      <chr>      <chr>      <int>   <dbl>   <dbl> <chr> <chr>
1 01         049       9991      44201         1    34.3   -86.0 NAD83 Ozone
2 01         049       9991      44201         1    34.3   -86.0 NAD83 Ozone
3 01         051        0004      44201         1    32.5   -86.3 NAD83 Ozone
4 01         051        0004      44201         1    32.5   -86.3 NAD83 Ozone
5 01         055        0011      44201         1    33.9   -86.1 NAD83 Ozone
6 01         055        0011      44201         1    33.9   -86.1 NAD83 Ozone
# i 15 more variables: Date.Local <chr>, Time.Local <chr>, Date.GMT <chr>, Time.GMT <chr>,
# Sample.Measurement <dbl>, Units.of.Measure <chr>, MDL <dbl>, Uncertainty <dbl>,
# Qualifier <chr>, Method.Type <chr>, Method.Code <chr>, Method.Name <chr>,
# State.Name <chr>, County.Name <chr>, Date.of.Last.Change <date>
```

dplyr package: filter()

- We can place an arbitrarily complex logical sequence inside of `filter()`,
 - ▶ example extract the rows where `Sample.Measurement > 0.075` and `Latitude` is greater than 30.5

```
ozone.f <- filter(ozone, Sample.Measurement > 0.075 & Latitude > 30.5)
head(ozone.f)
str(ozone.f)
select(ozone.f, starts_with("C"))
```

```
> select(ozone.f, starts_with("C"))
# A tibble: 20,197 × 2
  County.Code County.Name
  <chr>      <chr>
1 049       DeKalb
2 049       DeKalb
3 051       Elmore
4 051       Elmore
5 055       Etowah
6 055       Etowah
7 073       Jefferson
8 073       Jefferson
9 073       Jefferson
10 073      Jefferson
# i 20,187 more rows
```

dplyr package: arrange()

- **arrange()** :used to reorder rows of a data frame according to one of the variables/columns.

Ex: order the rows of the data frame by date, so that the first row is the earliest (oldest) observation and the last row is the latest (most recent) observation.

```
ozoneorder<- arrange(ozone,ozone$Date.Local)  
head(ozone$Date .Local,20)  
tail(ozone$Date.Local,20)
```

dplyr package: rename()

- **Rename(): to rename a variable**

- new name on the left-hand side of the = sign and the old name on the right hand side.
- Check the names of the first five variables in the ozone dataframe.

```
> head(ozone[,1:5],3)
# A tibble: 3 × 5
  State.Code County.Code Site.Num Parameter.Code POC
  <chr>      <chr>      <chr>      <chr>      <int>
1 01         003        0010        44201         1
2 01         003        0010        44201         1
3 01         003        0010        44201         1
```

- Rename Parameter.Code to code and Site.Num to sitenumber

```
> chicago <- rename(ozone, code = Parameter.Code, sitenumber = Site.Num)
> head(chicago[,1:5],3)
# A tibble: 3 × 5
  State.Code County.Code sitenumber code POC
  <chr>      <chr>      <chr>      <chr> <int>
1 01         003        0010        44201     1
2 01         003        0010        44201     1
3 01         003        0010        44201     1
```

dyplr package: mutate()

- **mutate()** : compute transformations of variables in a data frame.
 - create new variables that are derived from existing variables
- **Example:**
 - Let's check whether a given day's air pollution level is higher than or less than average (as opposed to looking at its absolute level).
 - Create a Measurement variable that subtracts the mean from the Sample.Measurement variable.

```
ozonen <- rmutate(ozone , measure= Sample.Measurement -mean(Sample.Measurement, na.rm = TRUE))  
names(ozonen)
```

```
> ozonen <- mutate(ozone, measure = Sample.Measurement - mean(Sample.Measurement, na.rm= TRUE))  
> names(ozonen)  
[1] "State.Code"      "County.Code"      "Site.Num"         "Parameter.Code"  
[5] "POC"             "Latitude"         "Longitude"        "Datum"  
[9] "Parameter.Name"  "Date.Local"       "Time.Local"       "Date.GMT"  
[13] "Time.GMT"        "Sample.Measurement" "Units.of.Measure" "MDL"  
[17] "Uncertainty"     "Qualifier"        "Method.Type"      "Method.Code"  
[21] "Method.Name"     "State.Name"       "County.Name"      "Date.of.Last.Change"  
[25] "measure"
```

dyplr package: group_by(), summarize()

- Group_by() groups the data frames.
- Group_by() function alone will not give any output.
- It should be followed by summarise()/ summarize() function with an appropriate action to perform.

```
ozonen <- mutate(ozone, day=as.Date(Date.Local))
names(ozonen)
tail(select(ozonen, day), 3)
head(select(ozonen, day), 3)
days <- group_by(ozonen, day)
summarize(days, measuremean = mean(Sample.Measurement, na.rm = TRUE), measuremax = max(Sample.Measurement,
na.rm = TRUE), measuremin = min(Sample.Measurement, na.rm = TRUE), uncertaintymedian =median(Sample.Measurement,
na.rm = TRUE))
```

```
# A tibble: 365 x 5
  day          measuremean measuremax measuremin uncertaintymedian
  <date>          <dbl>         <dbl>         <dbl>          <dbl>
1 2019-01-01    0.0235         0.057        -0.001         0.025
2 2019-01-02    0.0206         0.06         -0.001         0.02
3 2019-01-03    0.0195         0.068        -0.001         0.019
4 2019-01-04    0.0191         0.055        -0.001         0.019
5 2019-01-05    0.0219         0.06         -0.002         0.022
6 2019-01-06    0.0271         0.068        -0.001         0.029
7 2019-01-07    0.0269         0.056        -0.002         0.028
8 2019-01-08    0.0227         0.056        -0.001         0.025
9 2019-01-09    0.0251         0.068        -0.001         0.026
10 2019-01-10   0.0233         0.055        -0.001         0.025
# i 355 more rows
```

dyplr package: %>%

- **The pipeline operator %>%** :string together multiple dplyr functions in a sequence of operations.
 - `first(x) %>% second %>% third`
- **Once you travel down the pipeline with %>%, the first argument is taken to be the output of the previous element in the pipeline.**
- **Example:**

Detect the counties having the *highest* levels=> we need a list of counties that are ordered from highest to lowest with respect to their levels of ozone.

 - Let's just blindly take the average across the entire year for each county and then rank counties according to this metric.
 - To identify each county we will use a combination of the *State.Name* and the *County.Name* variables.

```
ranking <- group_by(ozone, State.Name, County.Name) %>%  
  summarize(ozone=mean(Sample.Measurement))%>%  
  as.data.frame %>%  
  arrange(desc(ozone))
```

Try the easy solution first

- **Our original question was:**

Which counties in the United States have the highest levels of ambient ozone pollution?

- **Detect the counties having the *highest* levels=> we need a list of counties that are ordered from highest to lowest with respect to their levels of ozone.**
 - let's just blindly take the average across the entire year for each county and then rank counties according to this metric.
 - To identify each county we will use a combination of the *State.Name* and the *County.Name* variables.

To do that we need dplyr package

```
ranking <- group_by(ozone, State.Name, County.Name) %>%  
  summarize(ozone= mean(Sample.Measurement)) %>%  
  as.data.frame %>%  
  arrange(desc(ozone))
```

Try the easy solution first

Which counties in the United States have the highest levels of ambient ozone pollution?

- Let's look at the top 10 counties in this ranking.

```
> head(ranking, 10)
```

	State.Name	County.Name	ozone
1	Colorado	Clear Creek	0.04815656
2	Wyoming	Albany	0.04811821
3	California	Mariposa	0.04586981
4	Colorado	Gunnison	0.04508115
5	Utah	San Juan	0.04439325
6	Nevada	White Pine	0.04421208
7	North Carolina	Yancey	0.04370322
8	Colorado	Gilpin	0.04356614
9	Arizona	Gila	0.04337246
10	California	Nevada	0.04314739

Try the easy solution first

Which counties in the United States have the highest levels of ambient ozone pollution?

- For comparison let's look at the 10 lowest counties too.

```
> tail(ranking, 10)
```

	State.Name	County.Name	ozone
767	Washington	Skagit	0.020409143
768	New York	Herkimer	0.020367564
769	New Jersey	Warren	0.019992538
770	Oregon	Multnomah	0.019655222
771	Oregon	Washington	0.017063476
772	Alaska	Fairbanks North Star	0.016948310
773	Montana	Missoula	0.012686067
774	Puerto Rico	Bayamon	0.008714503
775	Puerto Rico	Catano	0.006369736
776	Puerto Rico	Mayaguez	0.003310917

Try the easy solution first

Which counties in the United States have the highest levels of ambient ozone pollution?

- Check the number of observations there are in the highest level counties: Clear Creek, Albany County

```
> filter(ozone, State.Name == "Colorado" & County.Name == "Clear Creek") %>% nrow  
[1] 6943
```

Follow up questions

- **Do you have the right data?**

- Sometimes at the conclusion of an exploratory data analysis, the conclusion is that the dataset is not really appropriate for this question.
- In this case the dataset seemed perfectly fine for answering the question of which countries had the highest levels of ozone.

- **Do you need other data?**

- One sub-question we tried to address was whether the country rankings were stable across years. We addressed this by resampling the data once to see if the rankings changed, but the better way to do this would be to simply get the data for previous years and re-do the rankings.

- **Do you have the right question?**

- it's not clear that the question we tried to answer has immediate relevance, and the data didn't really indicate anything to increase the question's relevance.
- For example, it might have been more interesting to assess which countries were in violation of the national ambient air quality standard because determining this could have regulatory implications. However, this is a much more complicated calculation to do, requiring data from at least 3 previous years.

Summary

- The goal of exploratory data analysis is to get you think about your data and reason about your question.
- At this point, we can refine our question or collect new data, all in an iterative process to get at the truth.

Questions