

**WZB**

Wissenschaftszentrum Berlin  
für Sozialforschung

# R Tutorial at the WZB

## 6 - Recap / Transforming data with R II

Markus Konrad

November 29, 2018

# Today's schedule

1. Review of last weeks' tasks and recap
2. Data aggregation and summarization

# Review of last weeks' tasks and recap

# Solutions and review of previous tasks

Solutions for all previous tasks are online at [https://wzbsocialsciencecenter.github.io/wzb\\_r\\_tutorial/](https://wzbsocialsciencecenter.github.io/wzb_r_tutorial/)

Any problems, questions?

# Recap

I prepared a small quiz for you...

# Transforming data with R II

# What we've learned so far

We already got to know several dplyr "verbs" for transforming data:

- `filter()` for filtering (subsetting) rows (observations) according to some criteria
- `distinct()` for selecting only unique rows
- `arrange()` for ordering rows
- `select()` for selecting only certain columns (variables)
- `rename()` for giving new names to columns
- `mutate()` and `transmute()` for adding new columns

Today: Combining several verbs in one step and learning new verbs for **grouping and summarizing data**.

# Combining several dplyr verbs

Applying several steps is cumbersome this way:

```
air_june <- filter(airquality, Month == 6)
air_june <- select(air_june, -Month)  # we don't need Month any more
head(air_june)
```

```
##   Ozone Solar.R Wind Temp Day
## 1    NA     286  8.6   78   1
## 2    NA     287  9.7   74   2
## 3    NA     242 16.1   67   3
## 4    NA     186  9.2   84   4
## 5    NA     220  8.6   85   5
## 6    NA     264 14.3   79   6
```

You can always nest functions:

```
air_june <- select(filter(airquality, Month == 6), -Month)
```

→ makes code harder to read (you have to read "inside-out")



# Combining several dplyr verbs

A common approach to chain several data transformation steps is to use the **pipe operator**:

```
step_one() %>% step_two() %>% ... %>% last_step()
```

→ the output of one function is passed as input to the next function

```
air_june <- filter(airquality, Month == 6) %>% select(-Month)
```

Notice how **select()** has only one parameter, since it implicitly operates on the output of **filter()**.

For long complex data transformations, each step is usually written on a separate line:

```
air_june <- airquality %>%  
  filter(Month == 6) %>%  
  select(-Month) %>%  
  arrange(desc(Wind))
```

# Aggregates with `group_by()` and `summarise()`

`summarise()` can be used to create summary statistics on aggregate data:

```
summarise(airquality, mean_ozone = mean(Ozone, na.rm = TRUE))
```

```
##    mean_ozone  
## 1    42.12931
```

→ summary of the whole data frame

# Aggregates with `group_by()` and `summarise()`

It is more useful in combination with `group_by()`, which forms groups based on the variables you pass:

```
group_by(airquality, Month) %>% summarise(mean_ozone = mean(Ozone, na.rm = T
```

```
## # A tibble: 5 x 2
##   Month mean_ozone
##   <int>     <dbl>
## 1     5      23.6
## 2     6      29.4
## 3     7      59.1
## 4     8      60.0
## 5     9      31.4
```

→ summary per group, i.e. per month

# Aggregates with `group_by()` and `summarise()`

You can pass several aggregate values as arguments:

```
group_by(airquality, Month) %>%
  summarise(mean_ozone = mean(Ozone, na.rm = TRUE),
            sd_ozone = sd(Ozone, na.rm = TRUE),
            n = n(),                                     # number of obs. per group
            n_nonNA = sum(!is.na(Ozone)))               # number of non-NA Ozone obs.
```

```
## # A tibble: 5 x 5
##   Month mean_ozone sd_ozone      n n_nonNA
##   <int>      <dbl>   <dbl> <int>   <int>
## 1     5      23.6    22.2    31     26
## 2     6      29.4    18.2    30      9
## 3     7      59.1    31.6    31     26
## 4     8      60.0    39.7    31     26
## 5     9      31.4    24.1    30     29
```

- any function passed to `summarize` will operate on each group's observations
- `n()` counts the number of observations in each group

# Some more complex examples with the **flights** data

We'll use a subset `fl_sub` for some more grouping and summarizing examples.

```
library(nycflights13)
```

```
fl_sub <- select(flights, origin, dest, distance, arr_delay)  
head(fl_sub)
```

```
## # A tibble: 6 x 4  
##   origin dest  distance arr_delay  
##   <chr>  <chr>    <dbl>    <dbl>  
## 1 EWR    IAH      1400      11  
## 2 LGA    IAH      1416      20  
## 3 JFK    MIA      1089      33  
## 4 JFK    BQN      1576     -18  
## 5 LGA    ATL       762     -25  
## 6 EWR    ORD       719      12
```

- `origin` and `dest` are the origin and destination airports
- `distance` is the flight distance in miles
- `arr_delay` is the arrival delay in minutes

# Relationship between the distance and average delay

We want to find out what's the **relationship between distance and average delay per destination airport**. We want to **exclude small destination airports ( $\leq 20$  connections) and outlier Honolulu**. Which data transformation steps are necessary?

1. Group flights by destination with `group_by()`.
2. For each group, compute average distance, average delay and number of flights with `summarise()`.
3. Exclude observations according to mentioned criteria with `filter()`.

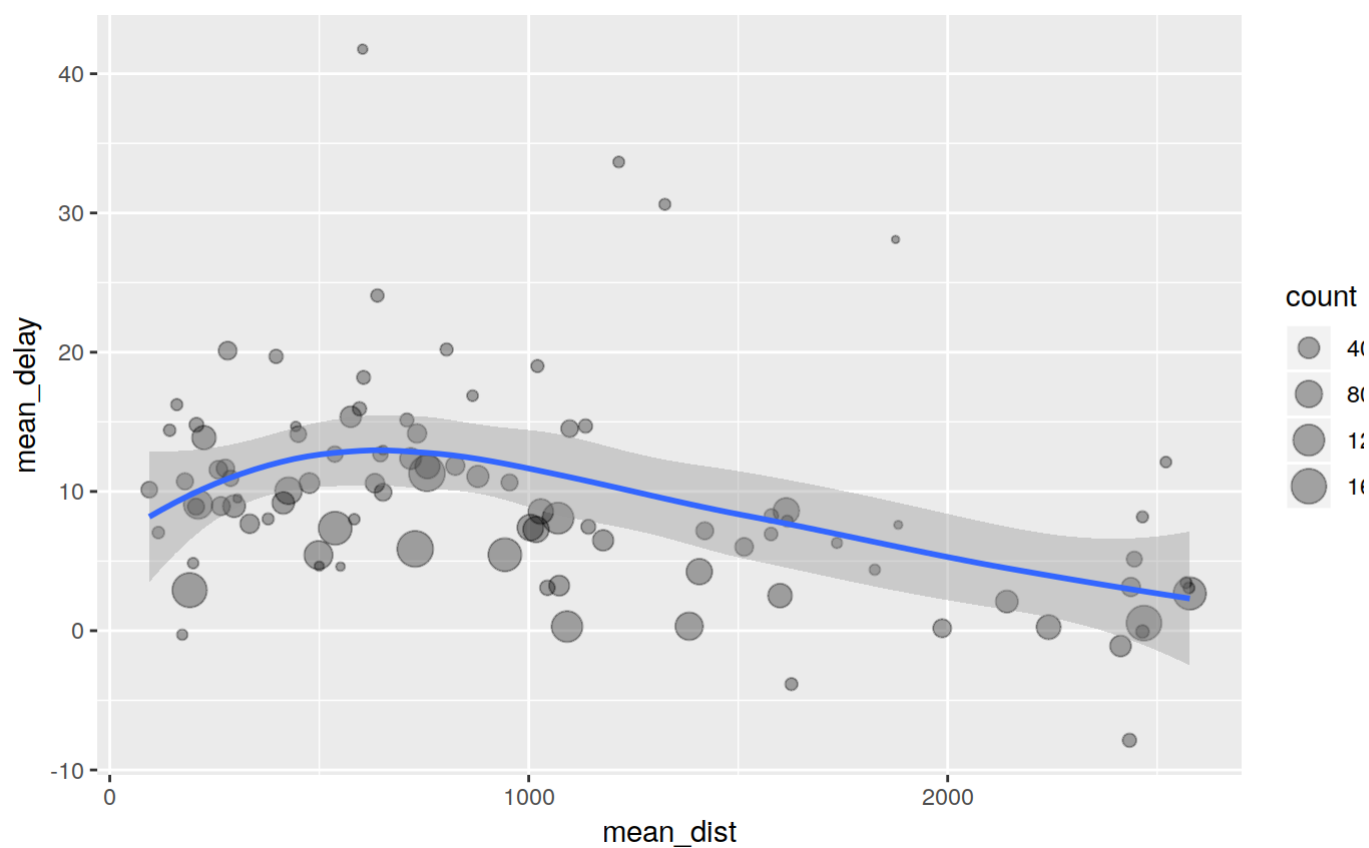
# Relationship between the distance and average delay

We want to find out what's the relationship between distance and average delay per destination airport. We want to exclude small destination airports ( $\leq 20$  connections) and outlier Honolulu.

```
delays <- fl_sub %>%
  group_by(dest) %>%      # step 1
  summarise(count = n(),  # step 2
            mean_dist = mean(distance, na.rm = TRUE),
            mean_delay = mean(arr_delay, na.rm = TRUE)) %>%
  filter(count > 20, dest != "HNL") # step 3
head(delays)
```

```
## # A tibble: 6 x 4
##   dest count mean_dist mean_delay
##   <chr> <int>    <dbl>    <dbl>
## 1 ABQ     254    1826      4.38
## 2 ACK     265     199      4.85
## 3 ALB     439     143     14.4
## 4 ATL   17215     757     11.3
## 5 AUS    2439    1514      6.02
## 6 AVL     275     584      8.00
```

# Relationship between the distance and average delay





# Grouping by multiple variables

We want to know what the **most popular flight connections** are, and also which connections have the **lowest or highest average delays**. We will **exclude rarely used connections with less than 100 flights**. Which data transformation steps are necessary?

1. Group by **origin** and **dest** to form groups of connections.
2. Summarise by counting the number of observations per connection and computing the mean delay.
3. Exclude observations according to mentioned criteria with **filter()**.

# Grouping by multiple variables

We want to know what the **most popular flight connections** are, and also which connections have the **lowest or highest average delays**. We will **exclude rarely used connections with less than 100 flights**. Which data transformation steps are necessary?

```
connections <- fl_sub %>%
  group_by(origin, dest) %>%      # step 1
  summarise(n = n(),              # step 2
            mean_delay = mean(arr_delay, na.rm = TRUE)) %>%
  filter(n >= 100)                # step 3
head(connections)
```

```
## # A tibble: 6 x 4
## # Groups:   origin [1]
##   origin dest      n mean_delay
##   <chr>  <chr> <int>      <dbl>
## 1 EWR    ALB     439      14.4
## 2 EWR    ATL    5022      13.2
## 3 EWR    AUS     968     -0.474
## 4 EWR    AVL     265       8.80
## 5 EWR    BDL     443       7.05
## 6 EWR    BNA    2336      12.7
```

# Grouping by multiple variables

Now we can obtain the top three connections by using `arrange()` and `head()`:

```
connections %>% arrange(desc(n)) %>% head(3)
```

```
## # A tibble: 3 x 4
## # Groups:   origin [2]
##   origin dest      n mean_delay
##   <chr>   <chr> <int>      <dbl>
## 1 JFK     LAX    11262    -0.481
## 2 LGA     ATL    10263     11.3
## 3 LGA     ORD     8857      1.83
```

Or the top three connections with the least delay time:

```
connections %>% arrange(mean_delay) %>% head(3)
```

```
## # A tibble: 3 x 4
## # Groups:   origin [2]
##   origin dest      n mean_delay
##   <chr>   <chr> <int>      <dbl>
## 1 EWR     SNA     825    -7.87
## 2 JFK     HNL     342    -6.92
## 3 JFK     STT     333    -6.37
```

# Grouping by multiple variables

Or the top three connections with the most delay time:

```
connections %>% arrange(desc(mean_delay)) %>% head(3)
```

```
## # A tibble: 3 x 4
## # Groups:   origin [1]
##   origin dest      n mean_delay
##   <chr>  <chr> <int>      <dbl>
## 1 EWR    CAE    104      44.6
## 2 EWR    TYS    323      41.2
## 3 EWR    TUL    315      33.7
```

# Ungrouping

Once you group a data frame and assign it to an object, the grouping information is retained. You can see this in the additional information that is printed above the data (Groups: ...):

```
head(connections)
```

```
## # A tibble: 6 x 4
## # Groups:   origin [1]
##   origin dest      n mean_delay
##   <chr>  <chr> <int>      <dbl>
## 1 EWR    ALB     439      14.4
## 2 EWR    ATL    5022      13.2
## 3 EWR    AUS     968     -0.474
## 4 EWR    AVL     265       8.80
## 5 EWR    BDL     443       7.05
## 6 EWR    BNA    2336      12.7
```

# Ungrouping

As long as a data frame is grouped, `summarise()` operates on the groups and not on the whole data frame:

```
connections %>% summarise(median_n_conn = median(n))
```

```
## # A tibble: 3 x 2
##   origin median_n_conn
##   <chr>         <dbl>
## 1 EWR           1080.
## 2 JFK           1130.
## 3 LGA            761
```

You can remove the grouping information via `ungroup()`. Now `summarise()` operates on the whole data frame:

```
connections %>% ungroup() %>% summarise(median_n_conn = median(n))
```

```
## # A tibble: 1 x 1
##   median_n_conn
##   <int>
## 1         997
```

# Tasks

# Tasks

See dedicated tasks sheet on the [tutorial website](#).