Solutions to Exercise Sheet - Data Manipulation

1. Flights - delay vs. time.

Load the package `nycflights13` which includes the data set `flights`:

```
require(nycflights13)
```

Take a first look at the data set:

```
str(flights)
head(flights, n = 3)
summary(flights)
```

a) Define the new variable

```
flights$time <- flights$hour + flights$minute / 60
```

b) Calculate the average delay per hour:

- using `dplyr`:

```
delay.per.hour  <- dplyr::summarise(dplyr::group_by(flights, time),
                                     arr_delay = mean(arr_delay,
                                                      na.rm = TRUE),
                                     n = n())
```

- using `plyr`:

```
delay.per.hour  <- plyr::ddply(.data = flights, .variables = "time",
                               plyr::summarise,
                               arr_delay = mean(arr_delay,
                                                na.rm = TRUE),
                               n = length(time))
```

- using “built-in functions” `aggregate()` of R:
  - Alternative 1:

```
delay.hour.temp <- aggregate(formula = arr_delay ~ time,
                             data = flights,
                             FUN = function(x)
                               {c(mean = mean(x, na.rm = TRUE),
                                  n = length(x))})
```

Note that it is a bit different to access the columns `mean` and `n` since they are saved in a matrix, which is saved in a variable in the data frame `delay.hour.temp`, i.e.

```
delay.hour.temp$arr_delay[ , "n"]
delay.hour.temp$arr_delay[ , "mean"]
```

One can change the structure of the data frame in order to make things easier: The next step is not required but it makes it easier to use the same code in the following task below:

```
delay.per.hour <-
  data.frame(time = delay.hour.temp$time,
             arr_delay = delay.hour.temp$arr_delay[ , "mean"],
             n = delay.hour.temp$arr_delay[ , "n"])
```

– Alternative 2:
  Calculate the average delay per hour:

```
delay.per.hour <- aggregate(formula = arr_delay ~ time,
                            data = flights, FUN = mean,
                            na.rm = TRUE)
```

  Calculate the number of plans:

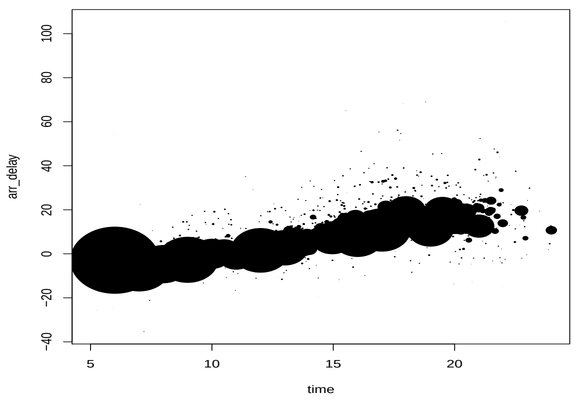
```
d.temp <- aggregate(formula = arr_delay ~ time, data = flights,
                    FUN = length)
```

  Save the variable in the data frame `delay.per.hour`:

```
delay.per.hour$n <- d.temp[ , "arr_delay"]
```

c) Plot the average arrival delay against the time:

```
plot(arr_delay ~ time, data = delay.per.hour, pch =  19,
     cex = n / 500)
```



We observe that the average arrival delay increases with respect to the time of departure of planes. This suggest that you are better off flying in the morning than in the evening. The plan departing after midnight have a very high average arrival delay but there are not many of them. There seems to be an hourly pattern which might be due to how flight are scheduled.

Note: Alternatively, you can make the plot using the package `ggplot2`.

```
require(ggplot2)
ggplot(data = delay.per.hour, aes(time, arr_delay, size = n)) +
  geom_point() + scale_size_area()
```

2. Flights - Dealing with missing values.

a) Redo Exercise 1. b) using the package `dplyr` and the function `aggregate()`:

```
delay.1  <- dplyr::summarise(dplyr::group_by(flights, time),
                             arr_delay = mean(arr_delay, na.rm = TRUE),
                             n = n())
```

```
delay.2 <- aggregate(formula = arr_delay ~ time, data = flights,
                     FUN = function(x) {c(mean = mean(x, na.rm = TRUE),
                                          n = length(x))})
```

Note that the function `dplyr::summarise()` is a bit faster compared to the function `aggregate()`.
Let us compare the dimensions of the two objects `delay.1` and `delay.2`.

```
dim(delay.1)
## [1] 1021    3
dim(delay.2)
## [1] 1020    2
```

As we can see, the dimensions of the two objects are not the same. Now we could also compare the number of flights. Do they match? Let us look at the first 10 values:

```
head(cbind(delay.1$n, delay.2$arr_delay[ , "n"]), n = 10)
## Warning in cbind(delay.1$n, delay.2$arr_delay[, "n"]):  number of rows
of result is not a multiple of vector length (arg 2)
##       [,1] [,2]
##  [1,]    1  340
##  [2,]  341    1
##  [3,]    1    2
##  [4,]    2    5
##  [5,]    5  205
##  [6,]  208    4
##  [7,]    4   27
##  [8,]   28    7
##  [9,]    7   37
## [10,]   37    1
```

Alternatively

```
cbind(delay.1$n[1:10], delay.2$arr_delay[1:10, "n"])
```

Note that the number of flights do not match.

b) We can either adjust the function call from `dplyr::summarise()` or the function call from `aggregate`. Note that we only have to change one of them in order to obtain the same output!

- Either we have to change the `dplyr::summarise()` function call, i.e. the solution is to omit the NA's in the following function call:

```
delay.temp <- dplyr::filter(flights, !is.na(arr_delay), !is.na(time))
delay.3 <- dplyr::summarise(dplyr::group_by(delay.temp, time),
                            arr_delay = mean(arr_delay, na.rm = TRUE),
                            n = n())
```

- Or we can adjust the function call of `aggregate()`, i.e. the solution is to not omit the NA's in the following function call:

```
delay.4 <- aggregate(formula = arr_delay ~ time, data = flights,
                     FUN = function(x) {c(mean = mean(x, na.rm = TRUE),
                                          n = length(x))},
                     na.action = na.pass)
```

c) We want to test if the objects are equal. First we test if `delay.2` is the same as `delay.3`:

```
identical(delay.3$time, delay.2$time)
## [1] TRUE
identical(delay.3$arr_delay, delay.2$arr_delay[ , "mean"])
## [1] TRUE
identical(as.numeric(delay.3$n), as.numeric(delay.2$arr_delay[ , "n"]))
## [1] TRUE
```

Then we test if `delay.1` is the same as `delay.4`. Note that `delay.1` is the same as `delay.4` after removing the planes with a missing value for the variable `time`, i.e. one first has to remove one row of the data `delay.1`.

```
delay.1b <- delay.1[!is.na(delay.1$time), ]

identical(delay.1b$time, delay.4$time)
## [1] TRUE
identical(delay.1b$arr_delay, delay.4$arr_delay[ , "mean"])
## [1] TRUE
identical(as.numeric(delay.1b$n), as.numeric(delay.4$arr_delay[ , "n"]))
## [1] TRUE
```

3. Flights - Plotting the destinations.

a) We can deal with the missing values in two different ways:

- by omitting the missing values:

  ▷ using `dplyr`

  ```
  flights.temp <- dplyr::filter(flights, !is.na(arr_delay))
  delay.per.dest  <-
    dplyr::summarise(dplyr::group_by(flights.temp, dest),
                     arr_delay = mean(arr_delay, na.rm = TRUE),
                     n = n())
  ```

  ▷ using ”built-in function” `aggregate()` in R

  – Version 1:

  ```
  delay.temp <- aggregate(formula = arr_delay ~ dest, data = flights,
                          FUN = function(x)
                            {c(mean = mean(x,na.rm = TRUE),
                               n = length(x))})
  # this step is not required but it makes it easier
  # to use the same code below
  delay.per.dest <-
    data.frame(dest = delay.temp$dest,
               arr_delay = delay.temp$arr_delay[ , "mean"],
               n = delay.temp$arr_delay[ , "n"])
  ```

  – Version 2:

  ```
  delay.per.dest <- aggregate(formula = arr_delay ~ dest,
                              data = flights,
                              FUN = mean, na.rm = TRUE)
  d.temp2 <- aggregate(formula = arr_delay ~ dest, data = flights,
                       FUN = length)
  delay.per.dest$n <- d.temp2[ , "arr_delay"]
  ```

- by keeping the missing values:
  ▷ using `dplyr`

```
delay.per.dest <- dplyr::summarise(dplyr::group_by(flights, dest),
                                   arr_delay = mean(arr_delay,
                                                    na.rm = TRUE),
                                   n = n())
```

  ▷ using "built-in function" `aggregate()` in R
  - Version 1:

```
delay.dest.temp <- aggregate(formula = arr_delay ~ dest,
                             data = flights,
                             FUN = function(x)
                               {c(mean = mean(x, na.rm = TRUE),
                                  n = length(x))},
                             na.action = na.pass)
# this step is not required but it makes it easier
# to use the same code below
delay.per.dest <-
  data.frame(dest = delay.dest.temp$dest,
             arr_delay = delay.dest.temp$arr_delay[ , "mean"],
             n = delay.dest.temp$arr_delay[ , "n"])
```

  - Version 2

```
delay.per.dest <- aggregate(formula = arr_delay ~ dest,
                            data = flights, FUN = mean,
                            na.rm = TRUE, na.action = na.pass)
d.temp2 <- aggregate(formula = arr_delay ~ dest, data = flights,
                     FUN = length, na.action = na.pass)
delay.per.dest$n <- d.temp2[ , "arr_delay"]
```

b) We merge `delay.per.dest` and the coordinates of the airports

- using `dplyr`:

```
delay.per.dest <- dplyr::left_join(x = delay.per.dest, y = airports,
                                   by = c("dest" = "faa"))
```

- using "built-in function" `merge()` in R:

```
delay.per.dest <- merge(x = delay.per.dest, y = airports,
                        by.x = "dest", by.y = "faa",
                        all.x = TRUE, all.y = FALSE)
```

c) We plot the latitude against the longitude and scale the points according to the number of departing planes

```
plot(lat ~ lon, data = delay.per.dest, pch =  19, cex = n / 6000)
```

Alternative using the package `ggplot2`:

```
require(ggplot2)
ggplot(data = delay.per.dest, aes(lon, lat, size = n)) + geom_point()
```

d) BONUS: We add a map of the US to the plot of part c).

```
require(maps)
delay.per.dest2 <- subset(delay.per.dest, lon > -140)
```

```
ggplot(data = delay.per.dest2, aes(lon, lat, size = n)) +
  geom_point() + borders(database = "state", size = 0.5)
```

You can even add some other maps to the plot. Try out following commands:

```
ggplot(data = delay.per.dest2, aes(lon, lat, size = n)) +
  geom_point() + borders(database = "usa", size = 0.5)
```

```
ggplot(data = delay.per.dest2, aes(lon, lat, size = n)) +
  geom_point() + borders(database = "county", size = 0.5)
```