Coding Lab: Grouped Data

Ari Anisfeld

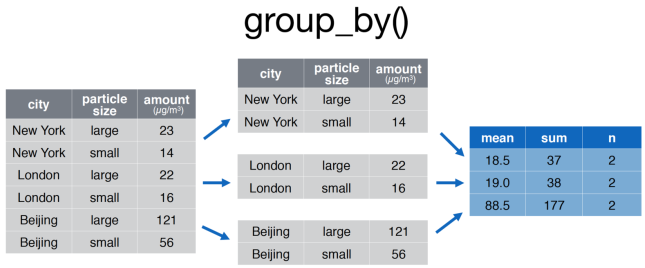
Summer 2020

Grouping data with `dplyr`

Often you want to repeat the same analysis across different subgroups. We can automate that with `group_by()`.

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

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

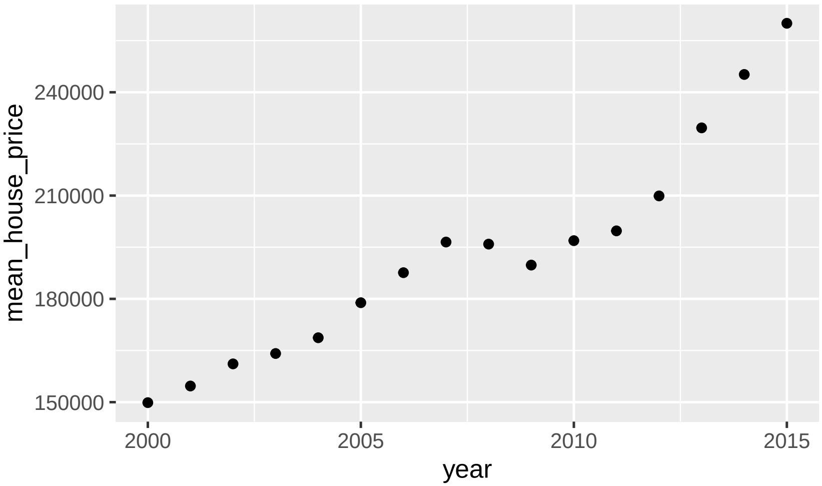
grouped summary with group_by() + summarize()

Use case: You want summary statistics for certain subsets of the data.

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

How have Texas housing prices changed over time?

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



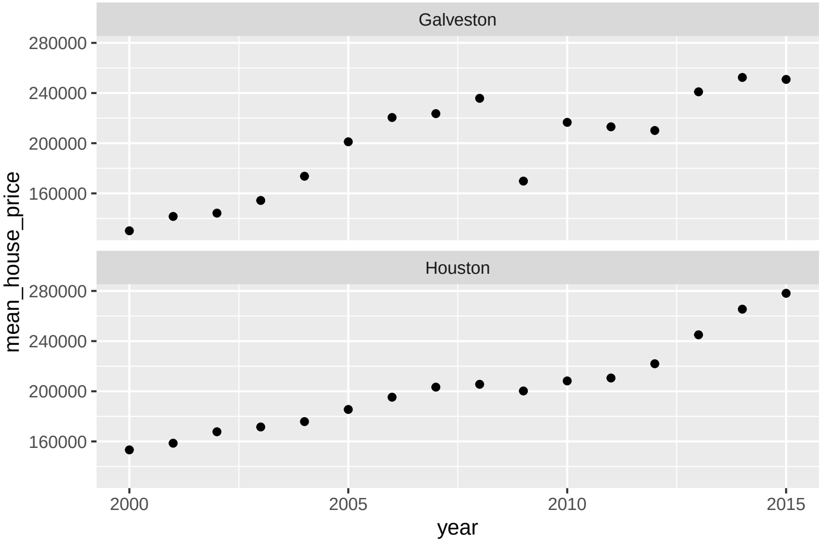
grouped summary with group_by() + summarize()

Use case: You want summary statistics for certain subsets of the data.

```r
texas_housing_data %>%
  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)
```

```
## # A tibble: 736 x 5
## # Groups:   city [46]
##    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.
##  6 Abilene  2005        1977    198855000          100584.
##  7 Abilene  2006        1997    227530000          113936.
##  8 Abilene  2007        2003    232062585          115858.
##  9 Abilene  2008        1651    192520335          116608.
```

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



What does group_by() do?

Let's make a grouped and non-grouped tibble for investigation.

```
a_non_grouped_df <-
  texas_housing_data %>%
  select(city, year)
```

```
a_grouped_df <-
  texas_housing_data %>%
    select(city, year) %>%
    group_by(city, year)
```

What does group_by() do?

```r
a_non_grouped_df %>% glimpse()
```

```
## Observations: 8,602
## Variables: 2
## $ city <chr> "Abilene", "Abilene", "Abilene", "Abilene", "Abilene",
## $ year <int> 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2
```

```r
a_grouped_df %>% glimpse()
```

```
## Observations: 8,602
## Variables: 2
## Groups: city, year [736]
## $ city <chr> "Abilene", "Abilene", "Abilene", "Abilene", "Abilene",
## $ year <int> 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2
```

What does group_by() do?

- Conceptually, `group_by` "tags" rows as belong to a group.
- In practice, R creates a list of row numbers assigned to each group.

As an analyst, you just need to understand the concept. But to see what's going on ...

```r
# Recall, our "groups" are city-year combos
# and there are 12 months of obs per year
a_grouped_df %>% group_rows()
```

```
## <list_of<integer>[736]>
## [[1]]
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
##
## [[2]]
##  [1] 13 14 15 16 17 18 19 20 21 22 23 24
##
## [[3]]
##  [1] 25 26 27 28 29 30 31 32 33 34 35 36
##
## [[4]]
##  [1] 37 38 39 40 41 42 43 44 45 46 47 48
```

Grouping columns have some restrictions

For example, you cannot remove them from the tibble

```
a_grouped_df %>%
  select(-year)
```

```
## Adding missing grouping variables: `year`
```

```
## # A tibble: 8,602 x 2
## # Groups:   city, year [736]
##      year city
##     <int> <chr>
##   1  2000 Abilene
##   2  2000 Abilene
##   3  2000 Abilene
##   4  2000 Abilene
##   5  2000 Abilene
##   6  2000 Abilene
##   7  2000 Abilene
##   8  2000 Abilene
```

To get rid of groups, use ungroup()

```r
a_grouped_df %>%
  ungroup() %>%
  select(-year)
```

```
## # A tibble: 8,602 x 1
##    city
##    <chr>
##  1 Abilene
##  2 Abilene
##  3 Abilene
##  4 Abilene
##  5 Abilene
##  6 Abilene
##  7 Abilene
##  8 Abilene
##  9 Abilene
## 10 Abilene
## # ... with 8,592 more rows
```

grouped mutate: differences

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

```
# I'm going to use this a bunch,
# so I'll store it in memory
july_texas_housing_data <-
  texas_housing_data %>%
    filter(month == 7) %>%
    select(city, year, sales)

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

grouped `mutate`: differences

Use case: You want to work with differences.[1]

```
differenced_data %>% head(5)
```

```
## # A tibble: 5 x 5
## # Groups:   city [1]
##   city     year sales last_year_sales delta_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: 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_housing_data %>%
  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 10 cities for each year, you can

```
july_texas_housing_data %>%
  group_by(year) %>%
  filter(rank(desc(sales)) <= 10) %>%
  arrange(year, sales)
```

```
## # A tibble: 160 x 3
## # Groups:   year [16]
##    city                year sales
##    <chr>              <int> <dbl>
##  1 Montgomery County  2000   463
##  2 Fort Worth         2000   548
##  3 Denton County      2000   638
##  4 NE Tarrant County  2000   686
##  5 Fort Bend          2000   753
##  6 Collin County      2000  1007
##  7 San Antonio        2000  1508
```

count() is a useful short cut

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

```
texas_housing_data %>%
  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(x) is nearly identical to group_by(x) %>% summarize(n = n()) %>% ungroup().

```
texas_housing_data %>%
  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

add_count(x) is nearly identical to group_by(x) %>% mutate(n = n()) %>% ungroup().

```
texas_housing_data %>%
  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(x) is nearly identical to group_by(x) %>% mutate(n = n()) %>% ungroup().

```
texas_housing_data  %>%
  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
```

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