Introduction to R

Presented by:

BROWN
School of Public Health

BROWN
School of Public Health

Intro to R Programming for Biostatistics

Day 2 - Arranging and Summarizing Data in R

Adam J Sullivan

Arranging the Data

Arranging the Data

- We also have need to make sure the data is ordered in a certain manner. This can be easily done in R with the `arrange()` function.
- Again we can do this in base R but this is not always a clear path.

Arranging the Data Example

- Let's say that we wish to look at only carriers and departure delay and we wish to order departure delays from what smallest to largest.
- In base R we would have to run the following command:

```
flights[order(flights$dep_delay), c("carrier", "dep_delay")]
```

Enter the `arrange()` Function

We could do this in an easy manner using the `arrange()` function:

```
arrange(.data, ...)
```

Where

- `.data` is a data frame of interest.
- `...` are the variables you wish to sort by.

Arranging the Data Example Continued

```
flights %>%
    select(carrier, dep_delay) %>%
    arrange(dep_delay)
```

Arranging the Data Example Continued

```
## # A tibble: 336,776 × 2
##    carrier dep_delay
##      <chr>     <dbl>
## 1       B6       -43
## 2       DL       -33
## 3       EV       -32
## 4       DL       -30
## 5       F9       -27
## 6       MQ       -26
## 7       EV       -25
## 8       MQ       -25
## 9       9E       -24
## 10      B6       -24
## # ... with 336,766 more rows
```

Arranging the Data Example Continued

- With `arrange()` we first use `select()` to pick the only columns that we want and then we arrange by the `dep_delay`.
- If we had wished to order them in a descending manner we could have simply used the `desc()` function:

```
flights %>%
    select(carrier, dep_delay) %>%
    arrange(desc(dep_delay))
```

More Complex Arrange

- Lets consider that we wish to look at the top 3 departure delays for each day.
- Then we wish to order them from largest to smallest departure delay.
- We then need to do the following:
  1. Group by month and Day
  2. Pick the top 3 departure delays
  3. order them largest to smallest

--- .class #id

More Complex Arrange Continued

```
flights %>%
  group_by(month, day) %>%
  top_n(3, dep_delay) %>%
  arrange(desc(dep_delay))
```

Where

- `group_by()` is a way to group data. This way we perform operations on a group. So top 3 delays are by a group of day and month.
- `top_n()`takes a tibble and returns a specific number of rows based on a chosen value.

More Complex Arrange Continued

```
## Source: local data frame [1,108 x 19]
## Groups: 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     9      641            900      1301     1242
## 2   2013     6    15     1432           1935      1137     1607
## 3   2013     1    10     1121           1635      1126     1239
## 4   2013     9    20     1139           1845      1014     1457
## 5   2013     7    22      845           1600      1005     1044
## 6   2013     4    10     1100           1900       960     1342
## 7   2013     3    17     2321            810       911      135
## 8   2013     6    27      959           1900       899     1236
## 9   2013     7    22     2257            759       898      121
## 10  2013    12     5      756           1700       896     1058
## # ... with 1,098 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>
```

On Your Own: RStudio Practice

- Perform the following operations:
    - Group by month and day.
    - use `sample_n()` to pick 1 observation per day.
    - Arrange by longest to smallest departure delay.

--- .class #id

On Your Own: RStudio Practice

Your answer ***may*** look like:

```
flights %>%
  group_by(month,day) %>%
  sample_m(1) %>%
  arrange(desc(dep_delay))
```

Summarizing Data

Summarizing Data

- As you have seen in your own work, being able to summarize information is crucial.
- We need to be able to take out data and summarize it as well.
- We will consider doing this using the `summarise()` function.

Summarizing Data

- Like in the rest of these lessons, let's consider what happens when we try to to do this in base R. We will:
  1. Create a table grouped by `dest`.
  2. Summarize each group by taking mean of `arr_delay`.

```
head(with(flights, tapply(arr_delay, dest, mean, na.rm=TRUE)))
head(aggregate(arr_delay ~ dest, flights, mean))
```

Enter `summarise()` Function

- The `summarise()` function is:

```
summarise(.data, ...)
```

- where
    - `.data` is the tibble of interest.
    - `...` is a list of name paired summary functions
    - Such as:
        - `mean()`
        - `median`
        - `var()`
        - `sd()`
        - `min()`
        - `max()
        -

Summarizing Data Example

```
flights %>%
    group_by(dest) %>%
    summarise(avg_delay = mean(arr_delay, na.rm=TRUE))
```

Summarizing Data Example

- Consider the logic here:
  1. Group flights by destination
  2. Find the average delay of the groups and call it `avg_delay`.
- This is much easier to understand than the Base R code.

```
## # A tibble: 105 × 2
##     dest avg_delay
##    <chr>     <dbl>
## 1    ABQ  4.381890
## 2    ACK  4.852273
## 3    ALB 14.397129
## 4    ANC -2.500000
## 5    ATL 11.300113
## 6    AUS  6.019909
## 7    AVL  8.003831
## 8    BDL  7.048544
## 9    BGR  8.027933
## 10   BHM 16.877323
## # ... with 95 more rows
```

Another Example

- Lets say that we would like to have more than just the averages but we wish to have the minimum and the maximum departure delays by carrier:

```
flights %>%
    group_by(carrier) %>%
    summarise_each(funs(min(., na.rm=TRUE), max(., na.rm=TRUE)), matches("delay"))
```

Another Example

```
## # A tibble: 16 × 5
##    carrier dep_delay_min arr_delay_min dep_delay_max arr_delay_max
##      <chr>         <dbl>         <dbl>         <dbl>         <dbl>
## 1       9E           -24           -68           747           744
## 2       AA           -24           -75          1014          1007
## 3       AS           -21           -74           225           198
## 4       B6           -43           -71           502           497
## 5       DL           -33           -71           960           931
## 6       EV           -32           -62           548           577
## 7       F9           -27           -47           853           834
## 8       FL           -22           -44           602           572
## 9       HA           -16           -70          1301          1272
## 10      MQ           -26           -53          1137          1127
## 11      OO           -14           -26           154           157
## 12      UA           -20           -75           483           455
## 13      US           -19           -70           500           492
## 14      VX           -20           -86           653           676
## 15      WN           -13           -58           471           453
## 16      YV           -16           -46           387           381
```

On Your Own: RStudio Practice

- The following is a new function:
  - Helper function `n()` counts the number of rows in a group
- Then for each day:
  - count total flights
  - Sort in descending order.

On Your Own: RStudio Practice

Your answer should look like:

```
## Source: local data frame [365 x 3]
## Groups: month [12]
##
##    month   day flight_count
##    <int> <int>        <int>
## 1     11    27         1014
## 2      7    11         1006
## 3      7     8         1004
## 4      7    10         1004
## 5     12     2         1004
## 6      7    18         1003
## 7      7    25         1003
## 8      7    12         1002
## 9      7     9         1001
## 10     7    17         1001
## # ... with 355 more rows
```

We could also have used what is called the `tally()` function:

```
flights %>%
    group_by(month, day) %>%
    tally(sort = TRUE)
```

Adding New Variables

Adding New Variables

- There is usually no way around needing a new variable in your data.
- For example, most medical studies have height and weight in them, however many times what a researcher is interested in using is Body Mass Index (BMI).
- We would need to add BMI in.

Adding New Variables

- Using the `tidyverse` we can add new variables in multiple ways

```
- mutate()

- transmute()
```

Adding New Variables

With `mutate()` we have

```
mutate(.data, ...)
```

where

- `.data` is your tibble of interest.
- `...` is the name paired with an expression

Adding New Variables

Then with `transmute()` we have:

```
transmute(.data, ...)
```

where

- `.data` is your tibble of interest.
- `...` is the name paired with an expression

Differences Between `mutate()` and `transmute()`

- There is only one major difference between `mutate()` and `transmutate` and that is what it keeps in your data.
    - `mutate()`
    - creates a new variable
    - It keeps all existing variables
    - `transmute()`
    - creates a new variable.
    - It only keeps the new variables

Example

- Let's say we wish to have a variable called speed. We want to basically do:

$$\text{speed} = \frac{\text{distance}}{\text{time}} * 60$$

We can first do this with `mutate()`:

```
flights %>%
  select(flight, distance, air_time) %>%
  mutate(speed = distance/air_time*60)
```

Example

```
## # A tibble: 336,776 × 4
##    flight distance air_time    speed
##     <int>    <dbl>    <dbl>    <dbl>
## 1    1545     1400      227 370.0441
## 2    1714     1416      227 374.2731
## 3    1141     1089      160 408.3750
## 4     725     1576      183 516.7213
## 5     461      762      116 394.1379
## 6    1696      719      150 287.6000
## 7     507     1065      158 404.4304
## 8    5708      229       53 259.2453
## 9      79      944      140 404.5714
## 10    301      733      138 318.6957
## # ... with 336,766 more rows
```

Example

```
flights %>%
  select(flight, distance, air_time) %>%
  transmute(speed = distance/air_time*60)
```

Example

```
flights %>%
  select(flight, distance, air_time) %>%
  transmute(speed = distance/air_time*60)
```

#Further Summaries

Further Summaries

- We have so far discussed how one could find the basic number summaries:
    - mean
    - median
    - standard deviation
    - variance
    - minimum
    - maximum
- However there are many more operations that you may wish to do for summarizing data.
- In fact many of the following examples are excellent choices for working with categorical data which does not always make sense to do the above summaries for.

Further Summaries

- We will consider:
  1. Grouping and Counting
  2. Grouping, Counting and Sorting
  3. Other Groupings
  4. Counting Groups

Grouping and Counting

- We have seen the functions `tally()` and `count()`.
- Both of these can be used for grouping and counting.
- They also are very concise in how they are called.

Grouping and Counting

- For example if we wished to know how many flights there were by month, we would use `tally()` in this manner:

```
flights %>%
  group_by(month) %>%
  tally()
```

Grouping and Counting

- Where as we could do the same thing with `count()`

```
flights %>%
  count(month)
```

*Notice: `count()` allowed for month to be called inside of it, removing the need for the `group_by()` function.

Grouping, counting and sorting.

- Both `tally()` and `count()` have an argument called `sort()`.
- This allows you to go one step further and group by, count and sort at the same time.
- For `tally()` this would be:

```
flights %>% group_by(month) %>% tally(sort=TRUE)
```

```
## # A tibble: 12 × 2
##    month     n
##    <int> <int>
## 1      7 29425
## 2      8 29327
## 3     10 28889
## 4      3 28834
## 5      5 28796
## 6      4 28330
## 7      6 28243
## 8     12 28135
## 9      9 27574
## 10    11 27268
## 11     1 27004
## 12     2 24951
```

- Then for `count()` we would have:

```
flights %>% count_(month, sort=TRUE)
```

- Then for `count()` we would have:

```
## Error in as.lazy_dots(.dots): object 'month' not found
```

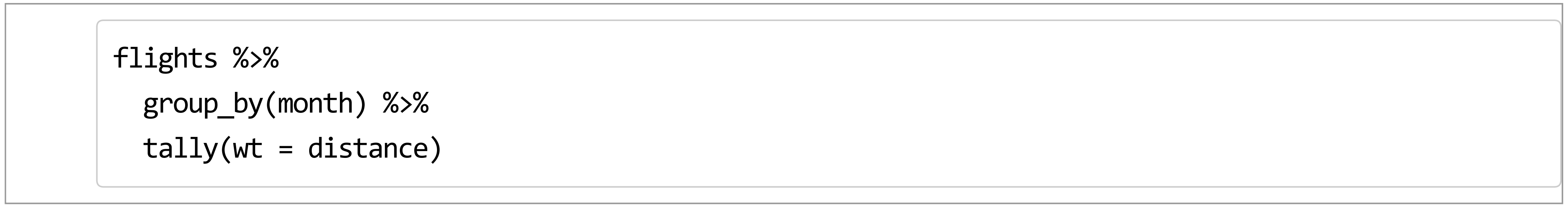
Grouping with other functions

- We can also sum over other values rather than just counting the rows like the above examples.
- For example let us say we were interested in knowing the total distance for planes in a given month.
- We could do this with the `summarise()` function, `tally()` function or the `count()` function:

```
flights %>%
  group_by(month) %>%
  summarise(dist = sum(distance))
```

Grouping with other functions

- We take flights then group by month and then create a new variable called distance, where we sum the distance.
- For `tally()` we could do:



```
flights %>%
  group_by(month) %>%
  tally(wt = distance)
```

Note: in `tally()` the `wt` stands for weight and allows you to weight the sum based on the distance.

Grouping with other functions

- With the `count()` function we also use `wt`:

```
flights %>% count(month, wt = distance)
```

```
## # A tibble: 12 × 2
##    month        n
##    <int>    <dbl>
## 1      1 27188805
## 2      2 24975509
## 3      3 29179636
## 4      4 29427294
## 5      5 29974128
## 6      6 29856388
## 7      7 31149199
## 8      8 31149334
## 9      9 28711426
## 10    10 30012086
## 11    11 28639718
## 12    12 29954084
```

##Counting Groups

- We may want to know how large our groups are. To do this we can use the following functions:

    - `group_size()` is a function that returns counts of group.

    - `n_groups()` returns the number of groups

##Counting Groups

- So if wanted to count the number of flights by month, we could group by month and find the groups size using `group_size()`:

```
flights %>%
  group_by(month) %>%
  group_size()
```

##Counting Groups

```
##  [1] 27004 24951 28834 28330 28796 28243 29425 29327 27574 28889 27268
## [12] 28135
```

##Counting Groups

- If we just wished to know how many months were represented in our data we could use the `n_groups()` function:

```
flights %>%
  group_by(month) %>%
  n_groups()
```

##Counting Groups

```
## [1] 12
```