Chapter_4_Aggregating_Data_With_R

Ravi Mummigatti

7/7/2021

Contents

Introduction

In this lesson you will learn about aggregates in R using dplyr.

An aggregate statistic is a way of creating a single number that describes a group of numbers. Common aggregate statistics include mean, median, and standard deviation.

Additionally, you will learn how you can group data into different subsets based on column values. This can help narrow the focus of a summary statistic to a subset of a dataset

We will analyze data from ShoeFly.com, a fictional e-commerce shoe store. The data includes information regarding customer orders as well as the source of page visits to ShoeFly.com's website

Calculating Column Statistics

In this exercise, you will learn how to *combine* all of the values from a column for a single calculation. This can be done with the help of the dplyr function `summarize()`, which returns a new data frame containing the desired calculation.

Some examples of this type of calculation include:

- The data frame `customers` contains the names and ages of all of your customers. You want to find the median age:

```
customers %>%
  select(age)
# c(23, 25, 31, 35, 35, 46, 62)
customers %>%
  summarize(median_age = median(age))
# 35
```

- The data frame `shipments` contains address information for all shipments that you've sent out in the past year. You want to know how many different states you have shipped to.

```
shipments %>%
  select(states)
# c('CA', 'CA', 'CA', 'CA', 'NY', 'NY', 'NJ', 'NJ', 'NJ', 'NJ', 'NJ', 'NJ', 'NJ')
shipments %>%
  summarize(n_distinct_states = n_distinct(states))
# 3
```

- The data frame `inventory` contains a list of types of t-shirts that your company makes. You want to know the standard deviation of the prices of your inventory.

  inventory %>% select(price) # c(31, 23, 30, 27, 30, 22, 27, 22, 39, 27, 36) inventory %>% summarize(sd_price = sd(price)) # 5.465595

The general syntax for these calculations is:

```
df %>%
  summarize(var_name = command(column_name))
```

- `df` is the data frame you are working with
- `summarize` is a dplyr function that reduces multiple values to a single value
- `var_name` is the name you assign to the column that stores the results of the summary function in the returned data frame
- `command` is the summary function that is applied to the column by `summarize()`
- `column_name` is the name of the column of `df` that is being summarized

The following table includes common summary functions that can be given as an argument to `summarize()`:

Command	Description
`mean()`	Average of all values in column
`median()`	Median value of column
`sd()`	Standard deviation of column
`var()`	Variance of column
`min()`	Minimum value in column
`max()`	Maximum value in column
`IQR()`	Interquartile range of column
`n_distinct()`	Number of unique values in column

Command	Description
`sum()`	Sum values of column

Let us load the required libraries for our analysis

```
library(dplyr)
library(readr)
library(tidyverse)
```

Let us load the required data-sets

```
orders <- read_csv("shoefly.csv")
page_visits <- read_csv("page_visits.csv")
```

ShoeFly.com has a new batch of orders stored in the data frame `orders`. Inspect the first `10` rows if the data frame using `head()`.

```
# inspect orders
head(orders , 10)
```

```
## # A tibble: 10 x 8
##       id first_name last_name  email    shoe_type shoe_material shoe_color price
##    <dbl> <chr>      <chr>      <chr>    <chr>     <chr>          <chr>       <dbl>
##  1 41874 Kyle       Peck       KylePec~ ballet f~ faux-leather  black         385
##  2 31349 Elizabeth  Velazquez  EVelazq~ boots     fabric         brown         388
##  3 43416 Keith      Saunders   KS4047@~ sandles   leather        navy          346
##  4 56054 Ryan       Sweeney    RyanSwe~ sandles   fabric         brown         344
##  5 77402 Donna      Blankensh~ DB3807@~ stilettos fabric         brown         289
##  6 97148 Albert     Dillon     Albert.~ wedges    fabric         brown         266
##  7 19998 Judith     Hewitt     JudithH~ stilettos leather        black         395
##  8 83290 Kayla      Hardin     Kayla.H~ stilettos leather        white         241
##  9 77867 Steven     Blankensh~ Steven.~ wedges    leather        navy          266
## 10 54885 Carol      Mclaughlin CM3415@~ ballet f~ faux-leather  brown         440
```

Our finance department wants to know the price of the most expensive pair of shoes purchased. Save your answer to the variable `most_expensive`

```
# maximum price
most_expensive <- orders %>%
  summarize(max_price = max(price))

# view
most_expensive
```

```
## # A tibble: 1 x 1
##   max_price
##       <dbl>
## 1        NA
```

The result for the most expensive pair of shoes is coming back as `NA`. Why is this happening?

If you View the `orders data frame` ,you can see that there is a missing information! It appears that the `price` for row `99` was not in the file, and this is causing your maximum value calculation to return `NA`

We add the argument na.rm = True to the max() function to ignore "Missing / NA Values"

```
# maximum price
orders %>%
  summarize(max_price = max(price , na.rm = TRUE))
```

```
## # A tibble: 1 x 1
##   max_price
##       <dbl>
## 1       493
```

Our fashion department wants to know how many different colors of shoes we are selling. Save your answer to the variable `num_colors`

```
# use n_distinct() on the shoe_color column
orders %>%
    summarize(distinct_colors = n_distinct(shoe_color))
```

```
## # A tibble: 1 x 1
##   distinct_colors
##             <int>
## 1               5
```

Calculating Aggregate Functions

Groupby single column and summarize

When we have a bunch of data, we often want to calculate aggregate statistics (mean, standard deviation, median, percentiles, etc.) over certain subsets of the data.

Suppose we have a grade book with columns `student`, `assignment_name`, and `grade`:

student	assignment_name	grade
Amy	Assignment 1	96
Amy	Assignment 2	87
Bob	Assignment 1	91
Bob	Assignment 2	75
Chris	Assignment 1	83
Chris	Assignment 2	88

We want to get an average grade for each student across all assignments. We can do this using the helpful dplyr function `group_by()`.

For this example, we'd use the following piece of code:

```
grades <- df %>%
  group_by(student) %>%
  summarize(mean_grade = mean(grade))
```

The output might look something like this:

student	mean_grade
Amy	91.5
Bob	83
Chris	85.5

In general, we use the following syntax to calculate aggregates:

```
df %>%
  group_by(column_1) %>%
  summarize(aggregate_name = command(column_2))
```

- `column_1` (`student` in our example) is the column that we want to `group_by()`
- `column_2` (`grade` in our example) is the column that we want to apply `command()`, a summary function, to using `summarize()`
- `aggregate_name` is the name assigned to the calculated aggregate

In addition to the summary functions discussed in the last exercise (`mean()`, `median()`, `sd()`, `var()`, `min()`, `max()`, `IQR()` and `n_distinct()`),

Another helpful summary function, especially for grouped data, is `n()`. `n()` will return the count of the rows within a group, and does not require a column as an argument. To get the count of the rows in each group of students from our example:

```
grades <- df %>%
  group_by(student) %>%
  summarize(count = n())
```

Our Finance department wants to know the price of the most expensive shoe for each `shoe_type` (i.e., the price of the most expensive boot, the price of the most expensive ballet flat, etc.).

Save your answer to the variable `pricey_shoes`, and view it.

```
# group by shoe_type and summarize max (price)
orders %>%
    group_by(shoe_type) %>%
    summarize(max_price = max(price , na.rm = TRUE))
```

```
## # A tibble: 6 x 2
##   shoe_type    max_price
##   <chr>            <dbl>
## 1 ballet flats       481
## 2 boots              478
## 3 clogs              493
## 4 sandles            456
## 5 stilettos          487
## 6 wedges             461
```

The inventory team wants to know how many of each `shoe_type` has been sold so they can forecast inventory for the future.

Save your answer to the variable `shoes_sold`, and view it.

```
# groupby shoe_type then summarize count
orders %>%
    group_by(shoe_type) %>%
    summarise(shoes_sold = n())
```

```
## # A tibble: 6 x 2
##   shoe_type    shoes_sold
##   <chr>             <int>
## 1 ballet flats         15
## 2 boots                19
## 3 clogs                16
## 4 sandles              17
## 5 stilettos            14
## 6 wedges               18
```

Groupby multiple columns and summarize

Sometimes, we want to group by more than one column. We can do this by passing multiple column names as arguments to the `group_by` function.

Imagine that we run a chain of stores and have data about the number of sales at different locations on different days:

location	date	day_of_week	total_sales
West Village	February 1	W	400
West Village	February 2	Th	450
Chelsea	February 1	W	375
Chelsea	February 2	Th	390
...	...	...	...

We suspect that sales are different at different locations on different days of the week. In order to test this hypothesis, we could calculate the average sales for each store on each day of the week across multiple months.

The code would look like this:

```
df %>%
  group_by(location,day_of_week) %>%
  summarize(mean_total_sales = mean(total_sales))
```

And the results might look something like this:

location	day_of_week	mean_total_sales
Chelsea	M	402.50
Chelsea	Tu	422.75
Chelsea	W	452.00

location	day_of_week	mean_total_sales
...	...	...
West Village	M	390
West Village	Tu	400
...	...	...

At ShoeFly.com, our Purchasing team thinks that certain `shoe_type`/`shoe_color` combinations are particularly popular this year (for example, blue ballet flats are all the rage in Paris).

Find the total number of shoes of each `shoe_type`/`shoe_color` combination purchased using `group_by`, `summarize()` and `n()`. Save your result to the variable `shoe_counts`, and view it.

```
# groupby shoe_type , shoe_color and summarize n()
orders %>%
    group_by(shoe_type , shoe_color) %>%
    summarise(shoes_sold = n())
```

```
## `summarise()` has grouped output by 'shoe_type'. You can override using the `.groups` argument.
```

```
## # A tibble: 29 x 3
## # Groups:   shoe_type [6]
##    shoe_type    shoe_color shoes_sold
##    <chr>        <chr>           <int>
##  1 ballet flats black               2
##  2 ballet flats brown               5
##  3 ballet flats red                 3
##  4 ballet flats white               5
##  5 boots        black               3
##  6 boots        brown               5
##  7 boots        navy                6
##  8 boots        red                 2
##  9 boots        white               3
## 10 clogs        black               4
## # ... with 19 more rows
```

The Marketing team wants to better understand the different price levels of the kinds of shoes that have been sold on the website, in particular looking at `shoe_type`/`shoe_material` combinations.

Find the mean price of each `shoe_type`/`shoe_material` combination purchased using `group_by`, `summarize()` and `mean()`. Save your result to the variable `shoe_prices`, and view it.

Don't forget to include `na.rm = TRUE` as an argument in the summary function that you call!

```
# groupby shoe_type , shoe_material , summarize mean price
orders %>%
    group_by(shoe_type , shoe_material) %>%
    summarize(avg_price = mean(price , na.rm = TRUE))
```

```
## `summarise()` has grouped output by 'shoe_type'. You can override using the `.groups` argument.
```

```
## # A tibble: 18 x 3
## # Groups:   shoe_type [6]
```

```
##    shoe_type    shoe_material avg_price
##    <chr>        <chr>             <dbl>
##  1 ballet flats fabric             277
##  2 ballet flats faux-leather       261.
##  3 ballet flats leather            230.
##  4 boots        fabric             279.
##  5 boots        faux-leather       252.
##  6 boots        leather            268.
##  7 clogs        fabric             284.
##  8 clogs        faux-leather       205.
##  9 clogs        leather            377.
## 10 sandles      fabric             279.
## 11 sandles      faux-leather       246.
## 12 sandles      leather            300.
## 13 stilettos    fabric             336.
## 14 stilettos    faux-leather       419
## 15 stilettos    leather            366
## 16 wedges       fabric             254.
## 17 wedges       faux-leather       316
## 18 wedges       leather            237.
```

Combining Grouping with Filter

While `group_by()` is most often used with `summarize()` to calculate summary statistics, it can also be used with the dplyr function `filter()` to filter rows of a data frame based on per-group metrics.

Suppose you work at an educational technology company that offers online courses and collects user data in an `enrollments` data frame:

user_id	course	quiz_score
1234	learn_r	80
1234	learn_python	95
4567	learn_r	90
4567	learn_python	55

You want to identify all the enrollments in difficult courses, which you define as courses with an average `quiz_score` less than `80`. To filter the data frame to just these rows:

```
enrollments %>%
  group_by(course) %>%
  filter(mean(quiz_score) < 80)
```

- `group_by()` groups the data frame by `course` into two groups: `learn-r` and `learn-python`
- `filter()` will keep all the rows of the data frame whose per-group (per-course) average `quiz_score` is less than `80`

Rather than filtering rows by the individual column values, the rows will be filtered by their group value since a summary function is used! The resulting data frame would look like this:

user_id	course	quiz_score
1234	learn_python	95
4567	learn_python	55

- The average `quiz_score` for the `learn-r` course is `85`, so all the rows of `enrollments` with a value of `learn-r` in the `course` column are filtered out.
- The average `quiz_score` for the `learn-python` course is `75`, so all the rows of `enrollments` with a value of `learn-python` in the `course` column remain.

ShoeFly.com wants to gain a better insight into the orders of the most popular `shoe_types`.

Group `orders` by `shoe_type` and filter to only include orders with a `shoe_type` that has been ordered more than `16` times. Save the result to `most_pop_orders`, and view it.

You can include any of the summary functions as part of an argument to `filter()`, including `n()`!

```
# groupby shoe_type
most_popular_orders <- orders %>%
    group_by(shoe_type) %>%
# filter count > 16
    filter(n() > 16)

# view
head(most_popular_orders)
```

```
## # A tibble: 6 x 8
## # Groups:   shoe_type [3]
##      id first_name last_name  email     shoe_type shoe_material shoe_color price
##   <dbl> <chr>      <chr>      <chr>     <chr>     <chr>         <chr>      <dbl>
## 1 31349 Elizabeth  Velazquez  EVelazqu~ boots     fabric        brown        388
## 2 43416 Keith      Saunders   KS4047@g~ sandles   leather       navy         346
## 3 56054 Ryan       Sweeney    RyanSwee~ sandles   fabric        brown        344
## 4 97148 Albert     Dillon     Albert.D~ wedges    fabric        brown        266
## 5 77867 Steven     Blankensh~ Steven.B~ wedges    leather       navy         266
## 6 11967 Maria      Whitfield  Maria.Wh~ wedges    fabric        white        180
```

```
# groupby shoe_type
most_popular_orders <- orders %>%
    group_by(shoe_type) %>%
# filter count > 16
    filter(n() > 16)

# view
head(most_popular_orders)
```

```
## # A tibble: 6 x 8
## # Groups:   shoe_type [3]
##      id first_name last_name  email     shoe_type shoe_material shoe_color price
##   <dbl> <chr>      <chr>      <chr>     <chr>     <chr>         <chr>      <dbl>
## 1 31349 Elizabeth  Velazquez  EVelazqu~ boots     fabric        brown        388
## 2 43416 Keith      Saunders   KS4047@g~ sandles   leather       navy         346
```

```
## 3 56054 Ryan       Sweeney    RyanSwee~ sandles   fabric        brown        344
## 4 97148 Albert     Dillon     Albert.D~ wedges    fabric        brown        266
## 5 77867 Steven     Blankensh~ Steven.B~ wedges    leather       navy         266
## 6 11967 Maria      Whitfield  Maria.Wh~ wedges    fabric        white        180
```

Combining Grouping with Mutate

`group_by()` can also be used with the dplyr function `mutate()` to add columns to a data frame that involve per-group metrics.

Consider the same educational technology company's `enrollments` table from the previous exercise:

user_id	course	quiz_score
1234	learn_r	80
1234	learn_python	95
4567	learn_r	90
4567	learn_python	55

You want to add a new column to the data frame that stores the difference between a row's `quiz_score` and the average `quiz_score` for that row's `course`. To add the column:

```
enrollments %>%
  group_by(course) %>%
  mutate(diff_from_course_mean = quiz_score - mean(quiz_score))
```

- `group_by()` groups the data frame by course into two groups: `learn-r` and `learn-python`
- `mutate()` will add a new column `diff_from_course_mean` which is calculated as the difference between a row's individual `quiz_score` and the `mean(quiz_score)` for that row's group (course)

The resulting data frame would look like this:

user_id	course	quiz_score	diff_from_course_mean
1234	learn_r	80	-5
1234	learn_python	95	20
4567	learn_r	90	5
4567	learn_python	55	-20

- The average `quiz_score` for the `learn-r` course is `85`, so `diff_from_course_mean` is calculated as `quiz_score - 85` for all the rows of `enrollments` with a value of `learn-r` in the `course` column.
- The average `quiz_score` for the `learn-python` course is `75`, so `diff_from_course_mean` is calculated as `quiz_score - 75` for all the rows of `enrollments` with a value of `learn-python` in the `course` column.

You want to be able to tell how expensive each order is compared to the average `price` of orders with the same `shoe_type`.

Group `orders` by `shoe_type` and create a new column named `diff_from_shoe_type_mean` that stores the difference in price between an orders `price` and the average `price` of orders with the same `shoe_type`.

Save the result to `diff_from_mean`, and view it.

Don't forget to include `na.rm = TRUE` as an argument in the summary function you call!

```
# groupby shoe_type
diff_from_mean <- orders %>%
    group_by(shoe_type) %>%
# add column diff_from_shoe_type_mean
    mutate(diff_from_shoe_type_mean = price - mean(price , na.rm = TRUE))

# inspect
head(diff_from_mean)
```

```
## # A tibble: 6 x 9
## # Groups:   shoe_type [5]
##      id first_name last_name  email     shoe_type shoe_material shoe_color price
##   <dbl> <chr>      <chr>      <chr>     <chr>     <chr>         <chr>      <dbl>
## 1 41874 Kyle       Peck       KylePeck~ ballet f~ faux-leather  black        385
## 2 31349 Elizabeth  Velazquez  EVelazqu~ boots     fabric        brown        388
## 3 43416 Keith      Saunders   KS4047@g~ sandles   leather       navy         346
## 4 56054 Ryan       Sweeney    RyanSwee~ sandles   fabric        brown        344
## 5 77402 Donna      Blankensh~ DB3807@g~ stilettos fabric        brown        289
## 6 97148 Albert     Dillon     Albert.D~ wedges    fabric        brown        266
## # ... with 1 more variable: diff_from_shoe_type_mean <dbl>
```

A/B Testing for ShoeFly.com

Our favorite online shoe store, ShoeFly.com is performing an A/B Test. They have two different versions of an ad, which they have placed in emails, as well as in banner ads on Facebook, Twitter, and Google. They want to know how the two ads are performing on each of the different platforms on each day of the week. Help them analyze the data using aggregate measures.

`ad_clicks` contains the following columns:

- `user_id`: unique user id
- `utm_source`: where user saw the ad. **UTM** stands for **U**rchin **T**racking **M**odule
- `day`: the day the ad was seen
- `ad_click_timestamp`: the time the ad was clicked
- `ad_clicked`: boolean indicating if ad was clicked (TRUE or FALSE)
- `experimental_group`: which ad version was shown (A or B)

Analyzing Ad Sources

Inspect the first few rows of `ad_clicks` using `head()`. What variables are stored in the columns of the data frame?

```
# load packages
library(readr)
library(dplyr)
```

```
# load data
ad_clicks = read_csv("add_clicks.csv")
# inspect data
head(ad_clicks)
```

```
## # A tibble: 6 x 5
##   user_id                utm_source day      ad_click_timesta~ experimental_gro~
##   <chr>                  <chr>      <chr>    <time>            <chr>
## 1 008b7c6c-7272-471e-b9~ google     6 - Sat~ 07:18             A
## 2 009abb94-5e14-4b6c-bb~ facebook   7 - Sun~    NA             B
## 3 00f5d532-ed58-4570-b6~ twitter    2 - Tue~    NA             A
## 4 011adc64-0f44-4fd9-a0~ google     2 - Tue~    NA             B
## 5 012137e6-7ae7-4649-af~ facebook   7 - Sun~    NA             B
## 6 013b0072-7b72-40e7-b6~ facebook   1 - Mon~    NA             A
```

We want to know which ad platform is getting the most views.

How many views (i.e., rows of the data frame) came from each `utm_source`?

Group `ad_clicks` by `utm_source` and count the number of rows in each group. Save your result to `views_by_utm`, and view it.

```
# group by utm-source
views_by_utm <- ad_clicks %>%
  group_by(utm_source)

# inspect
views_by_utm
```

```
## # A tibble: 1,654 x 5
## # Groups:   utm_source [4]
##    user_id                utm_source day       ad_click_timesta~ experimental_gr~
##    <chr>                  <chr>      <chr>     <time>            <chr>
##  1 008b7c6c-7272-471e-b~ google     6 - Satu~ 07:18             A
##  2 009abb94-5e14-4b6c-b~ facebook   7 - Sund~    NA             B
##  3 00f5d532-ed58-4570-b~ twitter    2 - Tues~    NA             A
##  4 011adc64-0f44-4fd9-a~ google     2 - Tues~    NA             B
##  5 012137e6-7ae7-4649-a~ facebook   7 - Sund~    NA             B
##  6 013b0072-7b72-40e7-b~ facebook   1 - Mond~    NA             A
##  7 0153d85b-7660-4c39-9~ google     4 - Thur~    NA             A
##  8 01555297-d6e6-49ae-a~ google     3 - Wedn~    NA             A
##  9 018cea61-19ea-4119-8~ email      1 - Mond~ 18:33             A
## 10 01a210c3-fde0-4e6f-8~ email      2 - Tues~ 15:21             B
## # ... with 1,644 more rows
```