

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 `|>`

[Day 3 Cheatsheet](#)

The Data

We can use the CO heat-related ER visits dataset to explore different ways of summarizing data.

Reminder: 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 Data

We can use the CO heat-related ER visits dataset to explore different ways of summarizing data.

The `head` function displays the first rows of an object:

```
er <-  
  read_csv("https://daseh.org/data/CO_ER_heat_visits.csv")
```

```
head(er)
```

```
# A tibble: 6 × 6  
  county   rate lower95cl upper95cl visits   year  
  <chr>   <dbl>    <dbl>     <dbl>   <dbl>   <dbl>  
1 Adams    6.73      NA       9.24     29   2011  
2 Adams    4.84      2.85     NA        23   2012  
3 Adams    6.84      4.36     9.31     31   2013  
4 Adams    3.08      1.71     4.85     15   2014  
5 Adams    3.36      1.89     5.23     16   2015  
6 Adams    8.85      6.12    11.6      42   2016
```

Behavior of **pull()** function

pull() converts a single data column into a **vector**.

```
er |> pull(visits)
```

Data Summarization

Now that we have a [vector of numbers..](#) what can we do with it?

- 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

Pipe (

>) vectors into summarizing functions

A vector can be summarized:

```
er |> pull(visits) |> mean()
```

```
[1] NA
```

Add the `na.rm =` argument for missing data

```
er |> pull(visits) |> mean(na.rm=T)
```

```
[1] 7.189247
```

GUT CHECK!

What kind of object do we need to run summary operators like `mean()` ?

- A. A vector of numbers
- B. A vector of characters
- C. A dataset

Summarization on tibbles (data frames)

Summarize the data: **dplyr summarize()** function

summarize works on datasets without pull().

Multiple summary statistics can be calculated at once!

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

Summarize the data: **dplyr summarize()** function

```
er |>
  summarize(mean_visits = mean(visits))

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

er |>
  summarize(mean_visits = mean(visits, na.rm = TRUE))

# A tibble: 1 × 1
  mean_visits
  <dbl>
1     7.19
```

Summarize the data: **dplyr summarize()** function

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

```
er |>
  summarize(mean_visits = mean(visits, na.rm = TRUE),
            median_visits = median(visits, na.rm = TRUE),
            mean_rate = mean(rate, na.rm = TRUE))

# A tibble: 1 × 3
  mean_visits median_visits mean_rate
  <dbl>          <dbl>      <dbl>
1     7.19           0       2.43
```

Summarize the data: **dplyr summarize()** function

Note that `summarize()` creates a separate tibble from the original data.

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(er)
```

| county | rate | lower95cl | upper95cl |
|------------------|----------------|----------------|----------------|
| Length:768 | Min. : 0.000 | Min. : 0.000 | Min. : 0.000 |
| Class :character | 1st Qu.: 0.000 | 1st Qu.: 0.000 | 1st Qu.: 0.000 |
| Mode :character | Median : 0.000 | Median : 0.000 | Median : 0.000 |
| | Mean : 2.431 | Mean : 1.449 | Mean : 3.526 |
| | 3rd Qu.: 3.509 | 3rd Qu.: 1.895 | 3rd Qu.: 5.173 |
| | Max. :89.275 | Max. :43.398 | Max. :151.420 |
| | NA's :303 | NA's :304 | NA's :304 |
| visits | year | | |
| Min. : 0.000 | Min. :2011 | | |
| 1st Qu.: 0.000 | 1st Qu.:2014 | | |
| Median : 0.000 | Median :2016 | | |
| Mean : 7.189 | Mean :2016 | | |
| 3rd Qu.:13.000 | 3rd Qu.:2019 | | |
| Max. :48.000 | Max. :2022 | | |
| NA's :303 | | | |

Summary & Lab Part 1

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

□ [Class Website](#)

□ [Lab](#)

distinct() values

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

```
er |>
  distinct(county)
```

```
# A tibble: 64 × 1
  county
  <chr>
  1 Adams
  2 Alamosa
  3 Arapahoe
  4 Archuleta
  5 Baca
  6 Bent
  7 Boulder
  8 Broomfield
  9 Chaffee
 10 Cheyenne
# ... 54 more rows
```

How many **distinct()** values?

`n_distinct()` tells you the number of unique elements (including NA).

It needs a vector so you *must pull the column first!*

```
er |>  
  pull(county) |>  
  n_distinct()
```

```
[1] 64
```

Use **count()** to return row count per category.

```
er |> count(county)
```

```
# A tibble: 64 × 2
```

```
  county      n
  <chr>     <int>
1 Adams        12
2 Alamosa      12
3 Arapahoe      12
4 Archuleta      12
5 Baca          12
6 Bent          12
7 Boulder        12
8 Broomfield      12
9 Chaffee        12
10 Cheyenne       12
# ... 54 more rows
```

Looks like 12 rows/observations per county!

Multiple columns listed further subdivides the `count()`

```
er |> count(county, year)
```

```
# A tibble: 768 × 3
```

| | county | year | n |
|----|--------|-------|-------|
| | <chr> | <dbl> | <int> |
| 1 | Adams | 2011 | 1 |
| 2 | Adams | 2012 | 1 |
| 3 | Adams | 2013 | 1 |
| 4 | Adams | 2014 | 1 |
| 5 | Adams | 2015 | 1 |
| 6 | Adams | 2016 | 1 |
| 7 | Adams | 2017 | 1 |
| 8 | Adams | 2018 | 1 |
| 9 | Adams | 2019 | 1 |
| 10 | Adams | 2020 | 1 |

... 758 more rows

Looks like 1 row/observation per county and year!

GUT CHECK!

The `count()` function can help us tally:

- A. Sample size
- B. Rows per each category
- C. How many categories

Grouping

Goal

We want to find the mean number of ER visits per year in the dataset.

How do we do this?

Perform Operations By Groups: dplyr

First, let's group the data.

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

```
er_grouped <- er |> group_by(year)
er_grouped

# A tibble: 768 × 6
# Groups:   year [12]
  county  rate lower95cl upper95cl visits  year
  <chr>   <dbl>    <dbl>     <dbl>   <dbl> <dbl>
1 Adams    6.73      NA        9.24    29    2011
2 Adams    4.84      2.85      NA       23    2012
3 Adams    6.84      4.36      9.31    31    2013
4 Adams    3.08      1.71      4.85    15    2014
5 Adams    3.36      1.89      5.23    16    2015
6 Adams    8.85      6.12     11.6     42    2016
7 Adams    6.63      4.29      8.98    32    2017
8 Adams    7.11      4.77      9.44    37    2018
9 Adams    6.76      4.53      8.99    36    2019
10 Adams   4.76      2.82      6.70    24    2020
# ... 758 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 `visits` by group:

```
er_grouped |>
  summarize(avg_visits = mean(visits, na.rm = TRUE))
```

```
# A tibble: 12 × 2
  year  avg_visits
  <dbl>     <dbl>
1 2011      5.20
2 2012      5.89
3 2013      5.63
4 2014      4.12
5 2015      6.4
6 2016     10.1
7 2017      7.24
8 2018     11.7
9 2019      9.12
10 2020      6.26
11 2021      8.06
12 2022      9.29
```

Do it in one step: use

> to string these together!

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

```
er |>
  group_by(year) |>
  summarize(avg_visits = mean(visits, na.rm = TRUE))
```

```
# A tibble: 12 × 2
  year  avg_visits
  <dbl>     <dbl>
1 2011      5.20
2 2012      5.89
3 2013      5.63
4 2014      4.12
5 2015      6.4
6 2016     10.1
7 2017      7.24
8 2018     11.7
9 2019      9.12
10 2020      6.26
11 2021      8.06
12 2022      9.29
```

Group by as many variables as you want

group_by county and year:

```
er |>
  group_by(year, county) |>
  summarize(avg_visits = mean(visits, na.rm = TRUE))

# A tibble: 768 × 3
# Groups:   year [12]
  year county    avg_visits
  <dbl> <chr>        <dbl>
1 2011 Adams         29
2 2011 Alamosa       0
3 2011 Arapahoe      33
4 2011 Archuleta     0
5 2011 Baca          0
6 2011 Bent           0
7 2011 Boulder        12
8 2011 Broomfield     NaN
9 2011 Chaffee        0
10 2011 Cheyenne       0
# ... 758 more rows
```

Counting rows/observations

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

```
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     64  5.20
2 2012     64  5.89
3 2013     64  5.63
4 2014     64  4.12
5 2015     64  6.4
6 2016     64 10.1
7 2017     64  7.24
8 2018     64 11.7
9 2019     64  9.12
10 2020    64  6.26
11 2021    64  8.06
12 2022    64  9.29
```

Counting: **count()** and **n()**

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

```
# Here we use count()
er |> count(year)
```

```
# A tibble: 12 × 2
```

| | year | n |
|----|-------|-------|
| | <dbl> | <int> |
| 1 | 2011 | 64 |
| 2 | 2012 | 64 |
| 3 | 2013 | 64 |
| 4 | 2014 | 64 |
| 5 | 2015 | 64 |
| 6 | 2016 | 64 |
| 7 | 2017 | 64 |
| 8 | 2018 | 64 |
| 9 | 2019 | 64 |
| 10 | 2020 | 64 |
| 11 | 2021 | 64 |
| 12 | 2022 | 64 |

Counting: **count()** and **n()**

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

```
# n() with summarize  
er |> group_by(year) |> summarize(n())
```

```
# A tibble: 12 × 2  
  year `n()`  
  <dbl> <int>  
1 2011    64  
2 2012    64  
3 2013    64  
4 2014    64  
5 2015    64  
6 2016    64  
7 2017    64  
8 2018    64  
9 2019    64  
10 2020   64  
11 2021   64  
12 2022   64
```

A few miscellaneous topics ..

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

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

```
er_year <- er |> pull(year) # pull() to make a vector
```

```
er_year |> unique() # similar to distinct()
```

```
[1] 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022
```

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

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

```
er_year |> unique() |> length() # similar to n_distinct()
```

```
[1] 12
```

summary() vs. summarize()

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

Summary

- `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
- `summarize()` with `n()` gives the count (NAs included)

Lab Part 2

- [Class Website](#)
- [Lab](#)
- [Day 4 Cheatsheet](#)
- [Posit's data transformation Cheatsheet](#)

For more advanced learning:

- [https://www.danielsjoberg.com/gtsummary/ for tables](https://www.danielsjoberg.com/gtsummary/)
- extra slides in this file.



Image by [Gerd Altmann from Pixabay](#)

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

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

rowMeans() example

Get means for each row.

Let's see what the mean CO2 emissions is across the years 2010-2014 for each row (country):

```
yearly_co2 |>  
  select(starts_with("201")) |>  
  rowMeans(na.rm = TRUE) |>  
  head(n = 5)
```

```
[1] 10254   5106 129800     487  32040
```

```
yearly_co2 |>  
  group_by(country) |>  
  summarize(mean = rowMeans(across(starts_with("201"))), na.rm = TRUE) |>  
  head(n = 5)
```

```
# A tibble: 5 × 2  
  country      mean  
  <chr>       <dbl>  
1 Afghanistan 10254  
2 Albania     5106  
3 Algeria    129800  
4 Andorra     487  
5 Angola      32040
```

colMeans() example

Get means for each column.

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

```
yearly_co2 |>  
  select(starts_with("201")) |>  
  colMeans(na.rm = TRUE) |>  
  head(n = 5)
```

```
2010      2011      2012      2013      2014  
165334.1 171764.9 174033.4 174856.2 175992.5
```

```
yearly_co2 |>  
  summarize(across(starts_with("201"), ~mean(.x, na.rm = TRUE)))  
  
# A tibble: 1 × 5  
`2010` `2011` `2012` `2013` `2014`  
  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  
1 165334. 171765. 174033. 174856. 175993.
```

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

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

```
er |>
  group_by(year) |>
  summarize(avg_visits = mean(visits, na.rm = TRUE),
            max_visits = max(visits, na.rm = TRUE))
```

is the same as..

```
er |>
  summarize(avg_visits = mean(visits, na.rm = TRUE),
            max_visits = max(visits, na.rm = TRUE),
            .by = year)
```