Accelerated Lecture 6: Grouped Analysis

Harris Coding Camp

Summer 2022

Grouping data with group_by()

Often we want to repeat the same analysis across different subgroups. We can automate that with group_by(). We will:

- summarize by group with group_by() + summarize()
- create new columns with group_by() + mutate()
- filter() data with group specific matching criteria

Grouping data with group_by()

By itself, group_by() doesn't do much. But, once an object is grouped, **all subsequent dplyr functions are run separately "by group"**:

- count() returns the number of observations

```
txhousing %>% count()
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1  8602
txhousing %>% group_by(year) %>% count() %>% glimpse()
#> Rows: 16
#> Columns: 2
#> Groups: year [16]
#> $ year <int> 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2
#> $ n    <int> 552, 552, 552, 552, 552, 552, 552, 552, 552, 552, 552,
```

Grouping data with group_by()

Ungrouped data . . .

```
txhousing %>% glimpse()
#> Rows: 8,602
#> Columns: 9
#> $ city      <chr> "Abilene", "Abilene", "Abilene", "Abi
#> $ year      <int> 2000, 2000, 2000, 2000, 2000, 2000, 20
#> $ month     <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
#> $ sales     <dbl> 72, 98, 130, 98, 141, 156, 152, 131,
#> $ volume    <dbl> 5380000, 6505000, 9285000, 9730000, 10
#> $ median    <dbl> 71400, 58700, 58100, 68600, 67300, 66
#> $ listings  <dbl> 701, 746, 784, 785, 794, 780, 742, 76
#> $ inventory <dbl> 6.3, 6.6, 6.8, 6.9, 6.8, 6.6, 6.2, 6.
#> $ date      <dbl> 2000.000, 2000.083, 2000.167, 2000.25
```

Grouping data with group_by()

group_by() adds meta-data indicating which rows/observations belong to which group.

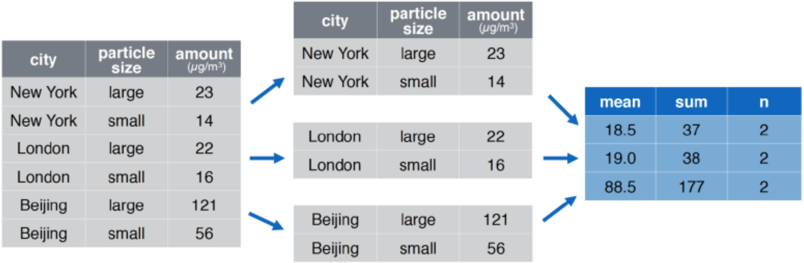
```
group_by(txhousing, city) %>%  glimpse()
#> Rows: 8,602
#> Columns: 9
#> Groups: city [46]
#> $ city      <chr> "Abilene", "Abilene", "Abilene", "Abi
#> $ year      <int> 2000, 2000, 2000, 2000, 2000, 2000, 20
#> $ month     <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
#> $ sales     <dbl> 72, 98, 130, 98, 141, 156, 152, 131, 
#> $ volume    <dbl> 5380000, 6505000, 9285000, 9730000, 10
#> $ median    <dbl> 71400, 58700, 58100, 68600, 67300, 66
#> $ listings  <dbl> 701, 746, 784, 785, 794, 780, 742, 76
#> $ inventory <dbl> 6.3, 6.6, 6.8, 6.9, 6.8, 6.6, 6.2, 6.
#> $ date      <dbl> 2000.000, 2000.083, 2000.167, 2000.25
```

We can group by multiple columns

We now have 46 cities $\times$ 16 years $=$ 736 groups!

```
group_by(txhousing, city, year) %>%  glimpse()
#> Rows: 8,602
#> Columns: 9
#> Groups: city, year [736]
#> $ city      <chr> "Abilene", "Abilene", "Abilene", "Abi
#> $ year      <int> 2000, 2000, 2000, 2000, 2000, 2000, 20
#> $ month     <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
#> $ sales     <dbl> 72, 98, 130, 98, 141, 156, 152, 131, 1
#> $ volume    <dbl> 5380000, 6505000, 9285000, 9730000, 10
#> $ median    <dbl> 71400, 58700, 58100, 68600, 67300, 669
#> $ listings  <dbl> 701, 746, 784, 785, 794, 780, 742, 765
#> $ inventory <dbl> 6.3, 6.6, 6.8, 6.9, 6.8, 6.6, 6.2, 6.4
#> $ date      <dbl> 2000.000, 2000.083, 2000.167, 2000.250
```

Grouped summary with `group_by() %>% summarize()`



city	particle size	amount ($\mu g/m^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	particle size	amount ($\mu g/m^3$)
New York	large	23
New York	small	14

London	large	22
London	small	16

Beijing	large	121
Beijing	small	56

mean	sum	n
18.5	37	2
19.0	38	2
88.5	177	2

In this example, we want to calculate the mean and sum of `amount` by city

- we must repeat the same analysis across different groups (i.e. cities)
- Using `group_by() %>% summarize()` makes it a lot easier!

Grouped summary with group_by() %>% summarize()

Let's see the upgrouped summary statistics first:

```
txhousing %>%
  summarize(total_sales = sum(sales, na.rm = TRUE),
            total_volume = sum(volume, na.rm = TRUE),
            mean_house_price = total_volume / total_sales)
#> # A tibble: 1 x 3
#>   total_sales total_volume mean_house_price
#>         <dbl>        <dbl>            <dbl>
#> 1     4415202 858502159353          194442.
```

Grouped summary with group_by() %>% summarize()

Use case: You want summary statistics for all subsets (i.e. each year):

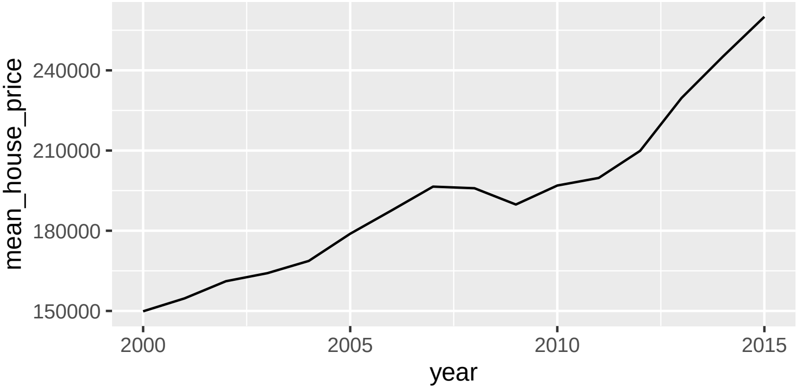
- summarize() collapses a data frame to a single row for each group
- The count function n() takes no arguments and returns the size of a group

```
annual_housing_prices <-
  txhousing %>%
  group_by(year) %>%
  summarize(n_within_group = n(),
            total_sales = sum(sales, na.rm = TRUE),
            total_volume = sum(volume, na.rm = TRUE),
            mean_house_price = total_volume / total_sales)

head(annual_housing_prices, n=5)
#> # A tibble: 5 x 5
#>    year n_within_group total_sales total_volume mean_house_price
#>   <int>          <int>       <dbl>        <dbl>            <dbl>
#> 1  2000            552      222483  33342410971          149865.
#> 2  2001            552      231453  35804815138          154696.
#> 3  2002            552      234600  37798888462          161121.
#> 4  2003            552      253909  41674204834          164130.
#> 5  2004            552      283999  47913188880          168709.
```

How have Texas housing prices changed over time?

```
annual_housing_prices %>%
  ggplot(aes(x = year,
             y = mean_house_price)) +
  geom_line()
```



Grouped summary with group_by() + summarize()

Use case: You want summary statistics for certain subsets (each year) in a specific city (e.g. Houston):

```
txhousing %>%
  filter(city == "Houston") %>%
  group_by(year) %>%
  summarize(n_within_group = n(),
            total_sales = sum(sales, na.rm = TRUE),
            total_volume = sum(volume, na.rm = TRUE),
            mean_house_price = total_volume / total_sales)
#> # A tibble: 16 x 5
#>     year n_within_group total_sales total_volume mean_house_price
#>    <int>          <int>       <dbl>        <dbl>            <dbl>
#>  1  2000             12       52459   8041166317          153285.
#>  2  2001             12       53856   8541022943          158590.
#>  3  2002             12       56563   9486396667          167714.
#>  4  2003             12       60732  10417774768          171537.
#>  5  2004             12       66979  11776381072          175822.
#>  6  2005             12       72800  13504202605          185497.
#>  7  2006             12       80994  15816104590          195275.
#>  8  2007             12       77668  15789736644          203298.
#>  9  2008             12       65169  13396719487          205569.
#> 10  2009             12       60106  12035965014          200246.
```

Grouped summary with group_by() + summarize()

Use case: You want summary statistics for certain subsets (each year, each city) of the data.

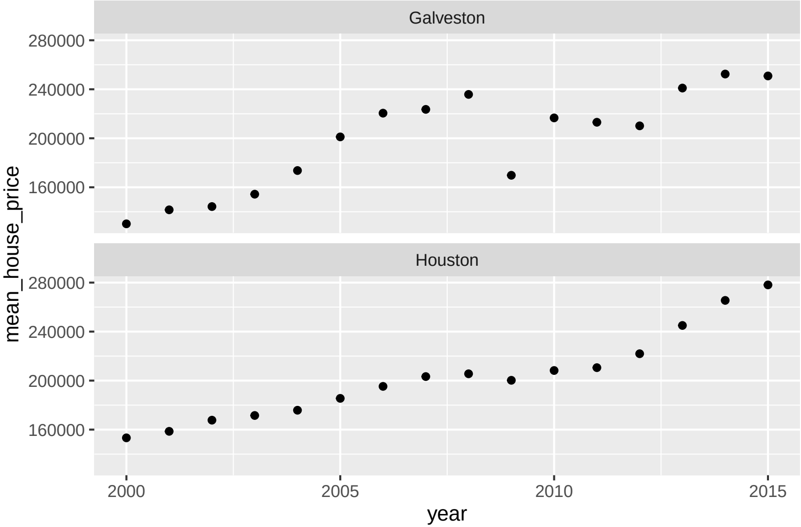
```
annual_city_housing_prices <-
  txhousing %>%
    group_by(city, year) %>%
    summarize(total_sales = sum(sales, na.rm = TRUE),
              total_volume = sum(volume, na.rm = TRUE),
              mean_house_price = total_volume / total_sales)

head(annual_city_housing_prices, n=5)
#> # A tibble: 5 x 5
#> # Groups:   city [1]
#>   city     year total_sales total_volume mean_house_price
#>   <chr>   <int>       <dbl>        <dbl>            <dbl>
#> 1 Abilene  2000        1375    108575000           78964.
#> 2 Abilene  2001        1431    114365000           79920.
#> 3 Abilene  2002        1516    118675000           78282.
#> 4 Abilene  2003        1632    135675000           83134.
#> 5 Abilene  2004        1830    159670000           87251.
```

How have Texas housing prices changed over time in certain cities?

```r
txhousing %>%
  group_by(city, year) %>%
  summarize(total_sales = sum(sales, na.rm = TRUE),
            total_volume = sum(volume, na.rm = TRUE),
            mean_house_price = total_volume / total_sales) %>%
  filter(city %in% c("Houston",  "Galveston")) %>%
  ggplot(aes(x = year, y = mean_house_price)) +
    geom_point() +
    facet_wrap(facets = "city", nrow = 2)
```

How have Texas housing prices changed over time in certain cities?



Grouping + Summarizing: Base R vs Tidyverse

Base R:

```
# use formula to indicate grouping vars
aggregate(formula = sales ~ city + year,
          data = txhousing,
          FUN = mean)
aggregate(formula = sales ~ city + year,
          data = txhousing,
          FUN = sd)
```

Tidyverse:

```
txhousing %>%
  group_by(city, year) %>%
  summarize(mean_sales = mean(sales, na.rm = TRUE),
            sd_sales = sd(sales, na.rm = TRUE))
```

Ungrouping data

To get rid of groups, use `ungroup()`

```
txhousing_grouped <- group_by(txhousing, year)
```

```
txhousing_grouped %>%
  ungroup() %>%
  summarize(total_sales = sum(sales, na.rm = TRUE))
#> # A tibble: 1 x 1
#>   total_sales
#>         <dbl>
#> 1     4415202
```

What's going on here?

```
txhousing_grouped %>%
  select(-year) %>%
  head()
#> # A tibble: 6 x 9
#> # Groups:   year [1]
#>    year city    month sales   volume median listings in
#>   <int> <chr>   <int> <dbl>    <dbl>  <dbl>    <dbl>
#> 1  2000 Abilene     1    72  5380000  71400      701
#> 2  2000 Abilene     2    98  6505000  58700      746
#> 3  2000 Abilene     3   130  9285000  58100      784
#> 4  2000 Abilene     4    98  9730000  68600      785
#> 5  2000 Abilene     5   141 10590000  67300      794
#> 6  2000 Abilene     6   156 13910000  66900      780
```

grouped data require the grouping variable

We got the message:

```
Adding missing grouping variables: ‘year‘
```

```
txhousing_grouped %>%
  ungroup() %>%
  select(-year) %>%
  head()
#> # A tibble: 6 x 8
#>   city    month sales   volume median listings inventor
#>   <chr>   <int> <dbl>    <dbl>  <dbl>    <dbl>     <dbl
#> 1 Abilene     1    72  5380000  71400      701       6.
#> 2 Abilene     2    98  6505000  58700      746       6.
#> 3 Abilene     3   130  9285000  58100      784       6.
#> 4 Abilene     4    98  9730000  68600      785       6.
#> 5 Abilene     5   141 10590000  67300      794       6.
#> 6 Abilene     6   156 13910000  66900      780       6.
```

Try it yourself

1. Using `txhousing`, filter observations where `city` is Brazoria County and group by `year`.
2. Next, create a variable that represents the total sales in each year.
3. Plot the total sales over time.
4. Create two variables that represent the sum of missing & non-missing obs for `sales` in Brazoria County.

- Whenever you do any aggregation, it's always a good idea to include either a count of missing values `sum(is.na(x))` or a count of non-missing values `sum(!is.na(x))`. That way, you can check that you are not making conclusions based on very small amounts of data!

group_by() %>% mutate() creates new columns with groups in mind

Often we want the “summary” information in the data context.

```r
ex <- tibble(period = c(1, 2, 1, 2),
             group = c("a", "a", "b", "b"),
             x = c(3, 1, 11, 13),)

ex %>%
  group_by(group) %>%
  mutate(group_mean = mean(x))
#> # A tibble: 4 x 4
#> # Groups:   group [2]
#>   period group     x group_mean
#>    <dbl> <chr> <dbl>      <dbl>
#> 1      1 a         3          2
#> 2      2 a         1          2
#> 3      1 b        11         12
#> 4      2 b        13         12
```

Grouped `mutate`: differences

Use case: You want to work with differences.

```
ex %>%
  group_by(group) %>%
  mutate(x_lag = lag(x),
          x_diff = x - lag(x))
#> # A tibble: 4 x 5
#> # Groups:   group [2]
#>   period group     x x_lag x_diff
#>    <dbl> <chr> <dbl> <dbl>  <dbl>
#> 1      1 a         3    NA     NA
#> 2      2 a         1     3     -2
#> 3      1 b        11    NA     NA
#> 4      2 b        13    11      2
```

Why is this wrong?

```
ex %>%
  mutate(x_lag = lag(x),
         x_diff = x - lag(x))
#> # A tibble: 4 x 5
#>   period group     x x_lag x_diff
#>    <dbl> <chr> <dbl> <dbl>  <dbl>
#> 1      1 a         3    NA     NA
#> 2      2 a         1     3     -2
#> 3      1 b        11     1     10
#> 4      2 b        13    11      2
```

Grouped mutate: differences

Use case: You want to work with differences. (Try running the code without group_by() and carefully compare the results.)

```
july_texas_txhousing <-
  txhousing %>%
    filter(month == 7) %>%
    select(city, year, sales)

differenced_data <-
  july_texas_txhousing %>%
    group_by(city) %>%
    mutate(last_year_sales = lag(sales),
           diff_sales = sales - lag(sales))
```

Grouped `mutate`: differences

Use case: You want to work with differences between sales in different years.[1]

```
differenced_data %>% head(5)
#> # A tibble: 5 x 5
#> # Groups:   city [1]
#>   city     year sales last_year_sales diff_sales
#>   <chr>   <int> <dbl>           <dbl>      <dbl>
#> 1 Abilene  2000   152              NA         NA
#> 2 Abilene  2001   134             152        -18
#> 3 Abilene  2002   159             134         25
#> 4 Abilene  2003   171             159         12
#> 5 Abilene  2004   176             171          5
```

[1] `lag()`'s sibling is `lead()` which will give you data from the following year.

Grouped `mutate`: other window functions

- See the “Data transformation with dplyr” cheatsheet (page 2) for more vectorized window functions.

```
ex %>%
  group_by(group) %>%
  mutate(cumulative = cumsum(x),
         # comparing values to summaries
         centered = (x - mean(x)))
#> # A tibble: 4 x 5
#> # Groups:   group [2]
#>   period group     x cumulative centered
#>    <dbl> <chr> <dbl>      <dbl>    <dbl>
#> 1      1 a         3          3        1
#> 2      2 a         1          4       -1
#> 3      1 b        11         11       -1
#> 4      2 b        13         24        1
```

Grouped mutate: ranking

```
ex %>%
  group_by(group) %>%
  mutate(rank = rank(desc(x))) %>%
  arrange(group, rank)
#> # A tibble: 4 x 4
#> # Groups:   group [2]
#>   period group     x  rank
#>    <dbl> <chr> <dbl> <dbl>
#> 1      1 a         3     1
#> 2      2 a         1     2
#> 3      2 b        13     1
#> 4      1 b        11     2
```

Aside: Grouped `arrange`

Say you want to order the data without explicitly adding a rank.

- Almost all `dplyr` function will operate group-by-group on grouped data.
- `arrange()` is an exception

```
ex %>%
  group_by(group) %>%
  arrange(row_number(desc(x)))
#> # A tibble: 4 x 3
#> # Groups:   group [2]
#>   period group     x
#>    <dbl> <chr> <dbl>
#> 1      2 b        13
#> 2      1 b        11
#> 3      1 a         3
#> 4      2 a         1
```

Aside: Grouped arrange

However, you only need to add the grouping column(s) or .by_group = TRUE to get the desired output.

```
ex %>%
  arrange(group, row_number(desc(x)))
```

```
ex %>%
  group_by(group) %>%
  # this option is nice if you have many grouping cols
  arrange(row_number(desc(x)), .by_group = TRUE)
#> # A tibble: 4 x 3
#> # Groups:   group [2]
#>   period group     x
#>    <dbl> <chr> <dbl>
#> 1      1 a         3
#> 2      2 a         1
#> 3      2 b        13
#> 4      1 b        11
```

Grouped `mutate`: ranking

Use case: You want to rank sales within group. (Try running the code without `group_by()` and carefully compare the results.)

```r
ranked_data <-
july_texas_txhousing %>%
  group_by(year) %>%
  mutate(sales_rank = rank(desc(sales)))
```

Grouped `mutate`: ranking

Use case: You want to rank sales within group.[2]

```
ranked_data %>% arrange(year, sales_rank) %>% head(10)
#> # A tibble: 10 x 4
#> # Groups:   year [1]
#>    city                year sales sales_rank
#>    <chr>              <int> <dbl>      <dbl>
#>  1 Houston             2000  5009          1
#>  2 Dallas              2000  4276          2
#>  3 Austin              2000  1818          3
#>  4 San Antonio         2000  1508          4
#>  5 Collin County       2000  1007          5
#>  6 Fort Bend           2000   753          6
#>  7 NE Tarrant County   2000   686          7
#>  8 Denton County       2000   638          8
#>  9 Fort Worth          2000   548          9
#> 10 Montgomery County   2000   463         10
```

[2] R has a variety of related functions see `?ranking`

Grouped filter

Use case: You want to work with the top 5 cities for each year.

```
ranked_data %>%
  # we already added ranks!
  filter(sales_rank <= 5) %>%
  arrange(year, sales_rank) %>%
  head()
#> # A tibble: 6 x 4
#> # Groups:   year [2]
#>   city          year sales sales_rank
#>   <chr>        <int> <dbl>      <dbl>
#> 1 Houston       2000  5009          1
#> 2 Dallas        2000  4276          2
#> 3 Austin        2000  1818          3
#> 4 San Antonio   2000  1508          4
#> 5 Collin County 2000  1007          5
#> 6 Houston       2001  5424          1
```

Grouped filter

Use case: You want to work with the top 5 cities for each year.

```
july_texas_txhousing %>%
  group_by(year) %>%
  # we don't need sales_rank to filter by ranks!
  filter(rank(desc(sales)) <=  5) %>%
  arrange(year, desc(sales)) %>%
  head()
#> # A tibble: 6 x 3
#> # Groups:   year [2]
#>   city           year sales
#>   <chr>         <int> <dbl>
#> 1 Houston        2000  5009
#> 2 Dallas         2000  4276
#> 3 Austin         2000  1818
#> 4 San Antonio    2000  1508
#> 5 Collin County  2000  1007
#> 6 Houston        2001  5424
```

count() is a useful short cut

Based on what you know about txhousing. Can you tell what count() does?

```
txhousing %>%
  count(city, year) %>%
  head(5)
#> # A tibble: 5 x 3
#>   city     year     n
#>   <chr>   <int> <int>
#> 1 Abilene  2000    12
#> 2 Abilene  2001    12
#> 3 Abilene  2002    12
#> 4 Abilene  2003    12
#> 5 Abilene  2004    12
```

count() is a useful short cut

count() does the following:

- applies group_by() on the specified column(s)
- applies summarize() and returns column n with the number of rows per group
- applies ungroup()

```
txhousing %>%
  group_by(city, year) %>%
  summarize(n = n()) %>%
  ungroup() %>%
  head(5)
#> # A tibble: 5 x 3
#>   city     year     n
#>   <chr>   <int> <int>
#> 1 Abilene  2000    12
#> 2 Abilene  2001    12
#> 3 Abilene  2002    12
#> 4 Abilene  2003    12
#> 5 Abilene  2004    12
```

add_count() is a useful short cut

Based on what you know about txhousing. Can you tell what add_count() does?

```
txhousing %>%
  select(city, year, sales) %>%
  add_count(city, year) %>%
  head(5)
#> # A tibble: 5 x 4
#>   city     year sales     n
#>   <chr>   <int> <dbl> <int>
#> 1 Abilene  2000    72    12
#> 2 Abilene  2000    98    12
#> 3 Abilene  2000   130    12
#> 4 Abilene  2000    98    12
#> 5 Abilene  2000   141    12
```

add_count() is a useful short cut

add_count() does the following:

- applies group_by() on the specified column(s)
- applies mutate() to add a new column n with the counts of rows per group while retaining all the other data frame columns
- applies ungroup()

```
txhousing  %>%
  select(city, year, sales) %>%
  group_by(city, year) %>%
  mutate(n = n()) %>%
  ungroup() %>%
  head(5)
#> # A tibble: 5 x 4
#>   city     year sales     n
#>   <chr>   <int> <dbl> <int>
#> 1 Abilene  2000    72    12
#> 2 Abilene  2000    98    12
#> 3 Abilene  2000   130    12
#> 4 Abilene  2000    98    12
#> 5 Abilene  2000   141    12
```

Try it yourself: Setup

midwest is a data set that comes bundled with tidyverse. First, let's calculate the total population of Ohio in the following way:

```
midwest %>% filter(state == "OH") %>%
  summarize(total_population = sum(poptotal))
#> # A tibble: 1 x 1
#>   total_population
#>              <int>
#> 1         10847115
```

With group_by, you can calculate the total population of all the states at once!

```
midwest %>% group_by(state) %>%
  summarize(total_population = sum(poptotal))
#> # A tibble: 5 x 2
#>   state total_population
#>   <chr>            <int>
#> 1 IL            11430602
#> 2 IN             5544159
#> 3 MI             9295297
#> 4 OH            10847115
#> 5 WI             4891769
```

Try it yourself: group_by()

1. For each state in the midwest data, calculate the total area.
2. For each state in the midwest data, calculate the proportion of counties that are in a metro area (inmetro).[3]
3. Add a column to midwest called pop_state that equals the state population. Compare your result to what you calculated early.

```
# fill in the ... with approriate code
midwest %>%
  group_by( ... ) %>%
  mutate(pop_state = ... )
```

4. Building off the previous question, create a column that shows the number of people living below the poverty line (percbelowpoverty) in each county.

[3] Recall that mean() of a column of 0 and 1s tell you the proportion of 1s.

Try it yourself: count()

5. Reproduce this table using count().

```
#> # A tibble: 2 x 2
#>   inmetro     n
#>     <int> <int>
#> 1       0   287
#> 2       1   150
```

Recap: Analysis by group with `dplyr`

This lesson gave you an idea about how to:

- summarize data by group with `group_by()` + `summarize()`
- created new columns with `group_by()` + `mutate()`
    - we saw `lag()` and `rank()`, but you could get also add group-level stats like `mean()` and `median()`
- `filter()` data with group specific matching criteria
- use `count()` and `add_count()` as short cuts for getting group level counts

Next steps:

Lab:

- Today: Grouped analysis

I can streamline analysis of subgroup data using group_by() and dplyr verbs

Lecture:

- Tomorrow: Writing your own functions!