Accelerated Lecture 6: Grouped Analysis

Harris Coding Camp

Summer 2022

Remember the world inequality data lab?

First, you did the analysis for France

```
french_data <-
  wid_data |>
    filter(type == "Net personal wealth",
           country == "France") |>
    mutate(perc_national_wealth = value * 100)

french_data |>
  filter(percentile == "p90p100") |>
  filter(between(year, 1995, 2010)) |>
  summarize(wealth_share_of_top_10 =
              mean(perc_national_wealth, na.rm=TRUE))
```

```
## # A tibble: 1 x 1
##   wealth_share_of_top_10
##                    <dbl>
## 1                   54.4
```

Then, you did the analysis for Russia

```r
russian_data <-
  wid_data |>
    filter(type == "Net personal wealth",
           country == "Russian Federation") |>
    mutate(perc_national_wealth = value * 100)

russian_data |>
  filter(percentile == "p90p100") |>
  filter(between(year, 1995, 2010)) |>
  summarize(wealth_share_of_top_10 =
              mean(perc_national_wealth, na.rm=TRUE))
```

```
## # A tibble: 1 x 1
##   wealth_share_of_top_10
##                    <dbl>
## 1                   63.3
```

Then, we asked you try for Korea . . .

Could there be a way to work on each country's subset of data at the same time?

group_by() is the answer!

```
wid_data |>
    mutate(perc_national_wealth = value * 100) |>
    filter(type == "Net personal wealth",
           percentile == "p90p100",
           between(year, 1995, 2010)) |>
  group_by(country) |>
  summarize(wealth_share_of_top_10 =
    round(mean(perc_national_wealth, na.rm=TRUE), 1))
```

group_by() is the answer!

```
## # A tibble: 8 x 2
##   country            wealth_share_of_top_10
##   <chr>                               <dbl>
## 1 China                                50.4
## 2 France                               54.4
## 3 India                                55.6
## 4 Korea                                63.7
## 5 Russian Federation                   63.3
## 6 South Africa                         85.4
## 7 United Kingdom                       50.7
## 8 USA                                  67.9
```

Analyzing data by groups

1. **"Split"** data into groups

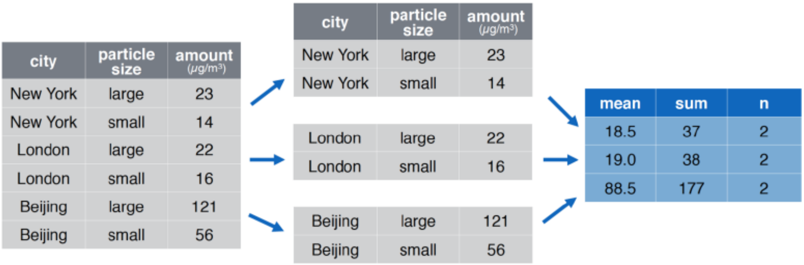
- done silently with `group_by()`

1. **Apply** - a function to each group

- use `summarize()`, `mutate()` or any dplyr verb

1. **Combine** - the results back into a tibble

split - apply - combine



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

Ungrouped data ...

```
txhousing |> glimpse()
```

```
## Rows: 8,602
## Columns: 9
## $ city      <chr> "Abilene", "Abilene", "Abilene", "Abil
## $ 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
```

“split” the data with group_by() (subtle)

- group_by() adds some meta-data about groups
- otherwise, no difference

```
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, 1
## $ 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
```

Let's make a tiny example

```r
ex_data <-
  tibble(period = rep(c(1, 2), 2),
         group_col = c("a", "a", "b", "b"),
         x = c(10, 6, 7, 7))

str(ex_data)
```

```
## tibble [4 x 3] (S3: tbl_df/tbl/data.frame)
##  $ period   : num [1:4] 1 2 1 2
##  $ group_col: chr [1:4] "a" "a" "b" "b"
##  $ x        : num [1:4] 10 6 7 7
```

create a hidden tibble that lists the `.rows` that belong to each group

```
grouped_ex_data <-
  group_by(ex_data, group_col)
str(grouped_ex_data)
```

```
## grouped_df [4 x 3] (S3: grouped_df/tbl_df/tbl/data.frame
##  $ period   : num [1:4] 1 2 1 2
##  $ group_col: chr [1:4] "a" "a" "b" "b"
##  $ x        : num [1:4] 10 6 7 7
##  - attr(*, "groups")= tibble [2 x 2] (S3: tbl_df/tbl/dat
##   ..$ group_col: chr [1:2] "a" "b"
##   ..$ .rows    : list<int> [1:2]
##   .. ..$ : int [1:2] 1 2
##   .. ..$ : int [1:2] 3 4
##   .. ..@ ptype: int(0)
##   ..- attr(*, ".drop")= logi TRUE
```

Start with upgrouped summary statistics

- `summarize()` collapses data to one row
- The count function `n()` takes no arguments and returns the size of a group

```
txhousing |>
  summarize(n = n(),
            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 4
##       n total_sales total_volume mean_house_price
##   <int>       <dbl>        <dbl>            <dbl>
## 1  8602     4415202 858502159353          194442.
```

apply and **combine** summarize()

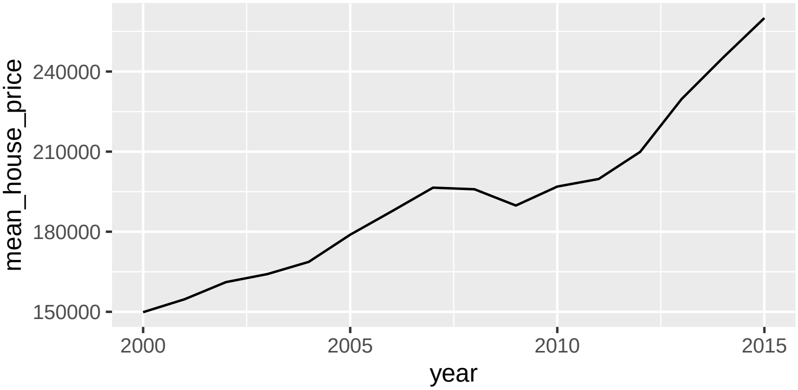
now summarize() collapses a data to one row *for each group*

```
annual_housing_prices <-
  txhousing |>
  group_by(year) |>    # Let's see summary for each year!
  summarize(group_n = 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 = 3)
```

```
## # A tibble: 3 x 5
##    year group_n 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.
```

Visualizing our summarized data with ggplot

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



What if we want the same trend by city?

We can filter the data and then **split-apply-combine**

```
txhousing |>
  filter(city == "Houston") |>
  group_by(year) |>
  summarize(group_n = n(),
            total_sales = sum(sales, na.rm = TRUE),
            total_volume = sum(volume, na.rm = TRUE),
            mean_house_price = total_volume / total_sales) |>
  head(3)
```

```
## # A tibble: 3 x 5
##    year group_n 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.
```

What if we want annual data for every Texas city?

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
```

Voilà

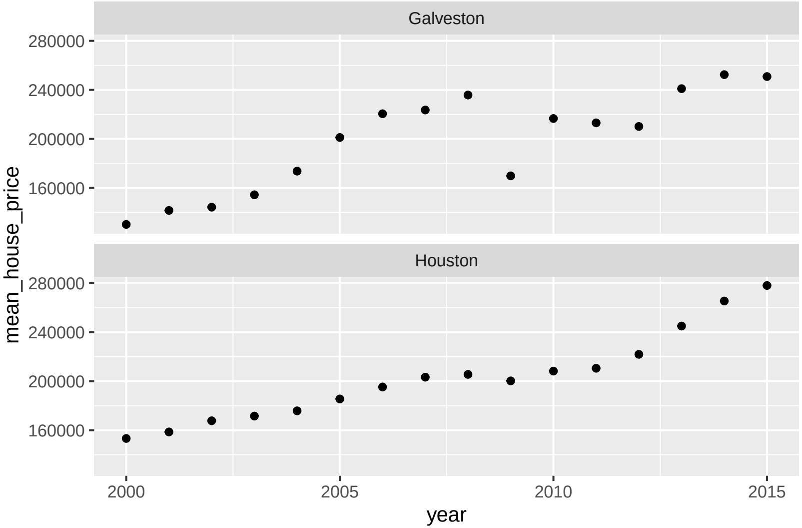
```r
annual_city_housing_prices <-
  txhousing |>
    group_by(city, year) |> # one tiny change!
    summarize(total_sales = sum(sales, na.rm = TRUE),
              total_volume = sum(volume, na.rm = TRUE),
              mean_house_price = total_volume / total_sales)

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?

```
annual_city_housing_prices |>
  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

Tidyverse:

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

Base R:

```
# use formula to indicate grouping vars
aggregate(sales ~ city + year,
          data = txhousing,
          FUN = mean)
aggregate(txhousing$sales,
          by = list(city = txhousing$city, year = txhousing$year),
          FUN = sd, na.rm = TRUE)
```

Ungrouping data

```
class(txhousing)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

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

```
## [1] "grouped_df" "tbl_df"     "tbl"        "data.frame"
```

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

```
txhousing_grouped |>  ungroup() |> class()
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

What's going on here?

```
txhousing_grouped |>
  select(-c(year, month, date, inventory)) |>
  head()
```

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

grouped data require the grouping variable

We got the message:

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

```
txhousing_grouped |>
  ungroup() |>
  select(-c(year, month, date, inventory)) |>
  head()
```

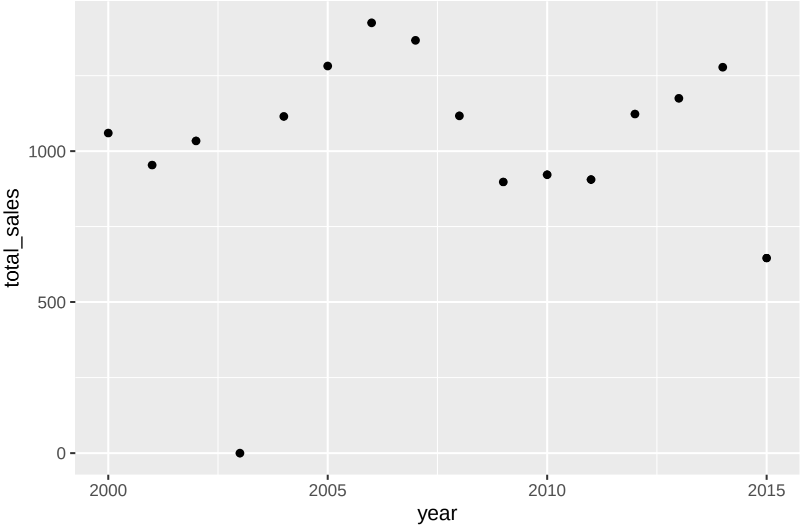
```
## # A tibble: 6 x 5
##   city    sales   volume median listings
##   <chr>   <dbl>    <dbl>  <dbl>    <dbl>
## 1 Abilene    72  5380000  71400      701
## 2 Abilene    98  6505000  58700      746
## 3 Abilene   130  9285000  58100      784
## 4 Abilene    98  9730000  68600      785
## 5 Abilene   141 10590000  67300      794
## 6 Abilene   156 13910000  66900      780
```

Try it yourself

`txhousing` loads with the tidyverse

1. Filter observations where `city` is "Brazoria County"
2. Next, determine total sales in each year
3. Plot the total sales over time
4. Create two variables to show the number of missing & non-missing obs for `sales` in Brazoria County.

What happened in 2003?



- Whenever you aggregate data, pay attention to `NA`s
- count missing values `sum(is.na(x))` or non-missing values `sum(!is.na(x))`

How do we get the summarized values back into the main dataset?

```
ex_data |>
  group_by(group_col) |>
  summarize(group_mean = mean(x))
```

```
## # A tibble: 2 x 2
##   group_col group_mean
##   <chr>          <dbl>
## 1 a                  8
## 2 b                  7
```

apply and combine `mutate()`

```
ex_data |>
  group_by(group_col) |>
  mutate(group_mean = mean(x))
```

```
## # A tibble: 4 x 4
## # Groups:   group_col [2]
##   period group_col     x group_mean
##    <dbl> <chr>     <dbl>      <dbl>
## 1      1 a            10          8
## 2      2 a             6          8
## 3      1 b             7          7
## 4      2 b             7          7
```

How would you calculate the change in sales from year to year?

Say your prof gives you the code …

```
txhousing |>
  group_by(city, year) |>
  summarize(sales = sum(sales, na.rm = TRUE)) |>
  group_by(city) |>
  mutate(diff_sales = sales - lag(sales)) |>
head()
```

```
## # A tibble: 6 x 4
## # Groups:   city [1]
##   city      year sales diff_sales
##   <chr>    <int> <dbl>      <dbl>
## 1 Abilene   2000  1375         NA
## 2 Abilene   2001  1431         56
## 3 Abilene   2002  1516         85
## 4 Abilene   2003  1632        116
## 5 Abilene   2004  1830        198
## 6 Abilene   2005  1977        147
```

Let's make sense of the code

1. Why do we need `group_by()`?
2. What is `lag()` and how does it work?

```
txhousing |>
  group_by(city, year) |>
  summarize(sales = sum(sales, na.rm = TRUE)) |>
  # new material ...
  group_by(city) |>
  mutate(diff_sales = sales - lag(sales))
```

What is `lag()` and how does it work?

Consider the time series `x`.

- Think of each row with `x` as the "present"

```
tibble(
  time = 1:6,
  x = c(4,2,4,1,6,8),    # the time series
  lag(x),     # x past (1 row)
  lag(x, 2), # x past (2 rows)
  lead(x))   # x future
```

```
## # A tibble: 6 x 5
##    time     x `lag(x)` `lag(x, 2)` `lead(x)`
##   <int> <dbl>    <dbl>       <dbl>     <dbl>
## 1     1     4       NA          NA         2
## 2     2     2        4          NA         4
## 3     3     4        2           4         1
## 4     4     1        4           2         6
## 5     5     6        1           4         8
```

If the timeseries is out of order ...

`lag()` and `lead()` still work, but not how you want

```
tibble(time = c(2, 3, 5, 1, 4, 6),
       x = c(4,2,4,1,6,8),
       lag(x),  # I take x shift it "down" ...
       lead(x)) # I take x shift it "up" ...
```

```
## # A tibble: 6 x 4
##    time     x `lag(x)` `lead(x)`
##   <dbl> <dbl>    <dbl>     <dbl>
## 1     2     4       NA         2
## 2     3     2        4         4
## 3     5     4        2         1
## 4     1     1        4         6
## 5     4     6        1         8
## 6     6     8        6        NA
```

```
          # regardless of what x is.
```

... put it back in order with arrange()

```
tibble(
  time = c(2, 3, 5, 1, 4, 6),
  x = c(4,2,4,1,6,8),) |>
  arrange(time) |>
  mutate(lag(x),
         lead(x))
```

```
## # A tibble: 6 x 4
##    time     x `lag(x)` `lead(x)`
##   <dbl> <dbl>    <dbl>     <dbl>
## 1     1     1       NA         4
## 2     2     4        1         2
## 3     3     2        4         6
## 4     4     6        2         4
## 5     5     4        6         8
## 6     6     8        4        NA
```

Why do we need group_by()?

We want to learn about growth from period 1 to period 2

- What goes wrong?

```
ex_data |>
  mutate(x_lag = lag(x),
         x_diff = x - lag(x))
```

```
## # A tibble: 4 x 5
##   period group_col     x x_lag x_diff
##    <dbl> <chr>     <dbl> <dbl>  <dbl>
## 1      1 a            10    NA     NA
## 2      2 a             6    10     -4
## 3      1 b             7     6      1
## 4      2 b             7     7      0
```

split-apply-combine to get differences

```
ex_data |>
  group_by(group_col) |>
  mutate(x_lag = lag(x),
         x_diff = x - lag(x))
```

```
## # A tibble: 4 x 5
## # Groups:   group_col [2]
##   period group_col     x x_lag x_diff
##    <dbl> <chr>     <dbl> <dbl>  <dbl>
## 1      1 a            10    NA     NA
## 2      2 a             6    10     -4
## 3      1 b             7    NA     NA
## 4      2 b             7     7      0
```

Grouped `mutate`: other window functions

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

```
ex_data |>
  group_by(group_col) |>
  mutate(cumulative = cumsum(x),
         # comparing values to summaries
         centered = (x - mean(x)))
```

```
## # A tibble: 4 x 5
## # Groups:   group_col [2]
##   period group_col     x cumulative centered
##    <dbl> <chr>     <dbl>      <dbl>    <dbl>
## 1      1 a            10         10        2
## 2      2 a             6         16       -2
## 3      1 b             7          7        0
## 4      2 b             7         14        0
```

Grouped mutate: ranking

```
ex_data |> mutate(rank = row_number(x))
```

```
## # A tibble: 4 x 4
##   period group_col     x  rank
##    <dbl> <chr>     <dbl> <int>
## 1      1 a            10     4
## 2      2 a             6     1
## 3      1 b             7     2
## 4      2 b             7     3
```

```
ex_data |> group_by(group_col) |> mutate(rank = row_number(x))
```

```
## # A tibble: 4 x 4
## # Groups:   group_col [2]
##   period group_col     x  rank
##    <dbl> <chr>     <dbl> <int>
## 1      1 a            10     2
## 2      2 a             6     1
## 3      1 b             7     1
## 4      2 b             7     2
```

Grouped `mutate`: You want to rank sales within group.

- (Try running the code without `group_by()` and carefully compare the results.)

```
ranked_data <-
txhousing |>
  group_by(year, city) |>
  summarize(total_sales = sum(sales, na.rm = TRUE)) |>
  group_by(year) |>
  mutate(sales_rank = rank(desc(total_sales)))
```

Grouped mutate: You want to rank sales within group.[1]

```
ranked_data |> arrange(year, sales_rank) |> head(10)
```

```
## # A tibble: 10 x 4
## # Groups:   year [1]
##     year city               total_sales sales_rank
##    <int> <chr>                    <dbl>      <dbl>
##  1  2000 Houston                  52459          1
##  2  2000 Dallas                   45446          2
##  3  2000 Austin                   18621          3
##  4  2000 San Antonio              15590          4
##  5  2000 Collin County            10000          5
##  6  2000 Fort Bend                 7254          6
##  7  2000 NE Tarrant County         7169          7
##  8  2000 Fort Worth                6380          8
##  9  2000 Denton County             6149          9
## 10  2000 El Paso                   5109         10
```

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

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]
##    year city          total_sales sales_rank
##   <int> <chr>               <dbl>      <dbl>
## 1  2000 Houston             52459          1
## 2  2000 Dallas              45446          2
## 3  2000 Austin              18621          3
## 4  2000 San Antonio         15590          4
## 5  2000 Collin County       10000          5
## 6  2001 Houston             53856          1
```

split-apply-combine `filter`:

- we don't need sales_rank to filter by ranks!

```
txhousing |>
  group_by(year, city) |>
  summarize(total_sales = sum(sales, na.rm = TRUE)) |>
  group_by(year) |>
  # we don't need sales_rank to filter by ranks!
  filter(rank(desc(total_sales)) <=  5) |>
  arrange(year, desc(total_sales)) |>
  head()
```

```
## # A tibble: 6 x 3
## # Groups:   year [2]
##    year city          total_sales
##   <int> <chr>               <dbl>
## 1  2000 Houston             52459
## 2  2000 Dallas              45446
## 3  2000 Austin              18621
## 4  2000 San Antonio         15590
```

Grouped arrange?

`arrange()` doesn't work on groups by default.

- has default argument `.by_group = FALSE`

```
ex_data |>
  group_by(group_col) |>
  arrange(x)
```

```
## # A tibble: 4 x 3
## # Groups:   group_col [2]
##   period group_col     x
##    <dbl> <chr>     <dbl>
## 1      2 a             6
## 2      1 b             7
## 3      2 b             7
## 4      1 a            10
```

Aside: Grouped `arrange`

Set `.by_group = TRUE`

```
ex_data |>
  group_by(group_col) |>
  # this option is nice if you have many grouping cols
  arrange(x, .by_group = TRUE)
```

```
## # A tibble: 4 x 3
## # Groups:   group_col [2]
##   period group_col     x
##    <dbl> <chr>     <dbl>
## 1      2 a             6
## 2      1 a            10
## 3      1 b             7
## 4      2 b             7
```

Same idea as . . .

```
ex_data |>
```

Wrapping up commonly used chunks of code

We often want to know how many observations (rows) are in groups . . .

```
midwest |>
  count(state, inmetro) |>
  head()
```

```
## # A tibble: 6 x 3
##   state inmetro     n
##   <chr>   <int> <int>
## 1 IL          0    74
## 2 IL          1    28
## 3 IN          0    55
## 4 IN          1    37
## 5 MI          0    58
## 6 MI          1    25
```

How would you get the same output with group_by()?

```
## # A tibble: 6 x 3
##   state inmetro     n
##   <chr>   <int> <int>
## 1 IL          0    74
## 2 IL          1    28
## 3 IN          0    55
## 4 IN          1    37
## 5 MI          0    58
## 6 MI          1    25
```

How does count() work?

```
midwest |>
  group_by(state, inmetro) |>
  summarize(n = n()) |>
  ungroup() |>
  head(5)
```

```
## # A tibble: 5 x 3
##   state inmetro     n
##   <chr>   <int> <int>
## 1 IL          0    74
## 2 IL          1    28
## 3 IN          0    55
## 4 IN          1    37
## 5 MI          0    58
```

add_count() is a useful short cut

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
```

How does add_count() work?

```
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`.

- *Old way*: 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
```

- *New way*: With `group_by`, we 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
```

Try it yourself: group_by()

1. For each state in the `midwest` data, calculate the proportion of counties that are in a metro area (`inmetro`).[2]
2. Add a col to midwest called `pop_state` that equals the state population.
3. Reproduce this table using `count()`.

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

4. For each county, determine how far the poverty rate (`percbelowpoverty`) is from the state average. Pull out data for Cook and Will counties.

```
## # A tibble: 2 x 4
## # Groups:   state [1]
##   county state state_poverty_rate county_diff
##   <chr>  <chr>              <dbl>       <dbl>
## 1 COOK   IL                  13.1        1.12
## 2 WILL   IL                  13.1       -7.04
```

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

Recap: Analysis by group with `dplyr`

We learned the concept **split-apply-combine** and how to use it in tidyverse!

- 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!