Data Summarization

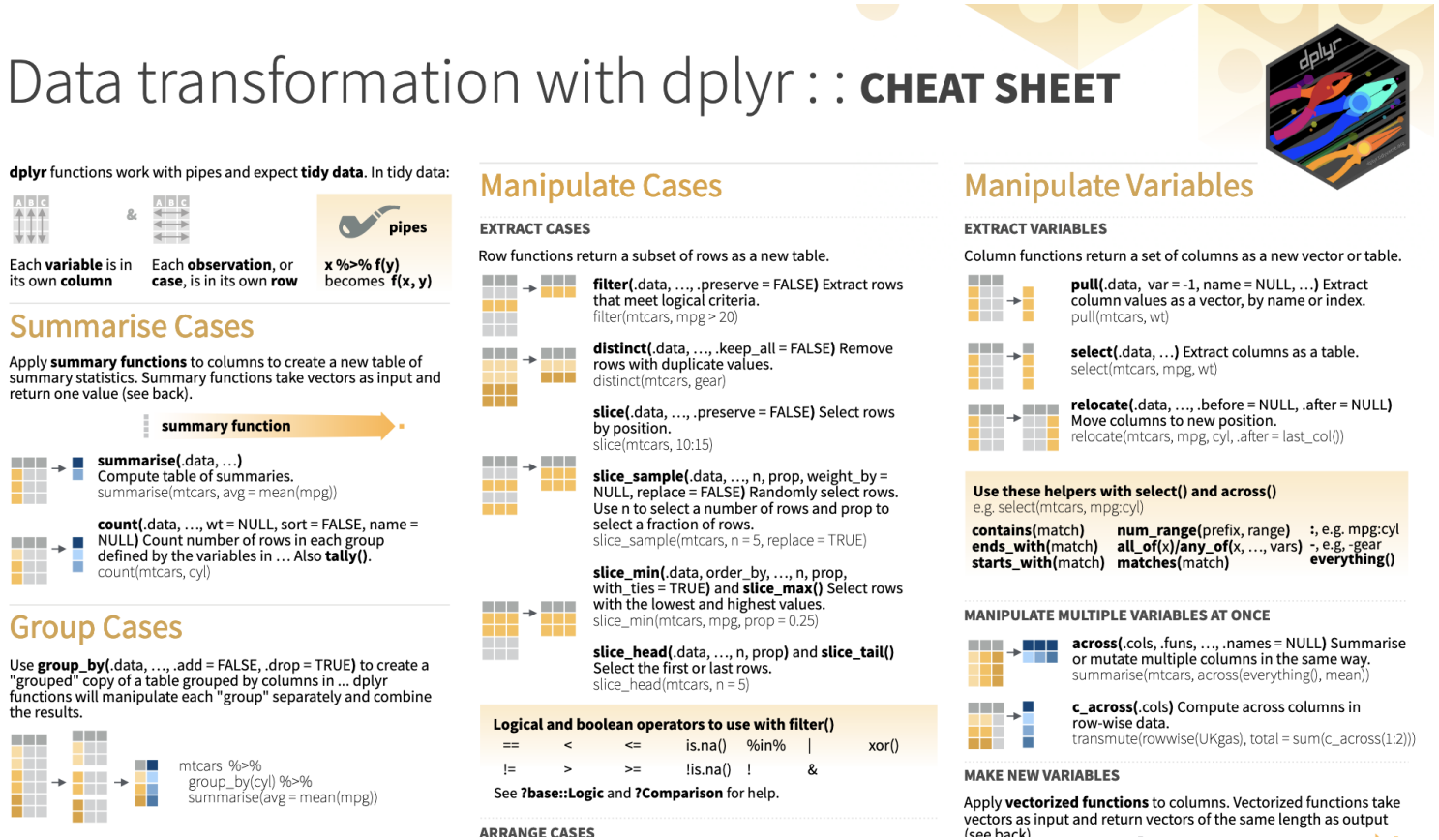
Recap

- `select()`: subset and/or reorder columns
- `filter()`: remove rows
- `arrange()`: reorder rows
- `mutate()`: create new columns or modify them
- `select()` and `filter()` can be combined together
- remove a column: `select()` with `!` mark (`!col_name`)
- you can do sequential steps: especially using pipes `%>%`

 Cheatsheet

Another Cheatsheet

https://raw.githubusercontent.com/rstudio/cheatsheets/main/data-transformation.pdf



Data transformation with dplyr : : CHEAT SHEET

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column** & Each **observation**, or **case**, is in its own **row**

pipes
x %>% f(y) becomes **f(x, y)**

Summarise Cases

Apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, …**)**
Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

count(.data, …, wt = NULL, sort = FALSE, name = NULL**)** Count number of rows in each group defined by the variables in … Also **tally()**.
count(mtcars, cyl)

Group Cases

Use **group_by(**.data, …, .add = FALSE, .drop = TRUE**)** to create a "grouped" copy of a table grouped by columns in ... dplyr functions will manipulate each "group" separately and combine the results.

```
mtcars %>%
  group_by(cyl) %>%
  summarise(avg = mean(mpg))
```

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, …, .preserve = FALSE**)** Extract rows that meet logical criteria.
filter(mtcars, mpg > 20)

distinct(.data, …, .keep_all = FALSE**)** Remove rows with duplicate values.
distinct(mtcars, gear)

slice(.data, …, .preserve = FALSE**)** Select rows by position.
slice(mtcars, 10:15)

slice_sample(.data, …, n, prop, weight_by = NULL, replace = FALSE**)** Randomly select rows. Use n to select a number of rows and prop to select a fraction of rows.
slice_sample(mtcars, n = 5, replace = TRUE)

slice_min(.data, order_by, …, n, prop, with_ties = TRUE**)** and **slice_max()** Select rows with the lowest and highest values.
slice_min(mtcars, mpg, prop = 0.25)

slice_head(.data, …, n, prop**)** and **slice_tail()** Select the first or last rows.
slice_head(mtcars, n = 5)

Logical and boolean operators to use with filter()

==	<	<=	is.na()	%in%	|	xor()
!=	>	>=	!is.na()	!	&	

See **?base::Logic** and **?Comparison** for help.

ARRANGE CASES

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1, name = NULL, …**)** Extract column values as a vector, by name or index.
pull(mtcars, wt)

select(.data, …**)** Extract columns as a table.
select(mtcars, mpg, wt)

relocate(.data, …, .before = NULL, .after = NULL**)** Move columns to new position.
relocate(mtcars, mpg, cyl, .after = last_col())

Use these helpers with select() and across()
e.g. select(mtcars, mpg:cyl)

contains(match**)**	**num_range(**prefix, range**)**	:, e.g. mpg:cyl
ends_with(match**)**	**all_of(**x**)/any_of(**x, …, vars**)**	-, e.g, -gear
starts_with(match**)**	**matches(**match**)**	**everything()**

MANIPULATE MULTIPLE VARIABLES AT ONCE

across(.cols, .funs, …, .names = NULL**)** Summarise or mutate multiple columns in the same way.
summarise(mtcars, across(everything(), mean))

c_across(.cols**)** Compute across columns in row-wise data.
transmute(rowwise(UKgas), total = sum(c_across(1:2)))

MAKE NEW VARIABLES

Apply **vectorized functions** to columns. Vectorized functions take vectors as input and return vectors of the same length as output (see back).

Data Summarization

- Basic statistical summarization
  - `mean(x)`: takes the mean of x
  - `sd(x)`: takes the standard deviation of x
  - `median(x)`: takes the median of x
  - `quantile(x)`: displays sample quantiles of x. Default is min, IQR, max
  - `range(x)`: displays the range. Same as `c(min(x), max(x))`
  - `sum(x)`: sum of x
  - `max(x)`: maximum value in x
  - `min(x)`: minimum value in x
- **all have the `na.rm =` argument for missing data**

Statistical summarization

The vector getting summarized goes inside the parentheses:

```
x <- c(1, 5, 7, 4, 2, 8)
mean(x)
```

```
[1] 4.5
```

```
range(x)
```

```
[1] 1 8
```

```
sum(x)
```

```
[1] 27
```

Statistical summarization

Note that many of these functions have additional inputs regarding missing data, typically requiring the `na.rm` argument ("remove NAs").

```
x <- c(1, 5, 7, 4, 2, 8, NA)
mean(x)
```

```
[1] NA
```

```
mean(x, na.rm = TRUE)
```

```
[1] 4.5
```

```
quantile(x)
```

```
Error in quantile.default(x): missing values and NaN's not allowed if 'na.rm' is FALSE
```

```
quantile(x, na.rm = TRUE)
```

```
  0%  25%  50%  75% 100% 
 1.0  2.5  4.5  6.5  8.0 
```

Statistical summarization

We will talk more about data types later, but you can only do summarization on numeric or logical types, NOT characters.

```
x <- c(1, 5, 7, 4, 2, 8)
sum(x)
```

```
[1] 27
```

```
y <- c(TRUE, FALSE, FALSE, TRUE) # FALSE == 0 and TRUE == 1
sum(y)
```

```
[1] 2
```

```
z <- c("TRUE", "FALSE", "FALSE", "TRUE")
sum(z)
```

```
Error in sum(z): invalid 'type' (character) of argument
```

Some examples

We can use the `CO_heat_ER` object from the `dasehr` package to explore different ways of summarizing data. (This dataset contains information about the number and rate of visits for heat-related illness to ERs in Colorado from 2011-2022, adjusted for age.) The `head` command displays the first rows of an object:

```
library(dasehr)
head(CO_heat_ER)
```

```
# A tibble: 6 × 7
  county     rate lower95cl upper95cl visits  year gender
  <chr>     <dbl>     <dbl>     <dbl>  <dbl> <dbl> <chr>
1 Statewide  5.64      4.70      6.59    140  2011 Female
2 Statewide  7.39      6.30      8.47    183  2011 Male
3 Statewide  6.51      5.80      7.23    323  2011 Both genders
4 Statewide  5.64      4.72      6.57    146  2012 Female
5 Statewide  7.56      6.48      8.65    193  2012 Male
6 Statewide  6.58      5.88      7.29    339  2012 Both genders
```

The dplyr pipe %>% operator

A nice and readable way to chain together multiple R functions.

Changes `f(x, y)` to `x %>% f(y)`.

```
# Going to work
get_dressed(me,
            pack_lunch(
              check_pockets(
                wallet = TRUE, phone = TRUE, keys = TRUE),
              items = c("sandwich", "chips", "apple"), lunchbox = TRUE),
            pants = TRUE, shirt = TRUE, footwear = "sandals")

# Going to work, the tidy way
me %>%
  get_dressed(pants = TRUE, shirt = TRUE, footwear = "sandals") %>%
  pack_lunch(items = c("sandwich", "chips", "apple"), lunchbox = TRUE) %>%
  check_pockets(wallet = TRUE, phone = TRUE, keys = TRUE)
```

Statistical summarization the "tidy" way

```
CO_heat_ER %>% pull(visits) %>% mean(na.rm=T) # alt: pull(CO_heat_ER, visits) %>% mean(na.rm=T)
```

```
[1] 9.791114
```

```
CO_heat_ER %>% pull(rate) %>% median(na.rm=T)
```

```
[1] 0
```

```
CO_heat_ER %>% pull(visits) %>% quantile(na.rm=T)
```

```
  0%  25%  50%  75% 100% 
   0    0    0    0  494 
```

```
CO_heat_ER %>% pull(rate) %>% quantile(probs = 0.9, na.rm=T)
```

```
     90% 
6.704074 
```

Behavior of `pull()` function

`pull()` converts a single data column into a vector. This allows you to run summary functions on these data. Once you have "pulled" the data column out, you don't have to name it again in any piped summary functions.

```
er_visits <- CO_heat_ER %>% pull(visits)
class(er_visits)
```

```
[1] "numeric"
```

```
er_visits
```

```
  [1] 140 183 323 146 193 339 124 178 302  92 145 237 140 215 355 172 295 467
 [19] 113 210 323 163 293 456 154 235 389 105 197 302 173 252 425 185 309 494
 [37]  17  12  29  NA  13  23  14  17  31  NA  NA  15  NA  NA  16  14  28  42
 [55]  11  21  32  16  21  37  17  19  36  NA  15  24  19  16  35  18  27  45
 [73]   0   0   0   0   0   0  NA   0  NA   0   0   0   0  NA  NA   0  NA  NA
 [91]   0  NA  NA   0   0   0  NA   0  NA   0  NA  NA   0   0   0   0   0   0
[109]  16  17  33  12  15  27  11  NA  20  NA  NA  NA  16  15  31  18  21  39
[127]  NA  NA  16  13  21  34  17  14  31  NA  11  16  12  18  30  15  24  39
[145]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
[163]  NA  NA  NA   0  NA  NA  NA   0  NA   0   0   0  NA  NA  NA  NA  NA  NA
[181]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
[199]   0   0   0  NA   0  NA  NA   0  NA   0   0   0   0   0   0   0  NA  NA
[217]   0   0   0   0  NA  NA   0   0   0   0  NA  NA   0  NA  NA   0  NA  NA
[235]   0  NA  NA  NA  NA  NA  NA  NA  NA   0  NA  NA   0  NA  NA   0   0   0
[253]  NA  NA  12  NA  NA  13  NA  NA  12  NA  NA  19  NA  NA  14  NA  NA  18
[271]  NA  NA  12  NA  11  18  NA  NA  13  NA  NA  12  NA  11  19  NA  15  19
[289]  NA  NA  NA  NA  NA  NA  NA   0  NA  NA   0  NA   0  NA  NA  NA   0  NA
```

Summarization on tibbles (data frames)

Historical CO2 emissions by country

Let's look at a dataset that tracks yearly estimated CO2 emissions by country. We will read it in as a `tibble`.

If you have the `dasehr` package installed successfully:

```
library(dasehr)
yearly_co2 <- yearly_co2_emissions
```

If not, download the `csv` file from https://daseh.org/data/Yearly_CO2_Emissions_1000_tonnes.csv and read it in:

```
yearly_co2 <-
  read_csv(file = "https://daseh.org/data/Yearly_CO2_Emissions_1000_tonnes.csv")
```

Check out the data:

```
head(yearly_co2)
```

```
# A tibble: 6 × 265
  country  `1751` `1752` `1753` `1754` `1755` `1756` `1757` `1758` `1759` `1760`
  <chr>     <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
1 Afghani…     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
2 Albania      NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
3 Algeria      NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
4 Andorra      NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
5 Angola       NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
```

Historical CO2 emissions by country

Check out the data:

```
str(yearly_co2)
```

```
spc_tbl_ [192 × 265] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ country: chr [1:192] "Afghanistan" "Albania" "Algeria" "Andorra" ...
 $ 1751   : num [1:192] NA NA NA NA NA NA NA NA NA NA ...
 $ 1752   : num [1:192] NA NA NA NA NA NA NA NA NA NA ...
 $ 1753   : num [1:192] NA NA NA NA NA NA NA NA NA NA ...
 $ 1754   : num [1:192] NA NA NA NA NA NA NA NA NA NA ...
 $ 1755   : num [1:192] NA NA NA NA NA NA NA NA NA NA ...
 $ 1756   : num [1:192] NA NA NA NA NA NA NA NA NA NA ...
 $ 1757   : num [1:192] NA NA NA NA NA NA NA NA NA NA ...
 $ 1758   : num [1:192] NA NA NA NA NA NA NA NA NA NA ...
 $ 1759   : num [1:192] NA NA NA NA NA NA NA NA NA NA ...
 $ 1760   : num [1:192] NA NA NA NA NA NA NA NA NA NA ...
 $ 1761   : num [1:192] NA NA NA NA NA NA NA NA NA NA ...
 $ 1762   : num [1:192] NA NA NA NA NA NA NA NA NA NA ...
 $ 1763   : num [1:192] NA NA NA NA NA NA NA NA NA NA ...
 $ 1764   : num [1:192] NA NA NA NA NA NA NA NA NA NA ...
 $ 1765   : num [1:192] NA NA NA NA NA NA NA NA NA NA ...
 $ 1766   : num [1:192] NA NA NA NA NA NA NA NA NA NA ...
 $ 1767   : num [1:192] NA NA NA NA NA NA NA NA NA NA ...
 $ 1768   : num [1:192] NA NA NA NA NA NA NA NA NA NA ...
 $ 1769   : num [1:192] NA NA NA NA NA NA NA NA NA NA ...
 $ 1770   : num [1:192] NA NA NA NA NA NA NA NA NA NA ...
 $ 1771   : num [1:192] NA NA NA NA NA NA NA NA NA NA ...
 $ 1772   : num [1:192] NA NA NA NA NA NA NA NA NA NA ...
 $ 1773   : num [1:192] NA NA NA NA NA NA NA NA NA NA ...
```

CO2 Emissions

Before we go further, let's rename the second column using the `rename()` function in `dplyr`.

In this case, we will use the backticks (`) because we will be referring to a column that has a numerical name. If there are funky spaces or characters in the column name, the backticks are required.

```
library(dplyr)
yearly_co2 <- yearly_co2 %>%
  rename(year1751 = `1751`)
```

CO2 Emissions

`colnames()` will show us the column names and show that the `1751` column is renamed:

```
colnames(yearly_co2)
```

```
  [1] "country"  "year1751" "1752"     "1753"     "1754"     "1755"
  [7] "1756"     "1757"     "1758"     "1759"     "1760"     "1761"
 [13] "1762"     "1763"     "1764"     "1765"     "1766"     "1767"
 [19] "1768"     "1769"     "1770"     "1771"     "1772"     "1773"
 [25] "1774"     "1775"     "1776"     "1777"     "1778"     "1779"
 [31] "1780"     "1781"     "1782"     "1783"     "1784"     "1785"
 [37] "1786"     "1787"     "1788"     "1789"     "1790"     "1791"
 [43] "1792"     "1793"     "1794"     "1795"     "1796"     "1797"
 [49] "1798"     "1799"     "1800"     "1801"     "1802"     "1803"
 [55] "1804"     "1805"     "1806"     "1807"     "1808"     "1809"
 [61] "1810"     "1811"     "1812"     "1813"     "1814"     "1815"
 [67] "1816"     "1817"     "1818"     "1819"     "1820"     "1821"
 [73] "1822"     "1823"     "1824"     "1825"     "1826"     "1827"
 [79] "1828"     "1829"     "1830"     "1831"     "1832"     "1833"
 [85] "1834"     "1835"     "1836"     "1837"     "1838"     "1839"
 [91] "1840"     "1841"     "1842"     "1843"     "1844"     "1845"
 [97] "1846"     "1847"     "1848"     "1849"     "1850"     "1851"
[103] "1852"     "1853"     "1854"     "1855"     "1856"     "1857"
[109] "1858"     "1859"     "1860"     "1861"     "1862"     "1863"
[115] "1864"     "1865"     "1866"     "1867"     "1868"     "1869"
[121] "1870"     "1871"     "1872"     "1873"     "1874"     "1875"
[127] "1876"     "1877"     "1878"     "1879"     "1880"     "1881"
[133] "1882"     "1883"     "1884"     "1885"     "1886"     "1887"
[139] "1888"     "1889"     "1890"     "1891"     "1892"     "1893"
```

Summarize the data: `dplyr summarize()` function

`summarize` creates a summary table of a column you're interested in.

Can run multiple summary statistics at once (unlike `pull()` which can only do a single calculation on one column).

You can also do more elaborate summaries across different groups of data using `group_by()`. More on this later!

```
# General format - Not the code!
{data to use} %>%
   summarize({summary column name} = {operator(source column)},
             {summary column name} = {operator(source column)})
```

Summarize the data: `dplyr summarize()` function

`summarize` creates a summary table of a column you're interested in.

```
# General format - Not the code!
{data to use} %>%
   summarize({summary column name} = {operator(source column)})

yearly_co2 %>%
  summarize(mean_1989 = mean(`1989`)) # Note the backticks, this is a column name!

# A tibble: 1 × 1
  mean_1989
      <dbl>
1        NA

yearly_co2 %>%
  summarize(mean_1989 = mean(`1989`, na.rm = TRUE))

# A tibble: 1 × 1
  mean_1989
      <dbl>
1   126046.
```

Summarize the data: `dplyr summarize()` function

`summarize()` can do multiple operations at once. Just separate by a comma.

```
yearly_co2 %>%
  summarize(mean_1989 = mean(`1989`, na.rm = TRUE),
            median_1989 = median(`1989`, na.rm = TRUE),
            median(`2000`, na.rm = TRUE))
```

```
# A tibble: 1 × 3
  mean_1989 median_1989 `median(\`2000\`, na.rm = TRUE)`
      <dbl>       <dbl>                            <dbl>
1   126046.        8690                             7510
```

Notice how when we forget to provide a new name, output is still provided, but the column name is messy.

Summarize the data: `dplyr summarize()` function

This looks better.

```
yearly_co2 %>%
  summarize(mean_1989 = mean(`1989`, na.rm = TRUE),
            median_1989 = median(`1989`, na.rm = TRUE),
            median_2000 = median(`2000`, na.rm = TRUE))
```

```
# A tibble: 1 × 3
  mean_1989 median_1989 median_2000
      <dbl>       <dbl>       <dbl>
1   126046.        8690        7510
```

Summarize the data: `dplyr summarize()` function

Note that `summarize()` creates a separate tibble from the original data, so you don't want to overwrite your original data if you decide to save the summary.

If you want to save a summary statistic in the original data, use `mutate()` instead to create a new column for the summary statistic.

`summary()` Function

Using `summary()` can give you rough snapshots of each numeric column (character columns are skipped):

```
summary(yearly_co2)
```

```
  country             year1751          1752          1753          1754     
Length:192         Min.   :9360   Min.   :9360   Min.   :9360   Min.   :9370  
Class :character   1st Qu.:9360   1st Qu.:9360   1st Qu.:9360   1st Qu.:9370  
Mode  :character   Median :9360   Median :9360   Median :9360   Median :9370  
                   Mean   :9360   Mean   :9360   Mean   :9360   Mean   :9370  
                   3rd Qu.:9360   3rd Qu.:9360   3rd Qu.:9360   3rd Qu.:9370  
                   Max.   :9360   Max.   :9360   Max.   :9360   Max.   :9370  
                   NA's   :191    NA's   :191    NA's   :191    NA's   :191   
     1755           1756           1757           1758           1759     
Min.   :9370   Min.   :10000   Min.   :10000   Min.   :10000   Min.   :10000  
1st Qu.:9370   1st Qu.:10000   1st Qu.:10000   1st Qu.:10000   1st Qu.:10000  
Median :9370   Median :10000   Median :10000   Median :10000   Median :10000  
Mean   :9370   Mean   :10000   Mean   :10000   Mean   :10000   Mean   :10000  
3rd Qu.:9370   3rd Qu.:10000   3rd Qu.:10000   3rd Qu.:10000   3rd Qu.:10000  
Max.   :9370   Max.   :10000   Max.   :10000   Max.   :10000   Max.   :10000  
NA's   :191    NA's   :191     NA's   :191     NA's   :191     NA's   :191    
     1760           1761           1762           1763     
Min.   :10000   Min.   :11000   Min.   :11000   Min.   :11000  
1st Qu.:10000   1st Qu.:11000   1st Qu.:11000   1st Qu.:11000  
Median :10000   Median :11000   Median :11000   Median :11000  
Mean   :10000   Mean   :11000   Mean   :11000   Mean   :11000  
3rd Qu.:10000   3rd Qu.:11000   3rd Qu.:11000   3rd Qu.:11000  
Max.   :10000   Max.   :11000   Max.   :11000   Max.   :11000  
NA's   :191     NA's   :191     NA's   :191     NA's   :191    
```

Summary & Lab Part 1

- summary stats (`mean()`) work with `pull()`
- don't forget the `na.rm = TRUE` argument!
- `summary(x)`: quantile information
- `summarize`: creates a summary table of columns of interest

- Class Website
- Lab

CO ER Heat Illness Visits

Let's go back to the dataset of CO ER visits for heat-related illness. Remember, we loaded this data into our session and saved it as the object `CO_heat_ER`.

```
head(CO_heat_ER)
```

```
# A tibble: 6 × 7
  county     rate lower95cl upper95cl visits  year gender
  <chr>     <dbl>     <dbl>     <dbl>  <dbl> <dbl> <chr>
1 Statewide  5.64      4.70      6.59    140  2011 Female
2 Statewide  7.39      6.30      8.47    183  2011 Male
3 Statewide  6.51      5.80      7.23    323  2011 Both genders
4 Statewide  5.64      4.72      6.57    146  2012 Female
5 Statewide  7.56      6.48      8.65    193  2012 Male
6 Statewide  6.58      5.88      7.29    339  2012 Both genders
```

`distinct()` values

`distinct(x)` will return the unique elements of column `x`.

```
CO_heat_ER %>%
  distinct(county)
```

```
# A tibble: 65 × 1
   county
   <chr>
 1 Statewide
 2 Adams
 3 Alamosa
 4 Arapahoe
 5 Archuleta
 6 Baca
 7 Bent
 8 Boulder
 9 Broomfield
10 Chaffee
# ℹ 55 more rows
```

How many `distinct()` values?

`n_distinct()` tells you the number of unique elements. *Must pull the column first!*

```
CO_heat_ER %>%
  pull(county) %>%
  n_distinct()
```

```
[1] 65
```

dplyr: count

Use `count` to return a frequency table of unique elements of a data.frame.

```
CO_heat_ER %>% count(county)
```

```
# A tibble: 65 × 2
   county         n
   <chr>      <int>
 1 Adams         36
 2 Alamosa       36
 3 Arapahoe      36
 4 Archuleta     36
 5 Baca          36
 6 Bent          36
 7 Boulder       36
 8 Broomfield    36
 9 Chaffee       36
10 Cheyenne      36
# ℹ 55 more rows
```

dplyr: count

Multiple columns listed further subdivides the count.

```
CO_heat_ER %>% count(county, gender)
```

```
# A tibble: 195 × 3
   county    gender           n
   <chr>     <chr>        <int>
 1 Adams     Both genders    12
 2 Adams     Female          12
 3 Adams     Male            12
 4 Alamosa   Both genders    12
 5 Alamosa   Female          12
 6 Alamosa   Male            12
 7 Arapahoe  Both genders    12
 8 Arapahoe  Female          12
 9 Arapahoe  Male            12
10 Archuleta Both genders    12
# ℹ 185 more rows
```

Note: `count()` includes NAs

Grouping

Perform Operations By Groups: dplyr

`group_by` allows you group the data set by variables/columns you specify:

```
# Regular data
CO_heat_ER
```

```
# A tibble: 2,340 × 7
   county     rate lower95cl upper95cl visits  year gender
   <chr>     <dbl>     <dbl>     <dbl>  <dbl> <dbl> <chr>
 1 Statewide  5.64      4.70      6.59    140  2011 Female
 2 Statewide  7.39      6.30      8.47    183  2011 Male
 3 Statewide  6.51      5.80      7.23    323  2011 Both genders
 4 Statewide  5.64      4.72      6.57    146  2012 Female
 5 Statewide  7.56      6.48      8.65    193  2012 Male
 6 Statewide  6.58      5.88      7.29    339  2012 Both genders
 7 Statewide  4.94      4.06      5.82    124  2013 Female
 8 Statewide  6.72      5.72      7.72    178  2013 Male
 9 Statewide  5.82      5.16      6.49    302  2013 Both genders
10 Statewide  3.52      2.80      4.25     92  2014 Female
# ℹ 2,330 more rows
```

Perform Operations By Groups: dplyr

`group_by` allows you group the data set by variables/columns you specify:

```
CO_heat_ER_grouped <- CO_heat_ER %>% group_by(gender)
CO_heat_ER_grouped
```

```
# A tibble: 2,340 × 7
# Groups:   gender [3]
   county     rate lower95cl upper95cl visits  year gender
   <chr>     <dbl>     <dbl>     <dbl>  <dbl> <dbl> <chr>
 1 Statewide  5.64      4.70      6.59    140  2011 Female
 2 Statewide  7.39      6.30      8.47    183  2011 Male
 3 Statewide  6.51      5.80      7.23    323  2011 Both genders
 4 Statewide  5.64      4.72      6.57    146  2012 Female
 5 Statewide  7.56      6.48      8.65    193  2012 Male
 6 Statewide  6.58      5.88      7.29    339  2012 Both genders
 7 Statewide  4.94      4.06      5.82    124  2013 Female
 8 Statewide  6.72      5.72      7.72    178  2013 Male
 9 Statewide  5.82      5.16      6.49    302  2013 Both genders
10 Statewide  3.52      2.80      4.25     92  2014 Female
# … 2,330 more rows
```

Summarize the grouped data

It's grouped! Grouping doesn't change the data in any way, but how **functions operate on it**. Now we can summarize `Data_Value` (percent of respondents) by group:

```
CO_heat_ER_grouped %>% summarize(avg_visits = mean(visits, na.rm = TRUE))
```

```
# A tibble: 3 × 2
  gender       avg_visits
  <chr>             <dbl>
1 Both genders      16.3
2 Female             4.77
3 Male               9.00
```

Use the `pipe` to string these together!

Pipe `yts` into `group_by`, then pipe that into `summarize`:

```
CO_heat_ER %>%
  group_by(gender) %>%
  summarize(avg_visits = mean(visits, na.rm = TRUE),
            max_visits = max(visits, na.rm = TRUE))
```

```
# A tibble: 3 × 3
  gender       avg_visits max_visits
  <chr>             <dbl>      <dbl>
1 Both genders      16.3         494
2 Female             4.77        185
3 Male               9.00        309
```

Group by as many variables as you want

`group_by` Response and Education:

```
CO_heat_ER %>%
  group_by(gender, year) %>%
  summarize(avg_visits = mean(visits, na.rm = TRUE),
            max_visits = max(visits, na.rm = TRUE))
```

```
# A tibble: 36 × 4
# Groups:   gender [3]
   gender        year avg_visits max_visits
   <chr>        <dbl>      <dbl>      <dbl>
 1 Both genders  2011      11.3         323
 2 Both genders  2012      12.8         339
 3 Both genders  2013      12.4         302
 4 Both genders  2014       9.67        237
 5 Both genders  2015      14.9         355
 6 Both genders  2016      22.4         467
 7 Both genders  2017      16.3         323
 8 Both genders  2018      25.6         456
 9 Both genders  2019      20.3         389
10 Both genders  2020      14.5         302
# ℹ 26 more rows
```

Only the last `group_by` is recognized…

You can overwrite the first `group_by` with a new one.

```
CO_heat_ER %>%
  group_by(gender, year)  %>%
  group_by(year)
```

```
# A tibble: 2,340 × 7
# Groups:   year [12]
   county     rate lower95cl upper95cl visits  year gender
   <chr>     <dbl>     <dbl>     <dbl>  <dbl> <dbl> <chr>
 1 Statewide  5.64      4.70      6.59    140  2011 Female
 2 Statewide  7.39      6.30      8.47    183  2011 Male
 3 Statewide  6.51      5.80      7.23    323  2011 Both genders
 4 Statewide  5.64      4.72      6.57    146  2012 Female
 5 Statewide  7.56      6.48      8.65    193  2012 Male
 6 Statewide  6.58      5.88      7.29    339  2012 Both genders
 7 Statewide  4.94      4.06      5.82    124  2013 Female
 8 Statewide  6.72      5.72      7.72    178  2013 Male
 9 Statewide  5.82      5.16      6.49    302  2013 Both genders
10 Statewide  3.52      2.80      4.25     92  2014 Female
# ℹ 2,330 more rows
```

Ungroup the data

The `ungroup` function will allow you to clear the groups from the data.

```
CO_heat_ER <- ungroup(CO_heat_ER)
CO_heat_ER
```

```
# A tibble: 2,340 × 7
   county     rate lower95cl upper95cl visits  year gender
   <chr>     <dbl>     <dbl>     <dbl>  <dbl> <dbl> <chr>
 1 Statewide  5.64      4.70      6.59    140  2011 Female
 2 Statewide  7.39      6.30      8.47    183  2011 Male
 3 Statewide  6.51      5.80      7.23    323  2011 Both genders
 4 Statewide  5.64      4.72      6.57    146  2012 Female
 5 Statewide  7.56      6.48      8.65    193  2012 Male
 6 Statewide  6.58      5.88      7.29    339  2012 Both genders
 7 Statewide  4.94      4.06      5.82    124  2013 Female
 8 Statewide  6.72      5.72      7.72    178  2013 Male
 9 Statewide  5.82      5.16      6.49    302  2013 Both genders
10 Statewide  3.52      2.80      4.25     92  2014 Female
# ℹ 2,330 more rows
```

group_by with mutate - just add data

We can also use `mutate` to calculate the mean value for each year and add it as a column:

```
CO_heat_ER %>%
  group_by(year, gender) %>%
  mutate(visits_year_avg = mean(visits, na.rm = TRUE)) %>%
  select(county, visits, visits_year_avg)
```

```
# A tibble: 2,340 × 5
# Groups:   year, gender [36]
    year gender       county    visits visits_year_avg
   <dbl> <chr>        <chr>      <dbl>           <dbl>
 1  2011 Female       Statewide    140            4.32
 2  2011 Male         Statewide    183            6.06
 3  2011 Both genders Statewide    323           11.3
 4  2012 Female       Statewide    146            4.76
 5  2012 Male         Statewide    193            6.71
 6  2012 Both genders Statewide    339           12.8
 7  2013 Female       Statewide    124            3.72
 8  2013 Male         Statewide    178            6.11
 9  2013 Both genders Statewide    302           12.4
10  2014 Female       Statewide     92            2.5
# ℹ 2,330 more rows
```

Counting

There are other functions, such as `n()` count the number of observations (NAs included).

```
CO_heat_ER %>%
  group_by(year) %>%
  summarize(n = n(),
            mean = mean(visits, na.rm = TRUE))
```

```
# A tibble: 12 × 3
    year     n  mean
   <dbl> <int> <dbl>
 1  2011   195  7.17
 2  2012   195  8.14
 3  2013   195  7.33
 4  2014   195  5.51
 5  2015   195  8.68
 6  2016   195 13.2
 7  2017   195  9.39
 8  2018   195 14.7
 9  2019   195 12.3
10  2020   195  8.45
11  2021   195 11.6
12  2022   195 13.3
```

Counting

`count()` and `n()` can give very similar information.

```
CO_heat_ER %>% count(county)
```

```
# A tibble: 65 × 2
   county         n
   <chr>      <int>
 1 Adams         36
 2 Alamosa       36
 3 Arapahoe      36
 4 Archuleta     36
 5 Baca          36
 6 Bent          36
 7 Boulder       36
 8 Broomfield    36
 9 Chaffee       36
10 Cheyenne      36
# ℹ 55 more rows
```

```
CO_heat_ER %>% group_by(county) %>% summarize(n()) # n() typically used with summarize
```

```
# A tibble: 65 × 2
   county     `n()`
   <chr>      <int>
 1 Adams         36
 2 Alamosa       36
 3 Arapahoe      36
 4 Archuleta     36
 5 Baca          36
```

A few miscellaneous topics ..

Base R functions you might see: `length` and `unique`

These functions require a column as a vector using `pull()`.

```
CO_heat_ER_loc <- CO_heat_ER %>% pull(county) # pull() to make a vector
CO_heat_ER_loc %>% unique() # similar to distinct()
```

```
 [1] "Statewide"   "Adams"       "Alamosa"     "Arapahoe"    "Archuleta"
 [6] "Baca"        "Bent"        "Boulder"     "Broomfield"  "Chaffee"
[11] "Cheyenne"    "Clear Creek" "Conejos"     "Costilla"    "Crowley"
[16] "Custer"      "Delta"       "Denver"      "Dolores"     "Douglas"
[21] "Eagle"       "Elbert"      "El Paso"     "Fremont"     "Garfield"
[26] "Gilpin"      "Grand"       "Gunnison"    "Hinsdale"    "Huerfano"
[31] "Jackson"     "Jefferson"   "Kiowa"       "Kit Carson"  "Lake"
[36] "La Plata"    "Larimer"     "Las Animas"  "Lincoln"     "Logan"
[41] "Mesa"        "Mineral"     "Moffat"      "Montezuma"   "Montrose"
[46] "Morgan"      "Otero"       "Ouray"       "Park"        "Phillips"
[51] "Pitkin"      "Prowers"     "Pueblo"      "Rio Blanco"  "Rio Grande"
[56] "Routt"       "Saguache"    "San Juan"    "San Miguel"  "Sedgwick"
[61] "Summit"      "Teller"      "Washington"  "Weld"        "Yuma"
```

Base R functions you might see: `length` and `unique`

These functions require a column as a vector using `pull()`.

```
CO_heat_ER_loc %>% unique() %>% length() # similar to n_distinct()
```

```
[1] 65
```

* New! * Many dplyr functions now have a `.by=` argument

Pipe `CO_heat_ER` into `group_by`, then pipe that into `summarize`:

```
CO_heat_ER %>%
  group_by(county) %>%
  summarize(avg_visits = mean(visits, na.rm = TRUE),
            max_visits = max(visits, na.rm = TRUE))
```

is the same as..

```
CO_heat_ER %>%
  summarize(avg_visits = mean(visits, na.rm = TRUE),
            max_visits = max(visits, na.rm = TRUE),
            .by = county)
```

`summary()` vs. `summarize()`

- `summary()` (base R) gives statistics table on a dataset.
- `summarize()` (dplyr) creates a more customized summary tibble/dataframe.

Summary & Lab Part 2

- `count(x)`: what unique values do you have?
  - `distinct()`: what are the distinct values?
  - `n_distinct()` with `pull()`: how many distinct values?
- `group_by()`: changes all subsequent functions
  - combine with `summarize()` to get statistics per group
  - combine with `mutate()` to add column
- `summarize()` with `n()` gives the count (NAs included)

- Class Website
- Lab

Extra Slides: More advanced summarization

Data Summarization on data frames

- Statistical summarization across the data frame
  - `rowMeans(x)`: takes the means of each row of x
  - `colMeans(x)`: takes the means of each column of x
  - `rowSums(x)`: takes the sum of each row of x
  - `colSums(x)`: takes the sum of each column of x

`rowMeans()` example

Get means for each row.

Let's see what the mean CO2 emissions is across years for each row (country):

```
yearly_co2 %>%
  select(starts_with("year")) %>%
  rowMeans(na.rm = TRUE) %>%
  head(n = 5)
```

```
[1] NaN NaN NaN NaN NaN
```

```
yearly_co2 %>%
  group_by(country) %>%
  summarize(mean = rowMeans(across(starts_with("year")), na.rm = TRUE)) %>%
  head(n = 5)
```

```
# A tibble: 5 × 2
  country      mean
  <chr>       <dbl>
1 Afghanistan   NaN
2 Albania       NaN
3 Algeria       NaN
4 Andorra       NaN
5 Angola        NaN
```

`colMeans()` example

Get means for each column.

Let's see what the mean is across each column (year):

```
yearly_co2 %>%
  select(starts_with("year")) %>%
  colMeans(na.rm = TRUE) %>%
  head(n = 5)

year1751
    9360

yearly_co2 %>%
  summarize(across(starts_with("year"), ~mean(.x, na.rm = TRUE)))

# A tibble: 1 × 1
  year1751
     <dbl>
1     9360
```