Chapter 6: Some concepts of `dplyr` package *

Contents

1 Data frame: `flights`

The data frame `flights` contains on-time data for all 336 776 flights that departed from New York City in 2013.

```
library(nycflights13)
library(tidyverse)
library(dplyr)
```

```
head(flights)
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     1      517            515         2      830            819
## 2  2013     1     1      533            529         4      850            830
## 3  2013     1     1      542            540         2      923            850
## 4  2013     1     1      544            545        -1     1004           1022
## 5  2013     1     1      554            600        -6      812            837
## 6  2013     1     1      554            558        -4      740            728
## # ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
```

*Used reference: R for Data Science, Garret Grolemund and Hadley Wickham (see http://r4ds.had.co.nz/transform.html)

```
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

This data frame is a **tibble**. It is a data frame with a special format in order to work better with functions of the `dplyr` package.

Abbreviation at column head	Data type
`int`	Integer
`dbl`	Double (real)
`chr`	Character
`dttm`	Date-time
`lgl`	Logical (`T` or `F`)
`fctr`	Factor
`date`	Dates

2 Basic functions in `dplyr` package

Action	Function
Select observations	`filter()`
Select variables	`select()`
Sort data frames	`arrange()`
Create new variables	`mutate()`, `transmute`
Aggregate	`summarise()`
Grouping	`group_by()`
Merging	`inner_join()`, `left_join()`, `right_join()`, `full_join()`

Always same structure:

```
function(data frame, arguments)
```

2.1 Select observations: `filter()`

The `filter()` function is used to subset a data frame, retaining all rows that satisfy your condition. To be retained, the row must produce a value of `TRUE` for all conditions.

- Select all flights of October 1st

```
oct <- filter(flights, month == 10, day ==1)
head(oct)
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013    10     1      447            500       -13      614            648
## 2  2013    10     1      522            517         5      735            757
## 3  2013    10     1      536            545        -9      809            855
## 4  2013    10     1      539            545        -6      801            827
## 5  2013    10     1      539            545        -6      917            933
## 6  2013    10     1      544            550        -6      912            932
## # ... with 11 more variables: 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>
```

- Select all flights of October and all flights of 1st day of the month

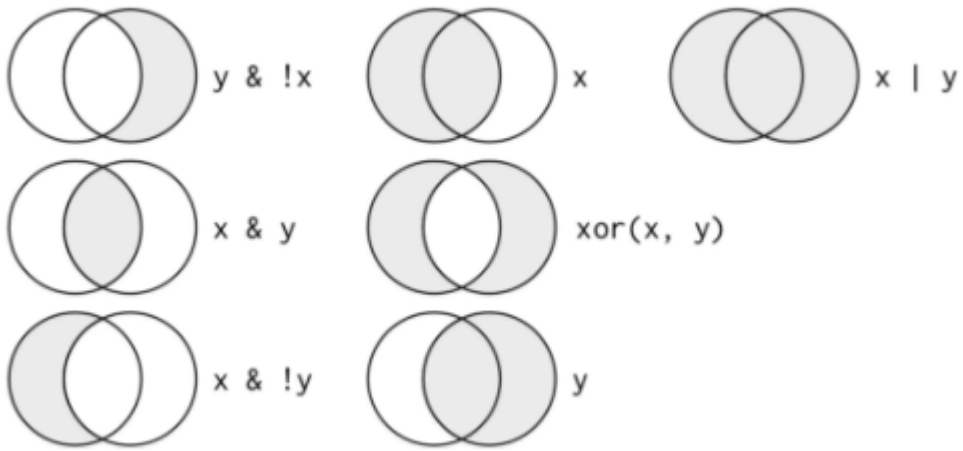
```
first <- filter(flights, month == 10 | day == 1)
head(first)
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     1      517            515         2      830            819
## 2  2013     1     1      533            529         4      850            830
## 3  2013     1     1      542            540         2      923            850
## 4  2013     1     1      544            545        -1     1004           1022
## 5  2013     1     1      554            600        -6      812            837
## 6  2013     1     1      554            558        -4      740            728
## # ... with 11 more variables: 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>
```

Comparison operators

`>`, `>=`, `<`, `<=`	Strict larger, larger or equal, ...
`!=`, `==`	Not equal, equal
`month == 7 | month == 8`, or alternatively `month %in% c(7,8)`	`month` $\in \{7, 8\}$

Visualization of Boolean operators



- Select flights on the 1st day of the month or flights of October but not the flights on the 1st of October

```
filter2 <- filter(flights, xor(month==10, day==1)) # Note the use of 'xor'
head(filter2)
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     1      517            515         2      830            819
## 2  2013     1     1      533            529         4      850            830
## 3  2013     1     1      542            540         2      923            850
```

```
## 4  2013     1     1      544            545        -1     1004           1022
## 5  2013     1     1      554            600        -6      812            837
## 6  2013     1     1      554            558        -4      740            728
## # ... with 11 more variables: 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>
```

2.2 Select variables: `select()`

The `select()` function selects variables in a data frame.

We first check the names of all the variables of the data frame `flights`:

```
names(flights)
```

```
##  [1] "year"           "month"          "day"            "dep_time"
##  [5] "sched_dep_time" "dep_delay"      "arr_time"       "sched_arr_time"
##  [9] "arr_delay"      "carrier"        "flight"         "tailnum"
## [13] "origin"         "dest"           "air_time"       "distance"
## [17] "hour"           "minute"         "time_hour"
```

There are several options to select some of these variables:

- Select the variables by *specifying the name*

```
select1 <- select(flights, year, month, day, dest)
names(select1)
```

```
## [1] "year"  "month" "day"   "dest"
```

- Select variables *between certain variables*.
  e.g. select the variables between `origin` and `distance` (inclusive):

```
select2 <- select(flights, origin:distance)
names(select2)
```

```
## [1] "origin"   "dest"     "air_time" "distance"
```

- Select all variables *except certain variables*.
  e.g. select the variables except the variables `origin`, `distance` and all the variables betweeen `origin` and `distance`:

```
select3 <- select(flights, -(origin:distance))
names(select3)
```

```
##  [1] "year"           "month"          "day"            "dep_time"
##  [5] "sched_dep_time" "dep_delay"      "arr_time"       "sched_arr_time"
##  [9] "arr_delay"      "carrier"        "flight"         "tailnum"
## [13] "hour"           "minute"         "time_hour"
```

- Select variables by their column number

```
select4 <- select(flights, c(3:6, 9))
names(select4)
```

```
## [1] "day"            "dep_time"       "sched_dep_time" "dep_delay"
## [5] "arr_delay"
```

- ***Helper functions***
  Helper functions which can be used in `select()`:

`Starts_with("abc")`	Matches names beginning with *abc*
`Ends_with("xyz")`	Matches names ending with *xyz*
`Contains("ijk")`	Matches names containing *ijk*

2.3 Sorting data frame by one (or more) of its variables: `arrange()`

The function `arrange()` orders the rows of a data frame by the values of selected columns.

Example:
Sort the data frame `select1` by `year`, `month`, `day` and descending `destination`

```
sort1 <- arrange(select1, year, month, day, desc(dest))
head(sort1)
```

```
## # A tibble: 6 x 4
##    year month   day dest
##   <int> <int> <int> <chr>
## 1  2013     1     1 XNA
## 2  2013     1     1 XNA
## 3  2013     1     1 XNA
## 4  2013     1     1 TYS
## 5  2013     1     1 TUL
## 6  2013     1     1 TPA
```

Remark: Missing values:
Missing values are always sorted at the end for logical, numerical, and character variables.

Example:
We create a tibble with missing data:

```
df2 <- tibble(x = c(5, 2, NA), y = c('start', NA, 'end'), z = c(NA, TRUE, FALSE))
df2
```

```
## # A tibble: 3 x 3
##       x y     z
##   <dbl> <chr> <lgl>
## 1     5 start NA
## 2     2 <NA>  TRUE
## 3    NA end   FALSE
```

```
arrange(df2, x)
```

```
## # A tibble: 3 x 3
##       x y     z
##   <dbl> <chr> <lgl>
## 1     2 <NA>  TRUE
## 2     5 start NA
## 3    NA end   FALSE
```

```
arrange(df2, desc(x))
```

```
## # A tibble: 3 x 3
##       x y     z
##   <dbl> <chr> <lgl>
## 1     5 start NA
## 2     2 <NA>  TRUE
## 3    NA end   FALSE
```

```
arrange(df2, y)
```

```
## # A tibble: 3 x 3
##       x y     z
##   <dbl> <chr> <lgl>
## 1    NA end   FALSE
## 2     5 start NA
## 3     2 <NA>  TRUE
```

```
arrange(df2, z)
```

```
## # A tibble: 3 x 3
##       x y     z
##   <dbl> <chr> <lgl>
## 1    NA end   FALSE
## 2     2 <NA>  TRUE
## 3     5 start NA
```

2.4 Create new variables: `mutate()` and `transmute()`

The function `mutate()` adds new variables and preserves existing ones. The function `transmute()` adds new variables and drops existing ones.

Example of `mutate()`:

1. Create a new tibble `flights_sml` by selecting the variables `dep_delay`, `arr_delay`, `distance`, and `air_time`
2. Add 2 new variables:
   - `extra` = `arr_delay - dep_delay`
   - `hours` = `air_time` / 60

Step 1:

```
flights_sml <- select(flights, ends_with("delay"), distance, air_time)
names(flights_sml)
```

```
## [1] "dep_delay" "arr_delay" "distance"  "air_time"
```

Step 2:

```
new1 <- mutate(flights_sml,
               extra = arr_delay - dep_delay,
               hours = air_time/60)
head(new1)
```

```
## # A tibble: 6 x 6
##   dep_delay arr_delay distance air_time extra hours
##       <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl>
## 1         2        11     1400      227     9  3.78
## 2         4        20     1416      227    16  3.78
## 3         2        33     1089      160    31  2.67
## 4        -1       -18     1576      183   -17  3.05
## 5        -6       -25      762      116   -19  1.93
## 6        -4        12      719      150    16  2.5
```

Example of `transmute()`:
If you only want to keep the new variables `extra` and `hours`

```
new2 <- transmute(flights_sml,
                  extra = arr_delay - dep_delay,
                  hours = air_time/60)
head(new2)
```

```
## # A tibble: 6 x 2
##   extra hours
##   <dbl> <dbl>
## 1     9  3.78
## 2    16  3.78
## 3    31  2.67
## 4   -17  3.05
## 5   -19  1.93
## 6    16  2.5
```

2.5 Grouped summaries: `summarise()`

The function `summarise()` aggregates or summarises the input data. This function is usually used on grouped data which is created by the function `group_by()`. If there are grouping variables, the returned data frame will have one (or more) rows for each combination of grouping variables. The output will have one column fr each grouping variable and one column for each of the summary statistics that you have specified.
If there are no grouping variables, the output will have a single row summarising all observations.

2.5.1 Without grouping variable

Using `summarize()` in the absence of grouping variables:
All observations are summarized in single value per summary statistic.

```
summarise(flights, Avgdelay = mean(dep_delay, na.rm=TRUE))
```

```
## # A tibble: 1 x 1
##   Avgdelay
##      <dbl>
## 1     12.6
```

Remark: How is R handling missing data:
Arithmetic functions on missing values yield missing values. For some functions, missing data can be neglected by using the option **`na.rm = TRUE`**. `na.rm` is a logical value indicating whether `NA` values should be deleted before the computation proceeds.

Example: *Titanic*
Creation of vector with missing data

```
test <- c(1:10, NA, NA)
test
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 NA NA
```

Calculating the mean value of this vector

```
mean1 <- mean(test, na.rm = TRUE)
mean1
```

```
## [1] 5.5
```

```
mean2 <- mean(test, na.rm = FALSE)
mean2
```

```
## [1] NA
```

The option `na.rm` is also available in other descriptive statistics (e.g., `median()`, `quantile()`, `sd()`, `var()`, `min()`, `max()`, ...).

There exist a function **`is.na()`** to check whether a vector contains missing values.

```
df2 <- tibble(x = c(5, 2, NA))
x1 <- is.na(df2$x)
x1
```

```
## [1] FALSE FALSE  TRUE
```

```
x2 <- !is.na(df2$x)
x2
```

```
## [1]  TRUE  TRUE FALSE
```

2.5.2 Grouping variable is present

Using `summarize()` in the presence of grouping variables:
For each summary statistic, there is an output value per group of observations. To compute summary statistics for every level of a categorical grouping variable or combination of levels of grouping variables, two functions are required:

1. First, the function `group_by()` is used to create a grouped tibble.
2. Afterwards, the function `summarise()` is applied on this grouped tibble to compute summary statistics.

```
# Create a grouped tibble from an existing tibble.
# Grouping tibble 'flights' by date
by_date <- group_by(flights, year, month, day)

# Compute the mean value per group
summarise(by_date, Avgdelay = mean(dep_delay, na.rm = TRUE))
```

```
## # A tibble: 365 x 4
## # Groups:   year, month [12]
##     year month   day Avgdelay
##    <int> <int> <int>    <dbl>
##  1  2013     1     1    11.5
##  2  2013     1     2    13.9
##  3  2013     1     3    11.0
##  4  2013     1     4     8.95
##  5  2013     1     5     5.73
##  6  2013     1     6     7.15
##  7  2013     1     7     5.42
##  8  2013     1     8     2.55
##  9  2013     1     9     2.28
## 10  2013     1    10     2.84
## # ... with 355 more rows
```

```
# Compute the number of flights per day
summarise(by_date, n_flights = n())
```

```
## # A tibble: 365 x 4
## # Groups:   year, month [12]
##     year month   day n_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
```

Remark:
You can use the output to compute other aggregate information.

2.5.3 Useful summary functions

Measure of ...	Summary functions
Center	`mean()`, `median()`
Spread	`sd()`, `IQR()`
Range	`min()`, `max()`, `quantile()`
Position	`first()`, `last()`, `nth()`
Count	`n()`, `n_distinct()`, `sum(!is.na)` (to count number of non-missing)

Example 1

1. Compute by destination:
   - average arrival delay
   - standard deviation of arrival delay
   - max arrival delay
   - count how many non-cancelled flights you have. A cancelled flight is a flight with a `NA` for 'arr_delay.
2. Order these destinations so that the destination with the highest number of flights is on top and then according to increasing average delay.

```
fl_group <- group_by(flights, dest)
fl_descr <- summarise(fl_group,
              avg_delay = mean(arr_delay, na.rm = TRUE),
              sd_delay = sd(arr_delay, na.rm = TRUE),
              max_delay = max(arr_delay, na.rm = TRUE),
              count = sum(!is.na(arr_delay)))
arrange(fl_descr, desc(count), avg_delay)
```

```
## # A tibble: 105 x 5
##    dest  avg_delay sd_delay max_delay count
##    <chr>     <dbl>    <dbl>     <dbl> <int>
##  1 ATL      11.3       47.0       895 16837
##  2 ORD       5.88      48.0      1109 16566
##  3 LAX       0.547     39.8       784 16026
##  4 BOS       2.91      38.3       422 15022
##  5 MCO       5.45      42.0       744 13967
##  6 CLT       7.36      41.0       744 13674
##  7 SFO       2.67      47.7      1007 13173
##  8 FLL       8.08      42.9       405 11897
##  9 MIA       0.299     41.3       878 11593
## 10 DCA       9.07      39.9       384  9111
## # ... with 95 more rows
```

Example 2

1. Compute the proportion of flights (by destination) which has an arrival delay of more than 1 hour.
2. Select only those observations where total number of non-missing flights > 100.
3. Order these by proportion long delay.

```
fl_group <- group_by(flights, dest)
fl_descr <- summarise(fl_group,
                count = sum(!is.na(arr_delay)),
                prop_long_delay = sum(arr_delay > 60, na.rm = TRUE) / count )
sub1 <- filter(fl_descr, count > 100)
arrange(sub1, prop_long_delay)
```

```
## # A tibble: 92 x 3
##    dest  count prop_long_delay
##    <chr> <int>           <dbl>
##  1 SNA     812          0.0283
##  2 STT     518          0.0347
##  3 HNL     701          0.0442
##  4 ACK     264          0.0455
##  5 MVY     210          0.0524
##  6 SLC    2451          0.0539
##  7 LAS    5952          0.0553
##  8 RSW    3502          0.0571
##  9 MIA   11593          0.0590
## 10 LAX   16026          0.0600
## # ... with 82 more rows
```

3 Combining multiple operations with the pipe

You can avoid creating data frames at every step by using **pipe %>%**.

`x %>% f(y)` turns into $f(x, y)$ `x %>% f(y) %>% g(z)` turns into $g(f(x, y), z)$

Example 1

1. Compute by destination:
   - average arrival delay
   - standard deviation of arrival delay
   - max arrival delay
   - count how many non-cancelled flights you have. A cancelled flight is a flight with a `NA` for 'arr_delay.
2. Order these destinations so that the destination with the highest number of flights is on top and then according to increasing average delay.

Example 1 **without** the use of pipe

```
fl_group <- group_by(flights, dest)
fl_descr <- summarise(fl_group,
              avg_delay = mean(arr_delay, na.rm = TRUE),
              sd_delay = sd(arr_delay, na.rm = TRUE),
              max_delay = max(arr_delay, na.rm = TRUE),
              count = sum(!is.na(arr_delay)))
sort_example1 <- arrange(fl_descr, desc(count), avg_delay)
```

Example 1 **with** the use of pipe

```
sort_example1 <- flights %>%
          group_by(dest) %>%
          summarise(
              avg_delay = mean(arr_delay, na.rm = T),
              sd_delay = sd(arr_delay, na.rm = T),
              count = sum(!is.na(arr_delay))
                        ) %>%
          arrange(desc(count), avg_delay)
```

Example 2

1. Compute the proportion of flights (by destination) which has an arrival delay of more than 1 hour.
2. Select only those observations where total number of non-missing flights > 100.
3. Order these by proportion long delay.

Example 2 **without** the use of pipe

```
fl_group <- group_by(flights, dest)
fl_descr <- summarise(fl_group,
                count = sum(!is.na(arr_delay)),
                prop_long_delay = sum(arr_delay > 60, na.rm = TRUE) / count )
sub1 <- filter(fl_descr, count > 100)
arrange(sub1, prop_long_delay)
```

Example 2 **with** the use of pipe

```
sort_example2 <- flights %>%
          group_by(dest) %>%
          summarise(
              count = sum(!is.na(arr_delay)),
              prop_long_delay = sum(arr_delay > 60, na.rm = T)/ count
                      ) %>%
          filter(count > 100) %>%
          arrange(prop_long_delay)
```

Example 3

Imagine that we want to explore the relationship between the *average distance* and *average arrival delay* for every destination.

1. Group data frame `flights` by destination
2. Compute average distance (`Avgdist`), average arrival delay (`AvgAdelay`) and number of flights (`count`)
3. Filter the obtained dataset because we are only interested in destination with at least 21 flights.
4. Make a scatterplot of average distance (X) versus average delay (Y). Fit a regression line.
5. We see that there is a destination which completely determines the regression line. Remover that data point and make the plot again.

Example 3 **without** the use of pipe

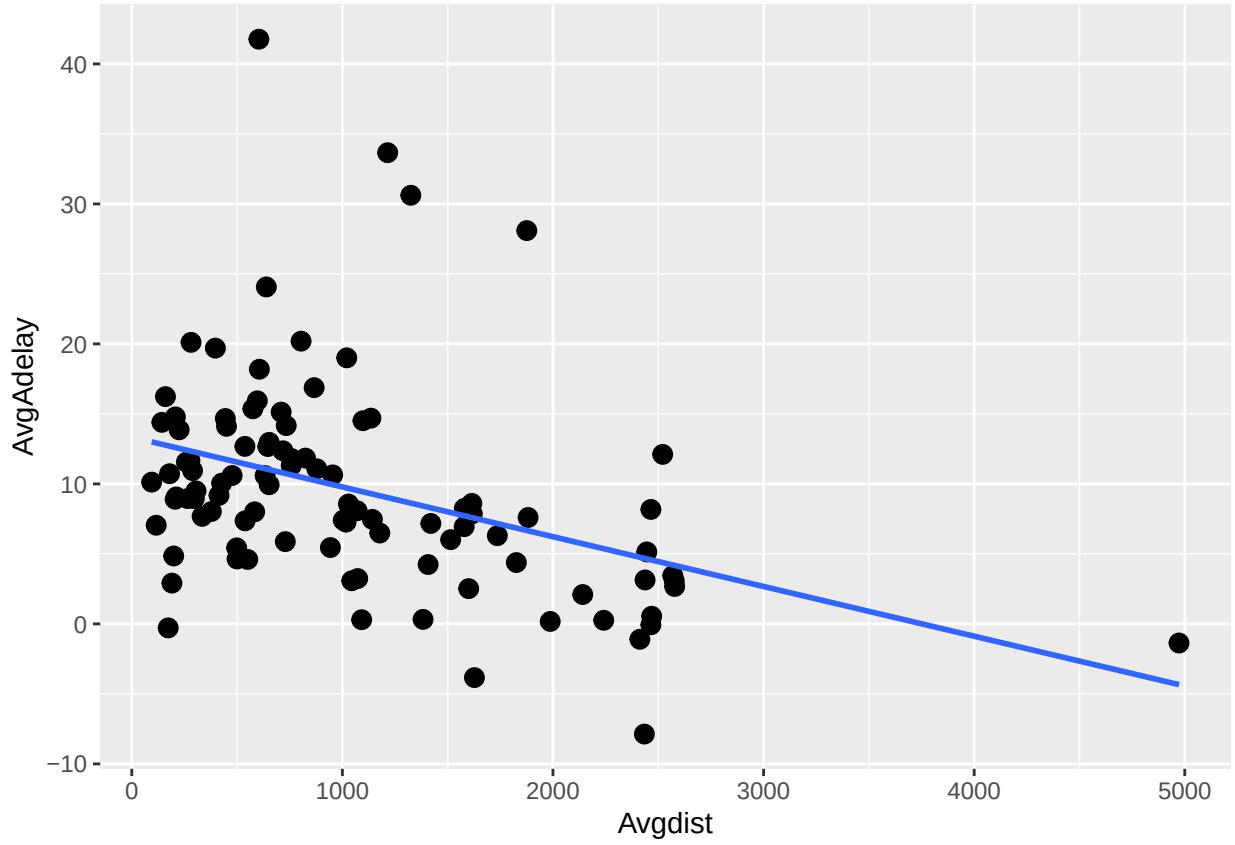
```
# Step 1
by_dest <- group_by(flights, dest)

# Step 2
sum_dest <- summarise(by_dest,
                      count = n(),
                      Avgdist = mean(distance, na.rm = TRUE),
                      AvgAdelay = mean(arr_delay, na.rm = TRUE))

# Step 3
```

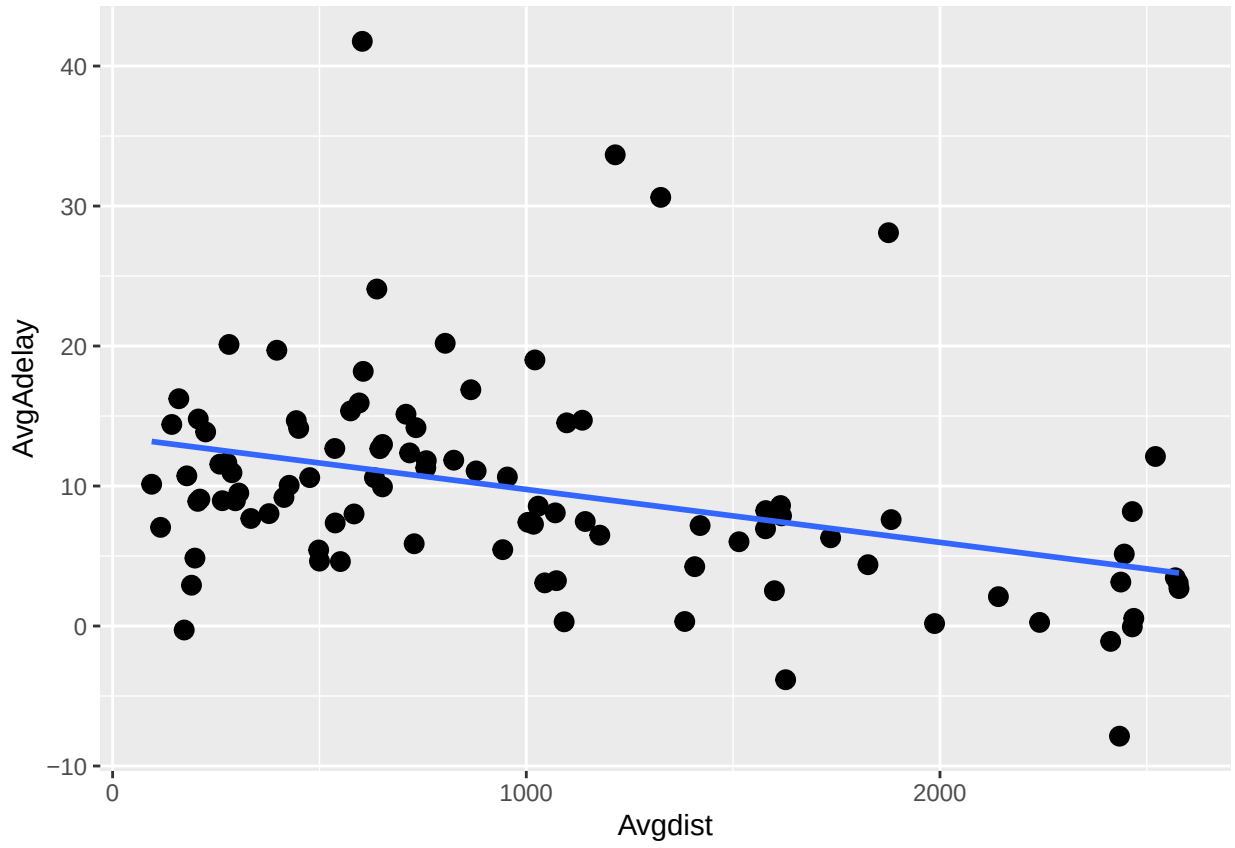
```r
sum_dest_sub <- filter(sum_dest, count > 20)

# Step 4
ggplot(data = sum_dest_sub, aes(x = Avgdist, y = AvgAdelay)) +
  geom_point(size = 3) + geom_smooth(se = FALSE, method = 'lm')
```



We see that there is a destination which completely determines the regression line. Remove that data point.

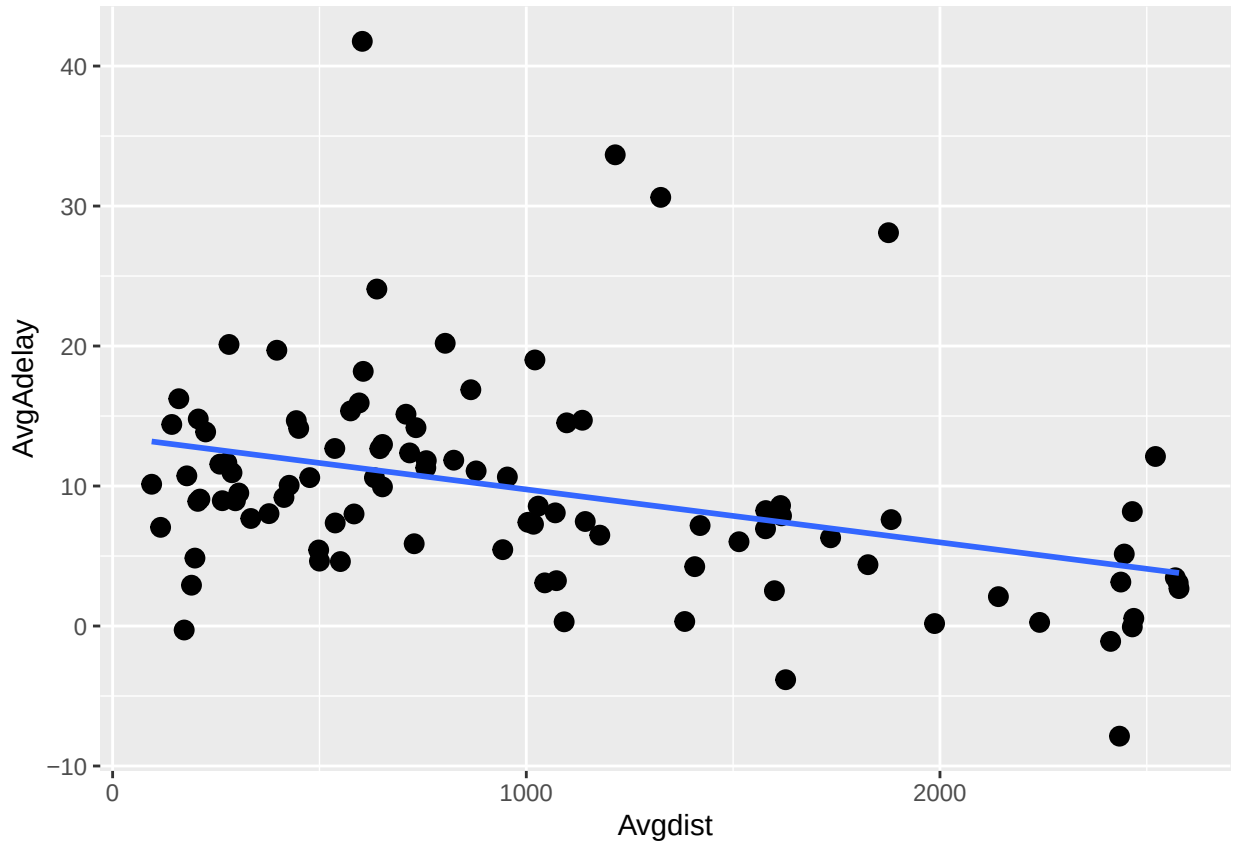
```r
# Step 5
sum_dest_sub2 <- filter(sum_dest_sub, Avgdist < 3000)
ggplot(data = sum_dest_sub2, aes(x = Avgdist, y = AvgAdelay)) +
  geom_point(size = 3) + geom_smooth(se = FALSE, method = 'lm')
```



Example 3 **with** the use of pipe

```
sum_dest_sub2 <- flights %>%
  group_by(dest) %>%
  summarise(
    count = n(),
    Avgdist = mean(distance, na.rm = TRUE),
    AvgAdelay = mean(arr_delay, na.rm = TRUE)
  ) %>%
  filter(count > 20, Avgdist < 3000)

ggplot(data = sum_dest_sub2, aes(x = Avgdist, y = AvgAdelay)) +
  geom_point(size = 3) + geom_smooth(se = FALSE, method = 'lm')
```



Remark:
When not to use pipe:

1. When you want to debug your code, you will need intermediate steps.
2. When you have multiple inputs (e.g. combining many data frames)

4 Integration of multiple sources: JOIN

1. Matching when key variables have a different name in the two data sets

Example *Offenses 1993 & 1994*
Import *stat93_nmk.xlsx* (reporting of offenses happened in 1993) and import *stat94_nmk.xlsx* (reporting of offenses happened in 1994).

```
stat93_nmk
```

```
## # A tibble: 7 x 3
##   off_code93 DESCR                          yr93
##   <chr>      <chr>                         <dbl>
## 1 1102       Making threats to kill           34
## 2 1112       Assault on an male person       300
## 3 1154       Attempted rape                    7
## 4 1214       Theft form shops                292
## 5 1220       Robbery                           6
## 6 1305       Delivering drugs to a person     64
## 7 1307       Trafficking in controlled drugs   1
```

```
stat94_nmk
```

```
## # A tibble: 6 x 3
##   off_code94 DESCR                           yr94
##   <chr>      <chr>                          <dbl>
## 1 1102       Making threats to kill            56
## 2 1112       Assault on an male person        297
## 3 1154       Attempted rape                     2
## 4 1220       Robbery                           15
## 5 1307       Trafficking in controlled drugs    1
## 6 1311       stalking                          30
```

These data sets can be merged in different ways. The key variable in each data table is `off_code93` or `off_code94`.

4.1 Inner join

An inner join matches pairs of observations whenever their keys are equal. It keeps observations that appear in both tables.
Function: `inner_join()`

```
injoin <- inner_join(stat93_nmk, stat94_nmk, by = c("off_code93" = "off_code94"))
injoin[, c(1:3, 5)]
```

```
## # A tibble: 5 x 4
##   off_code93 DESCR.x                         yr93  yr94
##   <chr>      <chr>                          <dbl> <dbl>
## 1 1102       Making threats to kill            34    56
## 2 1112       Assault on an male person        300   297
## 3 1154       Attempted rape                     7     2
## 4 1220       Robbery                            6    15
## 5 1307       Trafficking in controlled drugs    1     1
```

Remark:
Unmatched rows are deleted (e.g. observations with off_code 1214)

4.2 Outer join

An outer join keeps observations that appear in at least one of the tables.

Function:

- `left_join(x, y)`: Left join keeps all observations in `x`
- `right_join(x, y)`: Right join keeps all observations in `y`
- `full_join(x, y)`: Full join keeps all observations that appear in `x` or `y`

```
Ljoin <- left_join(stat93_nmk, stat94_nmk, by = c("off_code93" = "off_code94"))
Ljoin[, c(1:3, 5)]
```

```
## # A tibble: 7 x 4
##   off_code93 DESCR.x                         yr93  yr94
##   <chr>      <chr>                          <dbl> <dbl>
## 1 1102       Making threats to kill            34    56
## 2 1112       Assault on an male person        300   297
## 3 1154       Attempted rape                     7     2
## 4 1214       Theft form shops                 292    NA
## 5 1220       Robbery                            6    15
## 6 1305       Delivering drugs to a person      64    NA
```

```
## 7 1307       Trafficking in controlled drugs     1     1
```

```
Rjoin <- right_join(stat93_nmk, stat94_nmk, by = c("off_code93" = "off_code94"))
Rjoin[, c(1:3, 5)]
```

```
## # A tibble: 6 x 4
##   off_code93 DESCR.x                          yr93  yr94
##   <chr>      <chr>                           <dbl> <dbl>
## 1 1102       Making threats to kill             34    56
## 2 1112       Assault on an male person         300   297
## 3 1154       Attempted rape                      7     2
## 4 1220       Robbery                             6    15
## 5 1307       Trafficking in controlled drugs     1     1
## 6 1311       <NA>                               NA    30
```

```
Fjoin <- full_join(stat93_nmk, stat94_nmk, by = c("off_code93" = "off_code94"))
Fjoin
```

```
## # A tibble: 8 x 5
##   off_code93 DESCR.x                     yr93 DESCR.y                     yr94
##   <chr>      <chr>                      <dbl> <chr>                      <dbl>
## 1 1102       Making threats to kill        34 Making threats to kill        56
## 2 1112       Assault on an male person    300 Assault on an male person    297
## 3 1154       Attempted rape                 7 Attempted rape                 2
## 4 1214       Theft form shops             292 <NA>                          NA
## 5 1220       Robbery                        6 Robbery                        15
## 6 1305       Delivering drugs to a pers~   64 <NA>                          NA
## 7 1307       Trafficking in controlled ~    1 Trafficking in controlled ~    1
## 8 1311       <NA>                          NA stalking                      30
```

Remark:
the `base::merge()` can perform all four types of join:

dplyr	merge
`inner_join(x, y)`	`merge(x, y)`
`left_join(x, y)`	`merge(x, y, all.x = TRUE)`
`right_join(x, y)`	`merge(x, y, all.y = TRUE)`
`full_join(x, y)`	`merge(x, y, all.x = TRUE, all.y = TRUE)`

5 Differences between a *tibble* and a *data frame*

Most R packages work with data frames, sometimes you need to be able to switch between a tibble and a data frame.

- Make from a data frame a tibble by using the function `as_tibble()`

```
tips.tib <- as_tibble(tips)
head(tips.tib)
```

```
## # A tibble: 6 x 7
##   total_bill   tip sex    smoker day   time    size
##        <dbl> <dbl> <fct>  <fct>  <fct> <fct>  <int>
## 1       17.0  1.01 Female No     Sun   Dinner     2
## 2       10.3  1.66 Male   No     Sun   Dinner     3
## 3       21.0  3.5  Male   No     Sun   Dinner     3
```

```
## 4       23.7  3.31 Male   No     Sun   Dinner     2
## 5       24.6  3.61 Female No     Sun   Dinner     4
## 6       25.3  4.71 Male   No     Sun   Dinner     4
```

- Make from a tibble a data frame by using the function`as_data_frame`

```
flights.df <- as_data_frame(flights)
```

```
## Warning: `as_data_frame()` is deprecated, use `as_tibble()` (but mind the new semantics).
## This warning is displayed once per session.
```

```
head(flights.df)
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     1      517            515         2      830            819
## 2  2013     1     1      533            529         4      850            830
## 3  2013     1     1      542            540         2      923            850
## 4  2013     1     1      544            545        -1     1004           1022
## 5  2013     1     1      554            600        -6      812            837
## 6  2013     1     1      554            558        -4      740            728
## # ... with 11 more variables: 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>
```

6 Exercises

6.1 Exercise `tips`

In this exercise, the data set `tips` from the package `reshape` will be used.

One waiter recorded information about each tip he received over a period of a few months working in one restaurant. He collected several variables:

- `tip`: tip in dollars
- `total_bill`: bill in dollars
- `sex`: sex of the bill payer
- `smoker`: whether there were smokers in the party
- `day`: day of the week
- `time`: time of the day
- `size`: size of the party

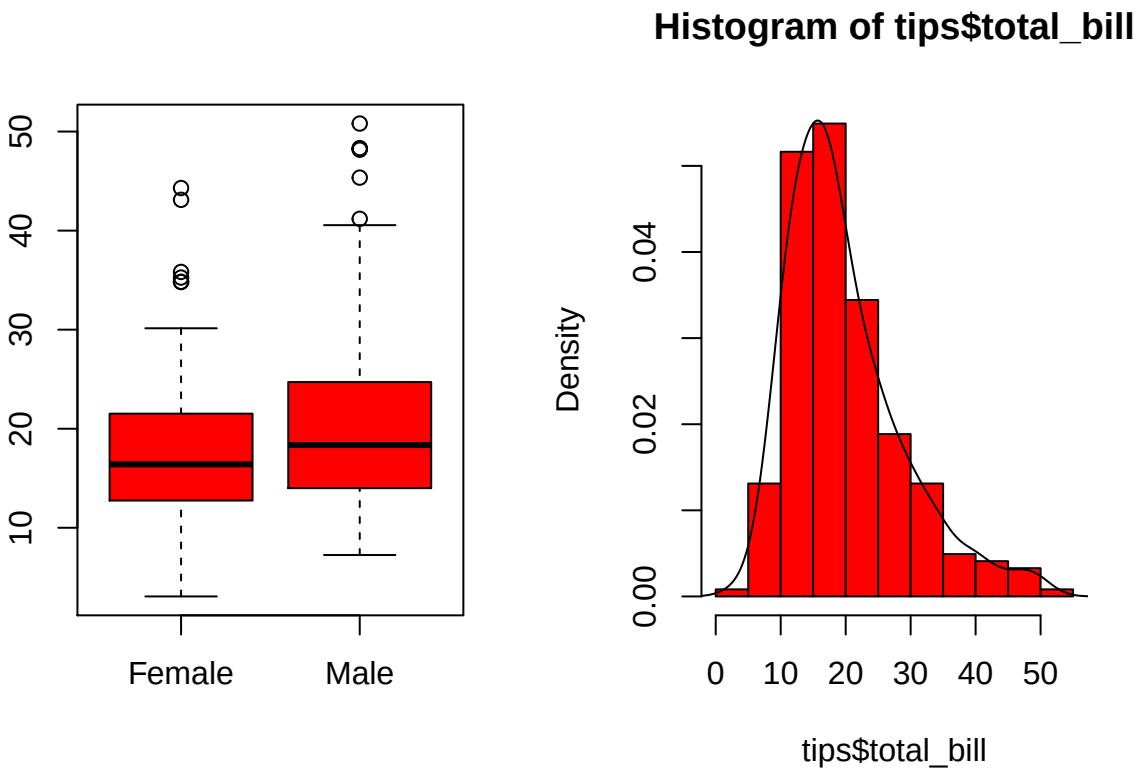
In all he recorded 244 tips.

1. Create a tibble `tips.tbl` from the data frame `tips`
2. Create a subset `sub1` from `tips.tbl` with only those observations with gender `Male` **and** with size of the table larger than or equal to three.
3. a) Create a subset `sub2` from `tips.tbl` with only those observations with gender `Male` **or** with size of the table larger than or equal to three or with both gender `Male` and size ≥ 3.
   b) Create a subset `sub3` from `sub2` with all variables except smoker.
   c) Create a tibble `sort1` by sorting `sub3` by time and decreasing size of table.
   d) Compute average tip by gender (use `sort1`)
   e) Use now the pipe operator to do steps a-d of exercise 3. (start from `tips.tbl`)
4. Compute average tip by day of the week (use the tibble `tips.tbl`).
5. Compute average tip by gender and day of the week (use the tibble `tips.tbl`).

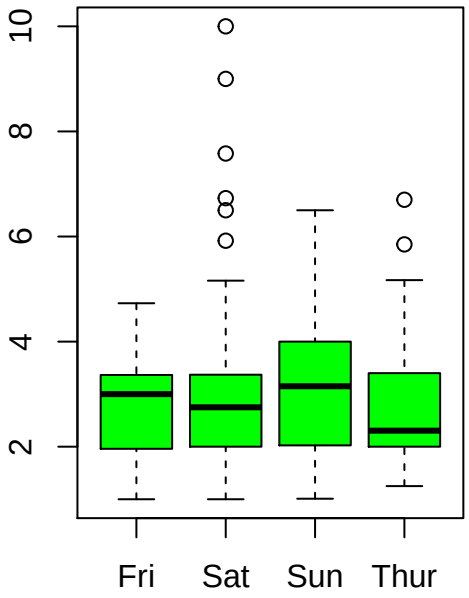
6.2 Overall exercise

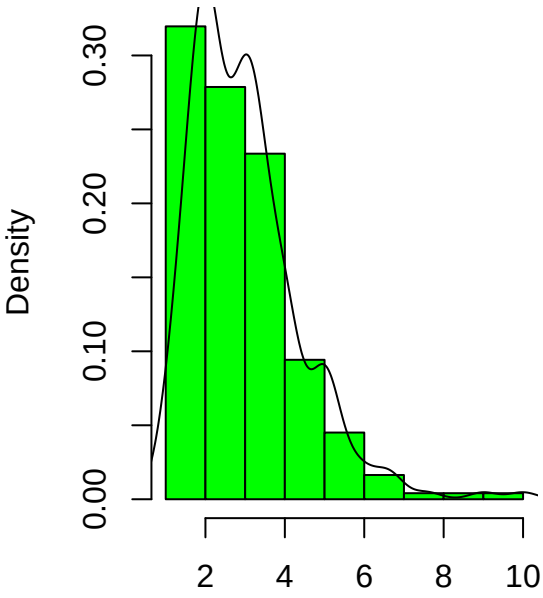
In this exercise, the data `tips` from the package `reshape` will be used.

1. Compute descriptive statistics (mean, median, n, stdev) of the variable `total_bill` by gender of the bill payer.
2. Make a grouped boxplot of total bill by gender of the bill payer.
3. Create a histogram of total bill and overlay it with a density curve.
4. Put both graphs next to each other in one graphical window (see below)



5. Create now a function `VISUAL` which is doing subquestion 2, 3, and 4 for you. The input parameters for this function are
   a) `DFR` which is the name of your data frame
   b) `CONT` which is the continuous variable
   c) `CAT` which is the categorical variable
   d) `COL1` which gives you the color number of your graph
6. Apply this function now to obtain visuals for tip by day of the week as below:





6.3 Exercise, missing data

1. Use the `airquality` data frame form the package `datasets`. Use the `summary` function and interpret the result.
2. Count the number of rows in this `airquality` dataframe (use `nrow` function)
3. Count the number of missing values for variable `Ozone`
4. Create a subset `air_complete` of `airquality` with only the complete cases (hint: you can use the function `complete.cases`).
5. Count the number of rows in `air_complete`.