Minimalistic examples of dplyr

In order to retrieve information from data frames we use the package *dplyr*. This package makes filtering, sorting and grouping operations on a data frame very easy.

To install the package, write

```
install.packages("dplyr")
```

After the package has been installed, you have to load it with the following command

```
library(dplyr)
```

The main commands of the dplyr package are:

- **select()**: choosing a subset of columns
- **filter()**: choosing a subset of rows
- **arrange()**: sort the rows
- **mutate()**: add new columns
- **summarise()**: aggregates the values
- **group_by()**: change the data into grouped data in order to apply functions to each of the groups separately
- **top_n()**: choose n first/last rows

The first argument of these functions is always the data.frame and all the functions also return a data.frame object.

Next we will show you some simple examples to demonstrate the functionality of dplyr package.

```
data = data.frame(gender = c("M", "M", "F"),
                  age = c(20, 60, 30),
                  height = c(180, 200, 150))
data
```

```
##   gender age height
## 1      M  20    180
## 2      M  60    200
## 3      F  30    150
```

select()

Selecting a subset of columns.

```
# Example 1
```

```
select(data, age)
```

```
##   age
## 1  20
## 2  60
## 3  30
```

```
# Example 2
select(data, gender, age)
```

```
##   gender age
## 1      M  20
## 2      M  60
## 3      F  30
```

```
# Example 3 (gives the same result as example 2)
select(data, -height)
```

```
##   gender age
## 1      M  20
## 2      M  60
## 3      F  30
```

filter()

Selecting a subset of rows.

```
# Example 1
filter(data, height > 160)
```

```
##   gender age height
## 1      M  20    180
## 2      M  60    200
```

```
# Example 2
filter(data, height > 160, age > 30)
```

```
##   gender age height
## 1      M  60    200
```

```
# Example 3 (gives the same result as example 2)
filter(data, height > 160 & age > 30)
```

```
##   gender age height
```

```
## 1      M  60    200
```

arrange()

Sorts rows.

```
# Example 1
arrange(data, height)
##   gender age height
## 1      F  30    150
## 2      M  20    180
## 3      M  60    200
# Example 2 (sort in decreasing order)
arrange(data, desc(height))
##   gender age height
## 1      M  60    200
## 2      M  20    180
## 3      F  30    150
```

mutate()

Adds new columns.

```
# Example 1
mutate(data, height2 = height / 100)
##   gender age height height2
## 1      M  20    180     1.8
## 2      M  60    200     2.0
## 3      F  30    150     1.5
# Example 2
mutate(data, height2 = height / 100,
       random_feature = height * age)
##   gender age height height2 random_feature
## 1      M  20    180     1.8           3600
## 2      M  60    200     2.0          12000
```

```
## 3      F  30    150     1.5           4500
```

summarise()

Aggregates the values.

```
summarise(data, average_height = mean(height))
```

```
##   average_height
## 1       176.6667
```

group_by()

Changes the data into grouped data where functions are applied separately to each group.

```
grouped_data = group_by(data, gender)


# Applying the function summarise to each group separately
summarise(grouped_data, average_height = mean(height))
```

```
## # A tibble: 2 <U+00D7> 2
##   gender average_height
##   <fctr>          <dbl>
## 1      F            150
## 2      M            190
```

```
# In addition to average height we can also count the number of observations
in that group
summarise(grouped_data,
          average_height = mean(height),
          nr_of_people = n())
```

```
## # A tibble: 2 <U+00D7> 3
##   gender average_height nr_of_people
##   <fctr>          <dbl>        <int>
## 1      F            150            1
## 2      M            190            2
```

These functions can be quite helpful:

- **distinct()**: separate unique values

- **sample_n()**: draw n random values from the selected column
- **n()**: count the number of rows
- **n_distinct()**: count the number of unique values

top_n()

Separates top n values from the dataset by some feature (column). NOTE: The resulting data.frame is not ordered by these values.

```
# top 1 by height
top_n(data, 1, height)
##   gender age height
## 1      M  60    200
# top 2 by height (we can see that it's not sorted)
top_n(data, 2, height)
##   gender age height
## 1      M  20    180
## 2      M  60    200
```

Before finding top n rows, we can apply functions to the column.

```
# bottom 2 by height
top_n(data, 2, -height)
##   gender age height
## 1      M  20    180
## 2      F  30    150
# person whose height is closest to 160
top_n(data, 1, -abs(height - 160))
##   gender age height
## 1      F  30    150
```

Applying multiple functions

Example: Let's sort the data by height and select only the rows where gender == "M".

```
# Version 1
sorted = arrange(data, height)
filter(sorted, gender == "M")
##   gender age height
```

```
## 1      M  20    180
## 2      M  60    200
```

```
# Version 2
filter(arrange(data, height),
       gender == "M")
```

```
##   gender age height
## 1      M  20    180
## 2      M  60    200
```

%>% operator

You can make your code more readable by using the dyplr's *pipe* operator (**%>%**). This operator takes the object from the left and gives it as the first argument to the function on the right. For example the function **f(x, y)** can be written as **x %>% f(y)**.

```
# Version 3 (continuation of the previous example using the pipe operator)
data %>%
  arrange(height) %>%
  filter(gender == "M")
```

```
##   gender age height
## 1      M  20    180
## 2      M  60    200
```

You can read the code written with the pipe operator in the following way:

- Take the dataset called "data", then
  - sort it by height, then
  - extract rows where gender == "M"

Code written in this way is easier to read, especially if multiple functions are applied.

For example the example written before

```
grouped_data = group_by(data, gender)
summarise(grouped_data, average_height = mean(height))
```

can be written as

```
data %>%
  group_by(gender) %>%
  summarise(average_height = mean(height))
```

Additional comment 1. If you need to save the result to a variable you can use the -> operator.

```
data %>%
  group_by(gender) %>%
  summarise(average_height = mean(height)) -> new_data
```

Additional comment 2. The pipe operator can be also used for other functions (not only for dplyr functions).

```
c(1, 3) %>% mean() %>% log(base = 2)
```

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