05 - Aggregation

ST 597 | Spring 2017
University of Alabama

05-aggregate.pdf

Contents

0.1 Required Packages and Data

```r
library(tidyverse)
library(nycflights13)
data(flights)
```

1 Single table verbs

1.1 dplyr single table verbs

- `filter()`: select rows
- `arrange()`: reorder rows
  - `desc()` to use descending order
- `select()`: select certain columns
  - helper functions: `starts_with()`, `ends_with()`, `matches()`, `contains()`, `?select`
- `mutate()`: modify or create new variables
  - `transmute()`: only return new variables
- `summarise()`: reduce variables to values
  - Most useful when data is grouped

1.2 `summarize()`

The `summarize()` function calculates summary statistics for a column (or multiple columns). It collapses a data frame to a *single row*:

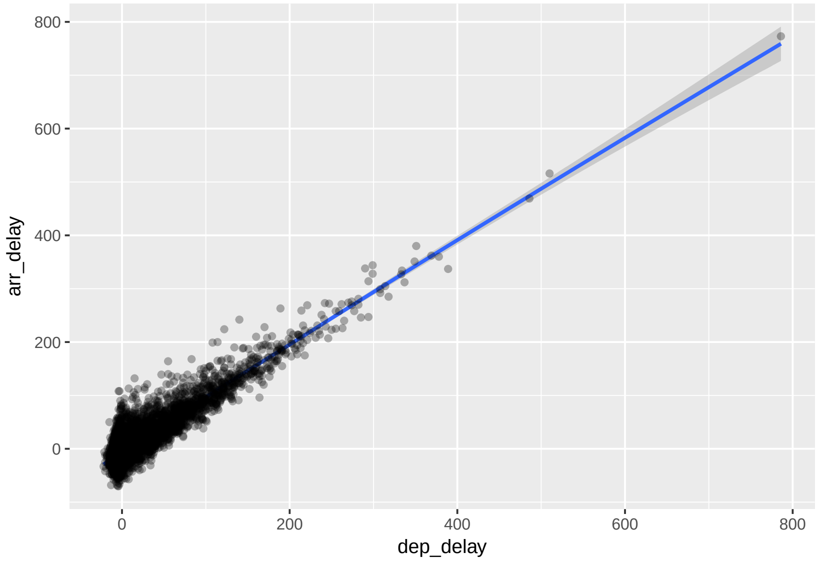
```
summarize(flights, avg.dist = mean(distance))    # mean distance
#> # A tibble: 1 × 1
#>   avg.dist
#>      <dbl>
#> 1     1040
summarize(flights, avg.dist = mean(distance), med.dist = median(distance))
#> # A tibble: 1 × 2
#>   avg.dist med.dist
#>      <dbl>    <dbl>
#> 1     1040      872
summarize(flights,
          n.records = n(),                        # number of records
          n.missing = sum(is.na(arr_delay)),      # number of NA's
          num.delay = sum(arr_delay>0, na.rm=TRUE),   # num of delayed flights
          prop.delay = mean(arr_delay>0, na.rm=TRUE) ) # proportion of delayed flights
#> # A tibble: 1 × 4
#>   n.records n.missing num.delay prop.delay
#>       <int>     <int>     <int>      <dbl>
#> 1    336776      9430    133004      0.406
```

Just to check the numbers, $133004/(336776 - 9430) = 0.406$

It works like this `summarize(<data>, name1 = f(<colname), name2=g(<colname>))` where f, g are some functions (e.g., `mean()`, `median()`, `sd()`).

It can also use functions that take more than one column as input, but most return a single value. E.g.,

```
#- correlation of dep_delay and arr_delay
summarize(flights, delay_cor = cor(dep_delay, arr_delay, use="complete.obs"))
#> # A tibble: 1 × 1
#>   delay_cor
#>       <dbl>
#> 1     0.915
# same as:
# cor(flights$dep_delay, flights$arr_delay, use="complete.obs")
ggplot(sample_n(flights,10000), aes(dep_delay, arr_delay)) +
  geom_smooth() + geom_point(alpha=.3)
#> `geom_smooth()` using method = 'gam'
#> Warning: Removed 281 rows containing non-finite values (stat_smooth).
#> Warning: Removed 281 rows containing missing values (geom_point).
```



1.3 `summary()` is not `summarize()`

The `summary()` function is a base R function that reports some basic summary stats for all columns.

```
summary(flights)                                # base R function
#>       year          month             day           dep_time   
#>  Min.   :2013   Min.   : 1.00   Min.   : 1.0   Min.   :   1  
#>  1st Qu.:2013   1st Qu.: 4.00   1st Qu.: 8.0   1st Qu.: 907  
#>  Median :2013   Median : 7.00   Median :16.0   Median :1401  
#>  Mean   :2013   Mean   : 6.55   Mean   :15.7   Mean   :1349  
#>  3rd Qu.:2013   3rd Qu.:10.00   3rd Qu.:23.0   3rd Qu.:1744  
#>  Max.   :2013   Max.   :12.00   Max.   :31.0   Max.   :2400  
#>                                                NA's   :8255  
#>  sched_dep_time   dep_delay        arr_time    sched_arr_time
#>  Min.   : 106   Min.   : -43   Min.   :   1   Min.   :   1  
#>  1st Qu.: 906   1st Qu.:  -5   1st Qu.:1104   1st Qu.:1124  
#>  Median :1359   Median :  -2   Median :1535   Median :1556  
#>  Mean   :1344   Mean   :  13   Mean   :1502   Mean   :1536  
#>  3rd Qu.:1729   3rd Qu.:  11   3rd Qu.:1940   3rd Qu.:1945  
#>  Max.   :2359   Max.   :1301   Max.   :2400   Max.   :2359  
#>                 NA's   :8255   NA's   :8713                 
#>    arr_delay      carrier              flight       tailnum
#>  Min.   : -86   Length:336776      Min.   :   1   Length:336776
#>  1st Qu.: -17   Class :character   1st Qu.: 553   Class :character
#>  Median :  -5   Mode  :character   Median :1496   Mode  :character
#>  Mean   :   7                      Mean   :1972
#>  3rd Qu.:  14                      3rd Qu.:3465
#>  Max.   :1272                      Max.   :8500
#>  NA's   :9430
#>     origin              dest              air_time       distance
#>  Length:336776      Length:336776      Min.   : 20    Min.   :  17
#>  Class :character   Class :character   1st Qu.: 82    1st Qu.: 502
#>  Mode  :character   Mode  :character   Median :129    Median : 872
#>                                        Mean   :151    Mean   :1040
#>                                        3rd Qu.:192    3rd Qu.:1389
#>                                        Max.   :695    Max.   :4983
#>                                        NA's   :9430
#>       hour          minute       time_hour
#>  Min.   : 1.0   Min.   : 0.0   Min.   :2013-01-01 05:00:00
#>  1st Qu.: 9.0   1st Qu.: 8.0   1st Qu.:2013-04-04 13:00:00
#>  Median :13.0   Median :29.0   Median :2013-07-03 10:00:00
#>  Mean   :13.2   Mean   :26.2   Mean   :2013-07-03 05:02:36
#>  3rd Qu.:17.0   3rd Qu.:44.0   3rd Qu.:2013-10-01 07:00:00
#>  Max.   :23.0   Max.   :59.0   Max.   :2013-12-31 23:00:00
#>
```

The summarize() function applies a function that summarizes each column down to a single number.

```
summarize(flights,
          min=min(arr_delay, na.rm=TRUE),
          Q1 = quantile(arr_delay, 0.25, na.rm=TRUE),
          median=median(arr_delay, na.rm=TRUE),
          mean = mean(arr_delay, na.rm=TRUE),
          Q3 = quantile(arr_delay, 0.75, na.rm=TRUE),
          max = max(arr_delay, na.rm=TRUE),
          count.NA = sum(is.na(arr_delay)))
#> # A tibble: 1 × 7
#>      min    Q1 median  mean    Q3   max count.NA
#>   <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl>    <int>
#> 1   -86   -17     -5   6.9    14  1272     9430
```

2 Group-wise operations

2.1 Split-Apply-Combine

These operations are more powerful when they can be used with grouping variables. Split - Apply - Combine.

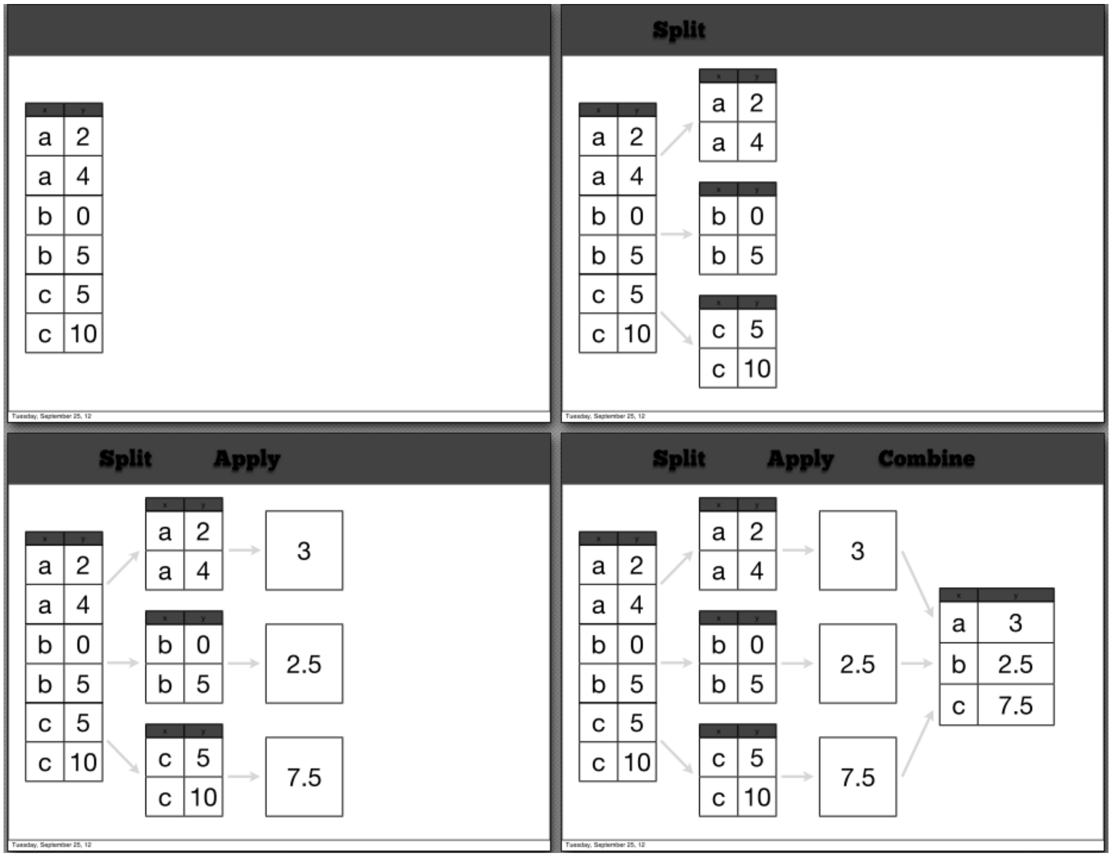


Image from Hadley Wickham UseR tutorial June 2014 http://www.dropbox.com/sh/i8qnluwmuieicxc/AAAgt9tIKoIm7WZKIyK25lh6a

2.2 group_by()

First use the `group_by()` function to group the data (determines how to split), then apply function(s) to each group using the `summarise()` function. Note: grouping should to be applied on discrete variables (categorical, factor, or maybe integer valued columns).

```
#- Get maximum delay by origin
by_origin = group_by(flights, origin)
summarize(by_origin, max.delay = max(arr_delay, na.rm=TRUE))
#> # A tibble: 3 × 2
#>   origin max.delay
#>    <chr>     <dbl>
#> 1    EWR      1109
#> 2    JFK      1272
#> 3    LGA       915


#- Get delay info by origin and destination
by_dest = group_by(flights, origin, dest)
summarize(by_dest,
          max.delay = max(arr_delay, na.rm=TRUE),
          avg.delay = mean(arr_delay, na.rm=TRUE),
          min.delay = min(arr_delay, na.rm=TRUE),
          count = n() )
#> Source: local data frame [224 x 6]
#> Groups: origin [?]
#>
#>    origin  dest max.delay avg.delay min.delay count
#>     <chr> <chr>     <dbl>     <dbl>     <dbl> <int>
#> 1     EWR   ALB       328    14.397       -34   439
#> 2     EWR   ANC        39    -2.500       -47     8
#> 3     EWR   ATL       796    13.233       -39  5022
#> 4     EWR   AUS       349    -0.474       -59   968
#> 5     EWR   AVL       228     8.805       -26   265
#> 6     EWR   BDL       266     7.049       -43   443
#> 7     EWR   BNA       364    12.708       -41  2336
#> 8     EWR   BOS       422     4.784       -47  5327
#> 9     EWR   BQN       208    10.864       -43   297
#> 10    EWR   BTV       306    12.186       -41   931
#> # ... with 214 more rows


#- derived columns: partition air_time into 5 categories (check the NA row too)
by_air_time = group_by(flights, air_time2 = cut(air_time, 5)) # added column
summarize(by_air_time,
          max.delay = max(arr_delay, na.rm=TRUE),
          avg.delay = mean(arr_delay, na.rm=TRUE),
          min.delay = min(arr_delay, na.rm=TRUE),
          count = n() )
#> # A tibble: 6 × 5
#>    air_time2 max.delay avg.delay min.delay  count
#>       <fctr>     <dbl>     <dbl>     <dbl>  <int>
#> 1 (19.3,155]      1127      7.63       -68 215127
#> 2  (155,290]       915      7.72       -75  64954
#> 3  (290,425]      1007      2.45       -86  46558
#> 4  (425,560]        92     48.00        -4      6
#> 5  (560,696]      1272     -1.37       -70    701
#> 6         NA        NA       NaN        NA   9430
```

2.2.1 Your Turn

Your Turn #1 : group_by

Which plane (`tailnum`) has the worst on-time record?

2.3 Counting

We often need to count the number of observations in each group. It is so frequently needed, that `dplyr` has included some shortcuts with `n()`, `tally()` and `count()`

1. use `summarize()` with `n()` (must use *grouped* data)

```
#- tally(.) is same as summarize(., n=n())
summarize(by_origin, n=n())
#> # A tibble: 3 × 2
#>   origin      n
#>    <chr>  <int>
#> 1    EWR 120835
#> 2    JFK 111279
#> 3    LGA 104662
```

2. use `tally()` (must use *grouped* data)

```
tally(by_origin, sort=TRUE)
#> # A tibble: 3 × 2
#>   origin      n
#>    <chr>  <int>
#> 1    EWR 120835
#> 2    JFK 111279
#> 3    LGA 104662
```

3. use `count()` (don't use *grouped* data)

```
#- count(., colname) is same as group_by(., colname) %>% tally()
count(flights, origin)
#> # A tibble: 3 × 2
#>   origin      n
#>    <chr>  <int>
#> 1    EWR 120835
#> 2    JFK 111279
#> 3    LGA 104662
```

2.3.1 Counts over multiple variables

```
#- Count for each route (origin and destination)
count(flights, origin, dest)
#> Source: local data frame [224 x 3]
#> Groups: origin [?]
#>
#>    origin  dest     n
#>     <chr> <chr> <int>
#> 1     EWR   ALB   439
#> 2     EWR   ANC     8
#> 3     EWR   ATL  5022
#> 4     EWR   AUS   968
#> 5     EWR   AVL   265
#> 6     EWR   BDL   443
#> 7     EWR   BNA  2336
#> 8     EWR   BOS  5327
#> 9     EWR   BQN   297
#> 10    EWR   BTV   931
#> # ... with 214 more rows

#- Count for each route by month
count(flights, origin, dest, month)
```

```
#> Source: local data frame [2,313 x 4]
#> Groups: origin, dest [?]
#> 
#>    origin  dest month     n
#>     <chr> <chr> <int> <int>
#> 1     EWR   ALB     1    64
#> 2     EWR   ALB     2    58
#> 3     EWR   ALB     3    57
#> 4     EWR   ALB     4    13
#> 5     EWR   ALB     5    59
#> 6     EWR   ALB     6    34
#> 7     EWR   ALB     7    15
#> 8     EWR   ALB     8    20
#> 9     EWR   ALB     9    20
#> 10    EWR   ALB    10     1
#> # ... with 2,303 more rows
```

2.3.2 Plotting the counts

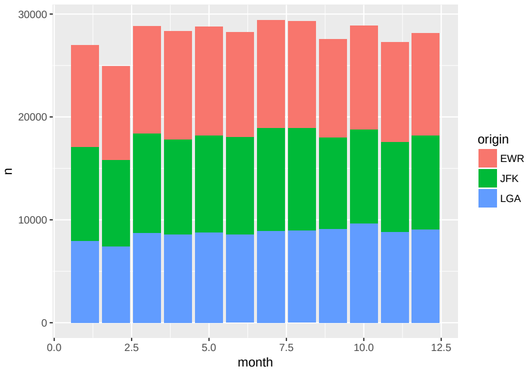
Get the monthly counts

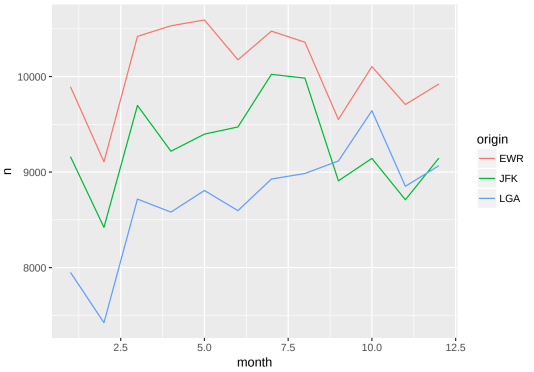
```
(monthly = count(flights, origin, month))
#> Source: local data frame [36 x 3]
#> Groups: origin [?]
#> 
#>    origin month     n
#>     <chr> <int> <int>
#> 1     EWR     1  9893
#> 2     EWR     2  9107
#> 3     EWR     3 10420
#> 4     EWR     4 10531
#> 5     EWR     5 10592
#> 6     EWR     6 10175
#> 7     EWR     7 10475
#> 8     EWR     8 10359
#> 9     EWR     9  9550
#> 10    EWR    10 10104
#> # ... with 26 more rows
```

Notice that `count()` creates the column named `n` (integer).

```
#- (left) Bar Plot
ggplot(monthly) + geom_col(aes(x=month, y=n, fill=origin))
# ggplot(flights) + geom_bar(aes(x=month, fill=origin)) # alternative

#- (right) Line Plot
ggplot(monthly) + geom_line(aes(x=month, y=n, col=origin))
```





2.3.3 Your Turn

Your Turn #2 : Thinking about plots

1. Are the plots better than the table?
2. Which plot do you think is better?
3. How can the plots be improved?

2.3.4 Additional arguments

Check out the help for the `count()` function: `?count` There are two additional arguments:

- `wt` gives a weighted sum (instead of plain count)
- `sort` will sort from largest to smallest

```
#-- total arrival delay by flight number
count(flights, flight, wt=arr_delay, sort=TRUE )
#> # A tibble: 3,844 × 2
#>    flight      n
#>     <int>  <dbl>
#> 1     4131 12989
#> 2      527 11694
#> 3     4333 11433
#> 4      415 11390
#> 5     4224 10204
#> 6     4543 10148
#> 7     1161 10132
#> 8      985 10014
#> 9     2042  9777
#> 10    4204  9478
#> # ... with 3,834 more rows
```

2.3.5 Other types of count

Some useful counts are

- count the number of distinct items with `n_distinct()`

```
summarize(by_origin, n_flights=n(), n_dests=n_distinct(dest))
#> # A tibble: 3 × 3
#>   origin n_flights n_dests
#>    <chr>     <int>   <int>
#> 1    EWR    120835      86
#> 2    JFK    111279      70
#> 3    LGA    104662      68
```

- count the number of non-missing (not-NA) values

```
summarize(by_origin, n_flights=n(), n_missing = sum(is.na(dep_time)),
          n_not_missing=sum(!is.na(dep_time)))
#> # A tibble: 3 × 4
#>   origin n_flights n_missing n_not_missing
#>    <chr>     <int>     <int>         <int>
#> 1    EWR    120835      3239        117596
#> 2    JFK    111279      1863        109416
#> 3    LGA    104662      3153        101509
```

2.3.6 Your Turn

Your Turn #3 : counting

1. How many flights does the plane with the worst on-time record have?
2. Which plane (tailnum) has made the most flights?
3. Which plane (tailnum) has flown the most distance?

2.4 Chaining

Multiple operations can be chained together with the %>% operator (pronounced as *then*). Technically, it performs x %>% f(y) -> f(x, y). This lets you focus on the verbs, or actions you are performing.

```
#- group then summarize then filter then arrange
by_dest = group_by(flights, dest)
delay = summarize(by_dest,
          count = n(),
          avg.delay = mean(arr_delay, na.rm=TRUE))
delay2 = filter(delay, count > 20)
arrange(delay2, desc(avg.delay))
#> # A tibble: 97 × 3
#>     dest count avg.delay
#>    <chr> <int>     <dbl>
#> 1    CAE   116      41.8
#> 2    TUL   315      33.7
#> 3    OKC   346      30.6
#> 4    JAC    25      28.1
#> 5    TYS   631      24.1
#> 6    MSN   572      20.2
#> 7    RIC  2454      20.1
#> 8     CAK   864      19.7
#> 9     DSM   569      19.0
#> 10    GRR   765      18.2
#> # ... with 87 more rows


flights %>%
  group_by(dest) %>%
  summarize( count = n(),
             avg.delay = mean(arr_delay, na.rm=TRUE)) %>%
  filter(count > 20) %>%
  arrange(desc(avg.delay))
#> # A tibble: 97 × 3
#>      dest count avg.delay
#>     <chr> <int>     <dbl>
#> 1     CAE   116      41.8
#> 2     TUL   315      33.7
#> 3     OKC   346      30.6
#> 4     JAC    25      28.1
#> 5     TYS   631      24.1
#> 6     MSN   572      20.2
#> 7     RIC  2454      20.1
#> 8     CAK   864      19.7
#> 9     DSM   569      19.0
#> 10    GRR   765      18.2
#> # ... with 87 more rows
```

Your Turn #4 : Chaining

Use chaining to redo:

1. How many flights does the plane with the worst on-time record have?
2. Which plane (`tailnum`) has made the most flights?
3. Which plane (`tailnum`) has flown the most distance?

2.5 Multiple grouping levels

Notice how each operation strips away a grouping level.

```
#- grouped by: year, month, day
(daily <- group_by(flights, year, month, day))
#> Source: local data frame [336,776 x 19]
#> Groups: year, month, day [365]
#>
#>      year month   day dep_time sched_dep_time dep_delay arr_time
#>     <int> <int> <int>    <int>          <int>     <dbl>    <int>
#> 1    2013     1     1      517            515         2      830
#> 2    2013     1     1      533            529         4      850
#> 3    2013     1     1      542            540         2      923
#> 4    2013     1     1      544            545        -1     1004
#> 5    2013     1     1      554            600        -6      812
#> 6    2013     1     1      554            558        -4      740
#> 7   2013     1     1      555            600       -5      913
#> 8   2013     1     1      557            600       -3      709
#> 9   2013     1     1      557            600       -3      838
#> 10  2013     1     1      558            600       -2      753
#> # ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
#> #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
#> #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
#> #   minute <dbl>, time_hour <dttm>

#- grouped by: year, month
(per_day   <- summarise(daily, flights = n()))
#> Source: local data frame [365 x 4]
#> Groups: year, month [?]
#>
#>     year month   day flights
#>    <int> <int> <int>   <int>
#> 1   2013     1     1     842
#> 2   2013     1     2     943
#> 3   2013     1     3     914
#> 4   2013     1     4     915
#> 5   2013     1     5     720
#> 6   2013     1     6     832
#> 7   2013     1     7     933
#> 8   2013     1     8     899
#> 9   2013     1     9     902
#> 10  2013     1    10     932
#> # ... with 355 more rows

#- grouped by: year
(per_month <- summarise(per_day, flights = sum(flights)))
#> Source: local data frame [12 x 3]
#> Groups: year [?]
#>
#>     year month flights
#>    <int> <int>   <int>
#> 1   2013     1   27004
#> 2   2013     2   24951
#> 3   2013     3   28834
#> 4   2013     4   28330
#> 5   2013     5   28796
#> 6   2013     6   28243
#> 7   2013     7   29425
#> 8   2013     8   29327
#> 9   2013     9   27574
#> 10  2013    10   28889
#> 11  2013    11   27268
#> 12  2013    12   28135

#- grouped by: nothing (i.e., this is not grouped data)
(per_year  <- summarise(per_month, flights = sum(flights)))
#> # A tibble: 1 × 2
#>    year flights
#>   <int>   <int>
#> 1  2013  336776
```

If you want to remove the grouping, use `ungroup()` function.

2.5.1 Your Turn

Your Turn #5 : Multiple groups

1. Find the top 5 routes (`origin`, `dest`), in terms of number of flights.
2. Which route (`origin`, `dest`) is most often delayed by more than 10 minutes? Are infrequent routes a concern? If so, what could we do about it?
3. Find the top 3 destinations (`dest`) for each origin (`origin`).

2.6 Grouped Mutate and Filter

The last exercise (find the top 3 destinations for each origin) requires a *grouped filter*. That is, perform filtering *within* each group separately.

A *grouped mutate* can calculate standardizations per group

```
#- proportion of carrier at each dest
flights %>%
  count(dest, carrier) %>%             # still grouped by dest
  mutate(total=sum(n), p=n/sum(n)) %>% # grouped mutate sum(n) is by group
  arrange(desc(total))                 # arrange by most freq dest
#> Source: local data frame [314 x 5]
#> Groups: dest [105]
#>
#>     dest carrier     n total        p
#>    <chr>   <chr> <int> <int>    <dbl>
#> 1    ORD      9E  1056 17283 6.11e-02
#> 2    ORD      AA  6059 17283 3.51e-01
#> 3    ORD      B6   905 17283 5.24e-02
#> 4    ORD      EV     2 17283 1.16e-04
#> 5    ORD      MQ  2276 17283 1.32e-01
#> 6    ORD      OO     1 17283 5.79e-05
#> 7    ORD      UA  6984 17283 4.04e-01
#> 8    ATL      9E    59 17215 3.43e-03
#> 9    ATL      DL 10571 17215 6.14e-01
#> 10   ATL      EV  1764 17215 1.02e-01
#> # ... with 304 more rows
```

To do a *grouped* filter or mutate, the data frame must be grouped with `group_by()` or `count()`. Functions that work most naturally in grouped mutates and filters are known as *window functions*.

2.7 Window Functions

A *window function* is a variation on an aggregation function. Where an aggregation function, like `sum()` and `mean()`, takes `n` inputs and return a single value, a window function returns `n` values. The output of a window function depends on all its input values, so window functions don't include functions that work element-wise, like + or `round()`. Window functions include variations on aggregate functions, like `cumsum()` and `cummean()`, functions for ranking and ordering, like `rank()`, and functions for taking offsets, like `lead()` and `lag()`. The Z-score can be considered a window function.

More description of some window functions and their use can be found:
- data transform cheatsheet
- Window function vignette

The `rank` family (`?min_rank`) of function can help you not get burned by the ordering. For example, we wanted the top 3 destinations (`dest`) for each origin (`origin`).

```
#- Note: still grouped by origin. So slice operates on group. Must be sorted!
flights %>% count(origin, dest, sort=TRUE) %>% slice(1:3)
#> Source: local data frame [9 x 3]
#> Groups: origin [3]
#>
#>   origin  dest     n
#>    <chr> <chr> <int>
#> 1    EWR   ORD  6100
#> 2    EWR   BOS  5327
#> 3    EWR   SFO  5127
#> 4    JFK   LAX 11262
#> 5    JFK   SFO  8204
#> 6    JFK   BOS  5898
#> 7    LGA   ATL 10263
#> 8    LGA   ORD  8857
#> 9    LGA   CLT  6168

#- incorrect with sort=TRUE
flights %>% count(origin, dest) %>% slice(1:3)
#> Source: local data frame [9 x 3]
#> Groups: origin [3]
#>
#>   origin  dest     n
#>    <chr> <chr> <int>
#> 1    EWR   ALB   439
#> 2    EWR   ANC     8
#> 3    EWR   ATL  5022
#> 4    JFK   ABQ   254
#> 5    JFK   ACK   265
#> 6    JFK   ATL  1930
#> 7    LGA   ATL 10263
#> 8    LGA   AVL    10
#> 9    LGA   BGR   375

#- always correct with rank:
flights %>% count(origin, dest) %>% filter(min_rank(-n)<=3) %>%
  arrange(origin, desc(n))
#> Source: local data frame [9 x 3]
#> Groups: origin [3]
#>
#>   origin  dest     n
#>    <chr> <chr> <int>
#> 1    EWR   ORD  6100
#> 2    EWR   BOS  5327
#> 3    EWR   SFO  5127
#> 4    JFK   LAX 11262
#> 5    JFK   SFO  8204
#> 6    JFK   BOS  5898
#> 7    LGA   ATL 10263
#> 8    LGA   ORD  8857
#> 9    LGA   CLT  6168
```