

III. Graphics with ggplot2 (presentation)

Data Science Laboratory, University of Copenhagen

2026-05-06

Table of contents

Importing libraries and data	1
ggplot2: The basic concepts	3
A simple bar chart	4
Flipping the bar chart	6
Adding monthly download info	7
Some other bar chart options	8
A bar chart with ordered bars	11
Daily summary statistics	12
A simple scatter plot	13
Plotting on the log-scale	14
Points colored by machine	15
Points shaped by machine	16
Bubble plot	17
Points colored by download size	18
Faceting	19
Cumulated total download size over the dates within machines	21
A box plot	22
Saving plots	24
Conclusion	25

Importing libraries and data

The examples below use the *downloads* dataset, which also was used for the presentation **Working with data in R**. This dataset is available as an Excel file, which we will import using the `readxl` package. Furthermore, `ggplot2` is part of the `tidyverse` package, so we also load that:

```
library(readxl)
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr      1.2.0      v readr      2.1.6
v forcats    1.0.1      v stringr    1.6.0
v ggplot2     4.0.2      v tibble     3.3.1
v lubridate  1.9.5      v tidyr      1.3.2
v purrr       1.2.1

-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()     masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
```

We read the *downloads* dataset from the Excel file, and assign it to a *tibble* named *downloads*. Moreover, we only keep the observations, where the *size* variable is strictly larger than zero:

```
downloads <-
  read_excel("downloads.xlsx") %>%
  filter(size > 0)
downloads
```

```
# A tibble: 36,708 x 6
  machineName userID size time date month
  <chr>      <dbl> <dbl> <dbl> <dtm> <chr>
1 cs18      146579 2464 0.493 1995-04-24 00:00:00 1995-04
2 cs18      995988 7745 0.326 1995-04-24 00:00:00 1995-04
3 cs18      317649 6727 0.314 1995-04-24 00:00:00 1995-04
4 cs18      748501 13049 0.583 1995-04-24 00:00:00 1995-04
5 cs18      955815   356 0.259 1995-04-24 00:00:00 1995-04
6 cs18      596819 15063 0.336 1995-04-24 00:00:00 1995-04
7 cs18      169424 2548 0.285 1995-04-24 00:00:00 1995-04
8 cs18      386686 1932 0.286 1995-04-24 00:00:00 1995-04
9 cs18      783767 7294 0.397 1995-04-24 00:00:00 1995-04
10 cs18      788633 4470 3.41 1995-04-24 00:00:00 1995-04
# i 36,698 more rows
```

We manually make a *Table-of-Variables* with the meta-information about the variables in the dataset:

Variable	Description	Range	Usage
<i>machineName</i>	Name the server	categorical (5 levels)	explanatory
<i>userID</i>	Id for user	categorical (35811 levels)	not used
<i>size</i>	download size in bytes	numerical (3 to 14518894)	response
<i>time</i>	download time in seconds	numerical (0.0437 to 1878.0762)	response
<i>date</i>	download date	date (Nov 22, 1994 to May 18, 1995)	explanatory
<i>month</i>	month of the date	categorical (1994-11 to 1995-05)	explanatory

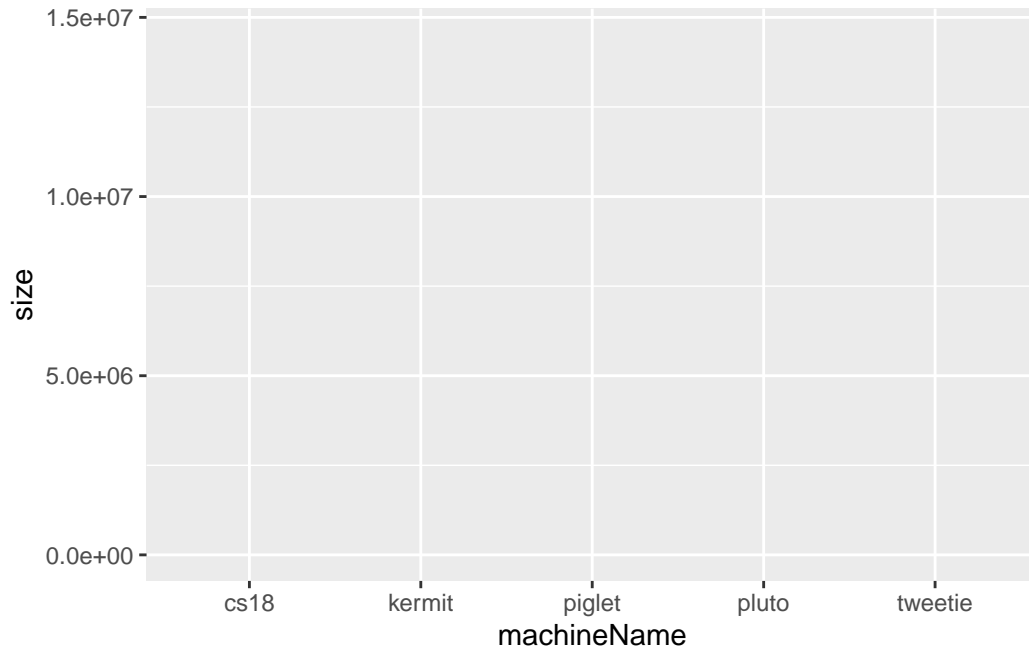
ggplot2: The basic concepts

A ggplot-object is a syntactical description of a plot. You may think of it as a recipe in a cookbook. To actually *cook the dish*, that is to make the plot, you *print* the ggplot-object. By default the results of all R commands executed in the *Console* are automatically printed. Thus, as a result the plot will be generated on the computer screen. If you execute an R script by *sourcing*, then you might need to print explicitly using the `print()` command. Alternatively, you can use the `ggsave()` command to print the syntactical plot description into a graphics file to be used in scientific papers and/or presentations.

To write down a recipe for a dish you usually start with a blank sheet of paper. We do the same for our graphical recipes. The equivalent of a blank sheet of paper is generated by the command `ggplot()`. The ingredients for our plot is a dataset, which should be available as a `data.frame` or as a `tibble`. We should also specify what the ingredients should be used for. In the language of ggplot2 this is done by specifying *aesthetics* via the `aes()` command.

Suppose we want to use data from the tibble `downloads`, and that `machineName` should be on the x-axis and `size` on the y-axis. Then we write

```
ggplot(downloads, aes(x=machineName, y=size))
```

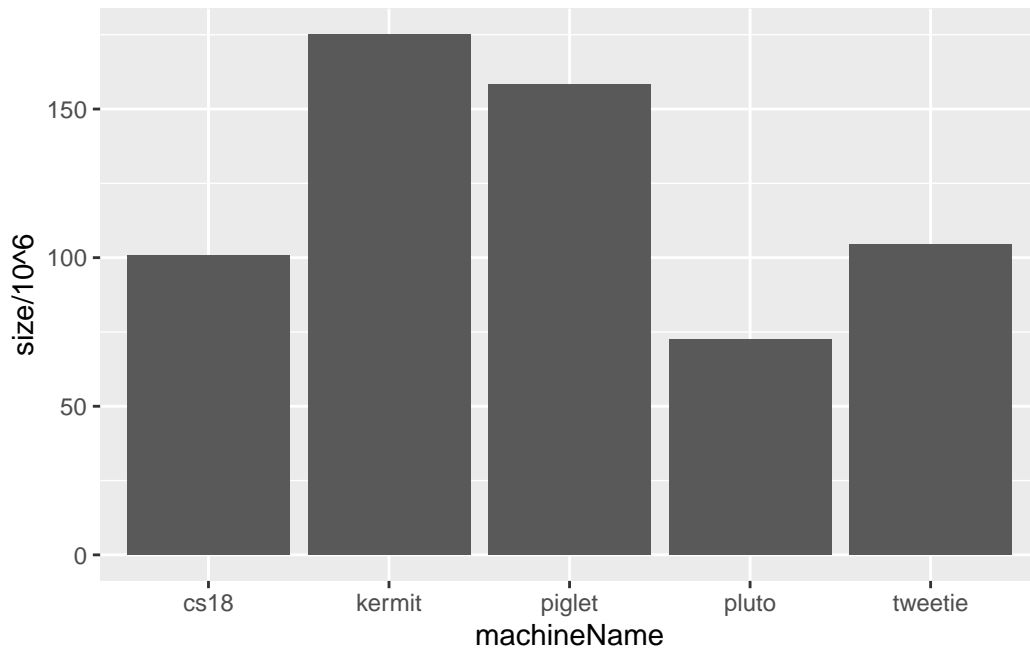


The reason that we don't see any points, lines or the like is, of course, that we did not yet ask for such things to be made! Geometrical objects like points and lines are called **geoms** in ggplot2. But although we did not yet add any geoms to our plot, we see that the plot already recognized the range (and type) of the variables specified in the aesthetics.

A simple bar chart

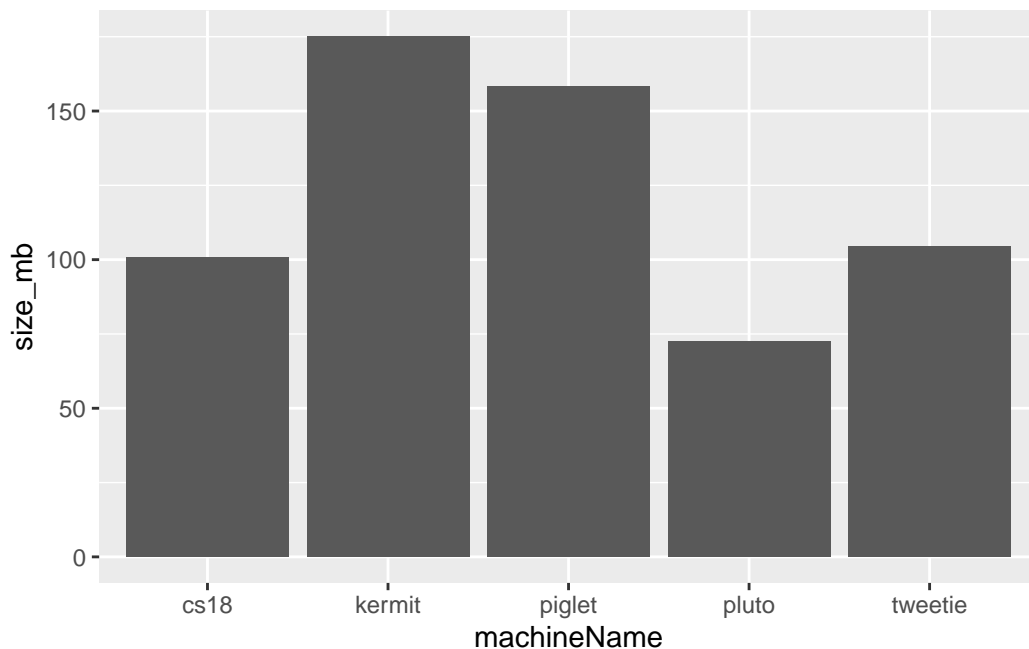
To make a bar chart we add `geom_col()` to the syntactical description. In the code below we have also rescaled the size of the downloaded files to be measured in mega bytes instead of bytes. This is done by downscaling the *size* variables by a factor 1,000,000.

```
ggplot(downloads, aes(x = machineName, y = size/106)) +  
  geom_col()
```



We notice, that `machineName` used on the x-axis is a categorical variable. In R the levels of a categorical variable by default are ordered alphabetically, which is also what we see on this plot. The y-axis shows the total number of mega bytes for each machine, i.e., the *sum* over all observations from each machine. The sum is computed automatically, but we could also have chosen to make those computations “manually” and made the plot like this instead:

```
downloads %>%  
  group_by(machineName) %>%  
  summarize(size_mb = sum(size/10^6)) %>%  
  ggplot(aes(x = machineName, y=size_mb)) + geom_col()
```



Then it is also obvious how to change the code if we wanted the average download size, say, rather than the total download on the y-axis (try it yourself).

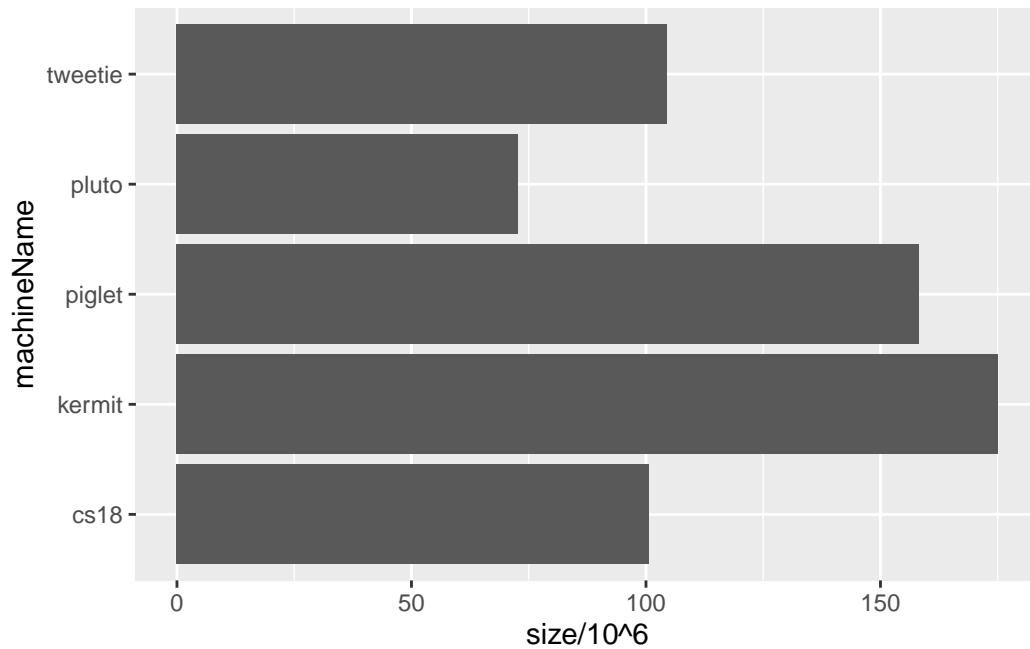
Flipping the bar chart

In the next R chunk we for the first time will try to save the syntactical description of a plot in an R variable. The benefit of doing this is that the syntactical description easily may be reused, possibly with variations. In the metaphor of a *recipe in a cookbook* think of giving a piece of paper with the recipe of your favorite dish to a friend. Then your friend may cook your dish, and possibly also with variations and new additions. Let's try out this idea, and write down the recipe for the bar chart in an R variable called `p`:

```
p <- ggplot(downloads, aes(x = machineName, y = size/106)) +  
  geom_col()
```

This does not yet produce a new plot. But a variable called `p` has appeared in the *global environment*, and if we gave the command `p`, then the graph would be plotted again. Now imagine you give the recipe to your friend. Your friend is happy and thank you for the wonderful recipe, but decide to cook the bar chart with horizontal bars instead of vertical bars. This is done by flipping the coordinate axes:

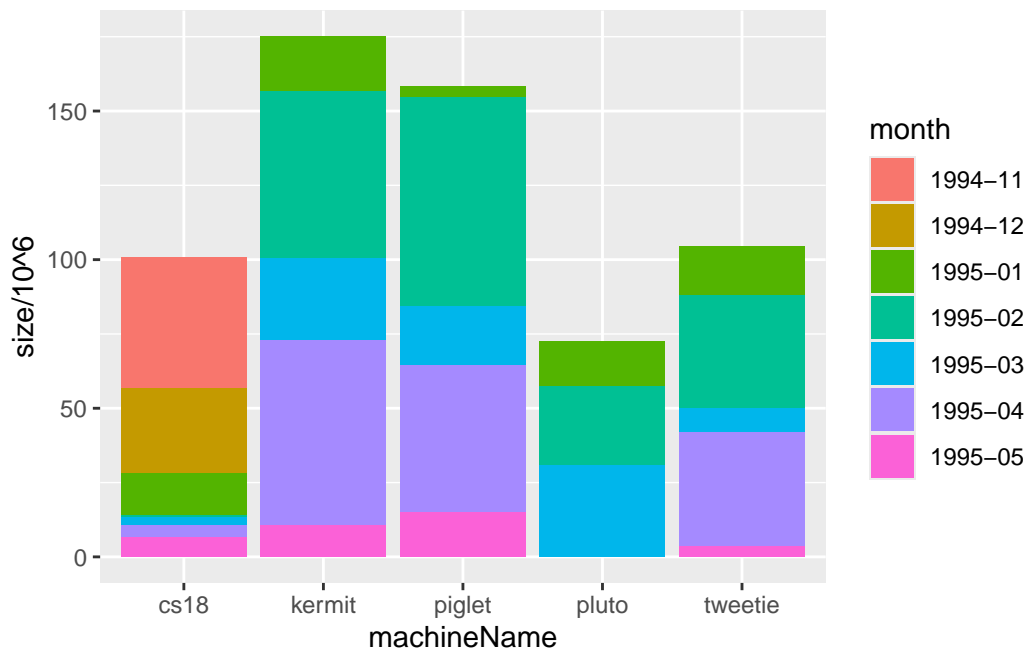
```
p + coord_flip()
```



Adding monthly download info

We can extend the graphics further by adding new *aesthetics* and/or *geoms*. Looking at the help page `?geom_col` we see that the `fill`-aesthetic will be interpreted by `geom_col()`. To see the effect of this aesthetic on that geom we simply try it out.

```
p + aes(fill = month)
```

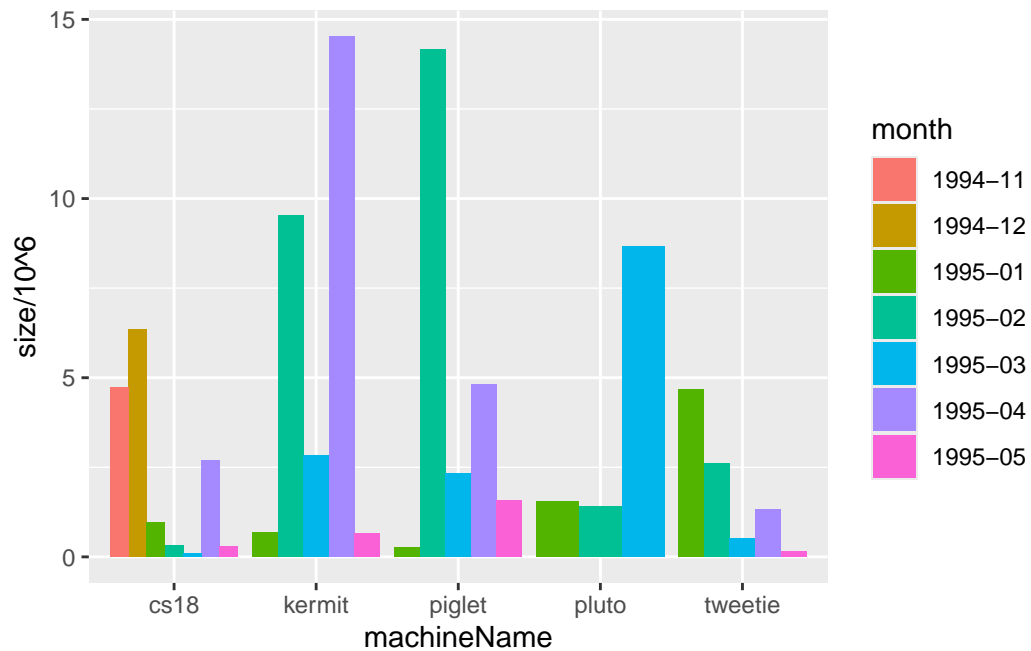


We see that the bars have the same height as before, and still visualize the total download size. But now the total download size also has been subdivided according to month. This is visualized by colors, and a legend for the interpretation of the colors is automatically added in the right panel of the plot.

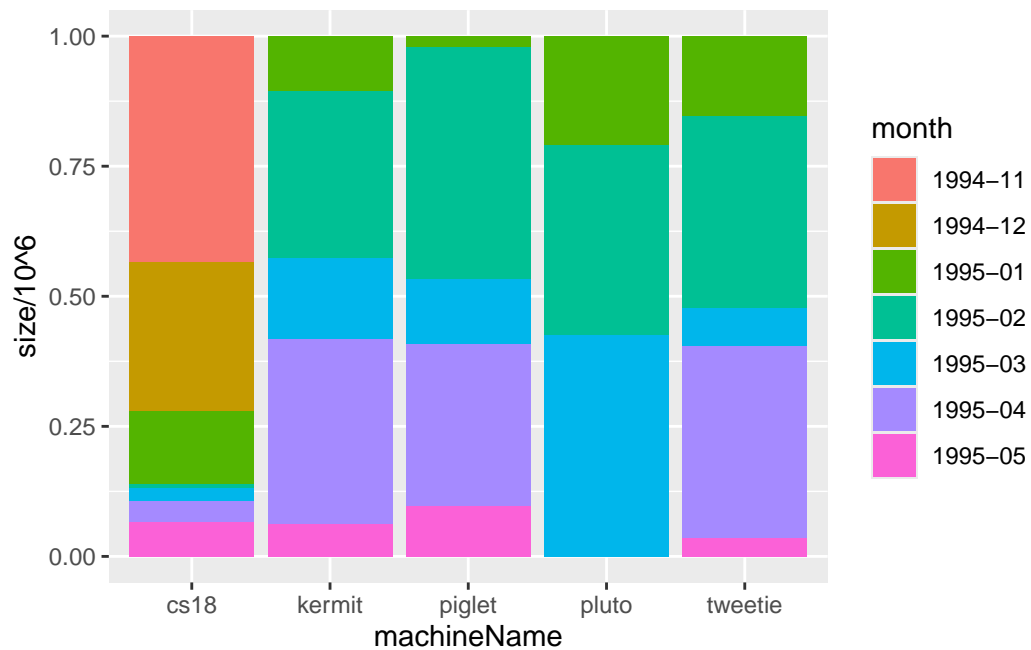
Some other bar chart options

Above we realized that setting the `fill`-aesthetic results in a subdivision of the contributions to the bars. How this is displayed may be changed by setting the `position`-option in the `geom_col()` call. Let's try it out!

```
p <- ggplot(downloads, aes(x = machineName, y = size/10^6, fill = month))
p + geom_col(position = "dodge") ## Left/first plot
```

```
p + geom_col(position = "fill") ## Right/second plot
```

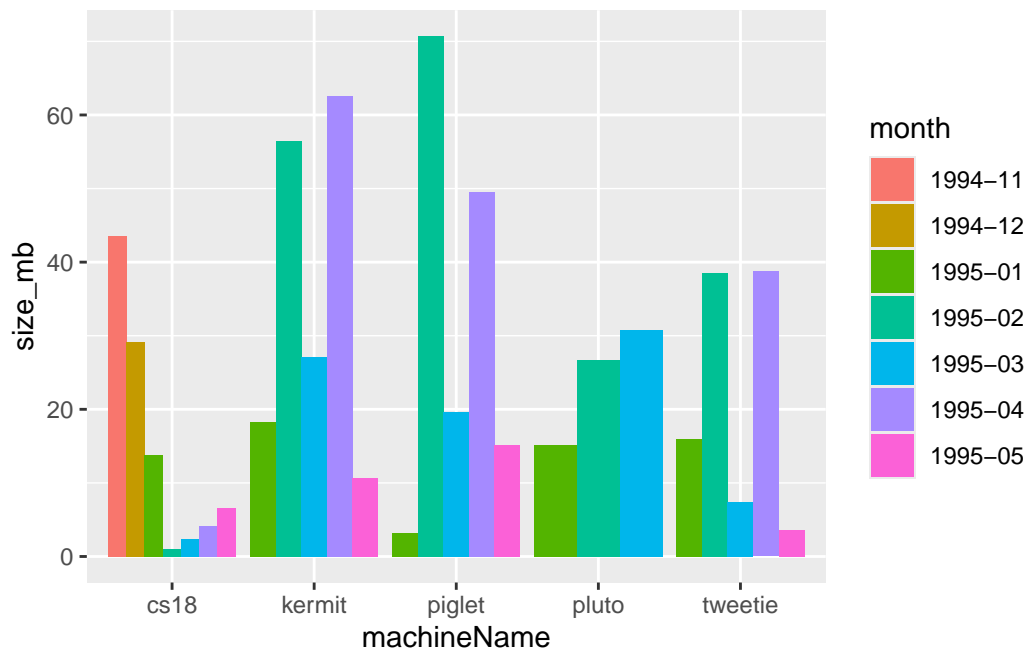


Caveat: Take a closer look at the output delivered via `position = "dodge"`. Is it doing what

you want? Or what you expected it to do? My guess is that it isn't, and that you presumably wanted and expected the output from the following:

```
downloads %>%
  group_by(machineName, month) %>%
  summarize(size_mb = sum(size/106)) %>%
  ggplot(aes(x = machineName, y=size_mb, fill=month)) + geom_col(position = "dodge")
```

```
`summarise()` has regrouped the output.
i Summaries were computed grouped by machineName and month.
i Output is grouped by machineName.
i Use `summarise(.groups = "drop_last")` to silence this message.
i Use `summarise(.by = c(machineName, month))` for per-operation grouping
  (`?dplyr::dplyr_by`) instead.
```



Very tricky question: Can you explain what is doing on here? (You might want to ask the teacher about this!)

A bar chart with ordered bars

Suppose we like the machines in the bar chart to be ordered according to increasing download size. One way to achieve this is to recode the variable `machineName` as a **factor** (in R categorical variables are called factors) with levels ordered according to increasing download size. Using the techniques presented in **Working with data in R** we generate a **tibble** containing the total download size from the five different machines, which we thereafter ordered according to total download size:

```
dl_sizes <- downloads %>%
  group_by(machineName) %>%
  summarize(size_mb = sum(size)/106) %>%
  arrange(size_mb)
dl_sizes
```

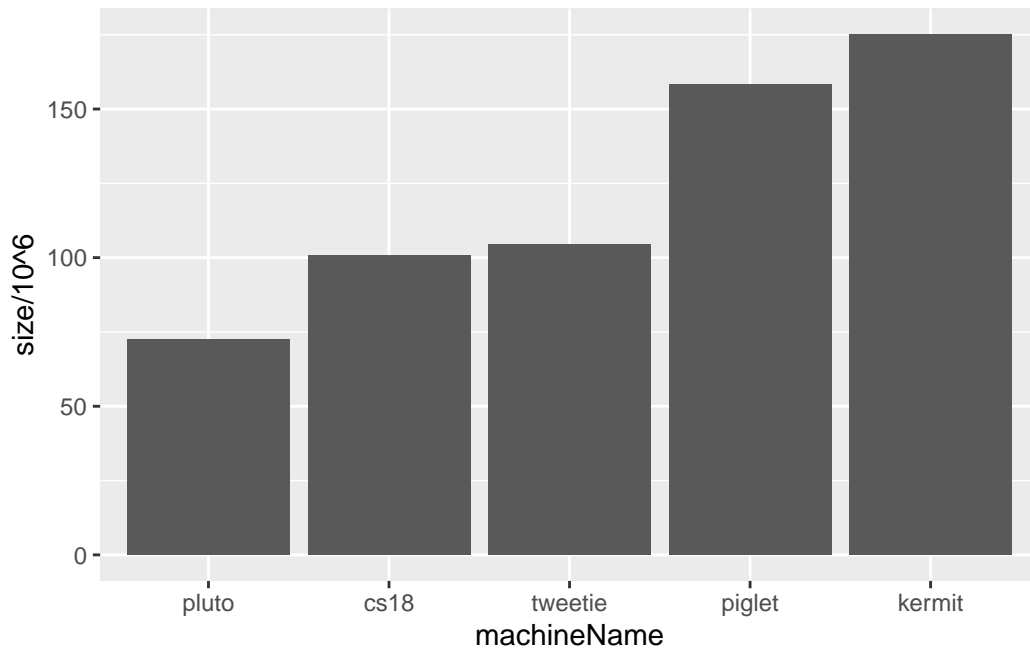
```
# A tibble: 5 x 2
  machineName size_mb
  <chr>         <dbl>
1 pluto         72.6
2 cs18          101.
3 tweetie       104.
4 piglet        158.
5 kermit        175.
```

Thereafter we recode the `machineName` variable, so that the levels appear in increasing size according to total download size:

```
downloads <- downloads %>%
  mutate(machineName = factor(machineName, levels = dl_sizes$machineName))
```

Finally, we can make the plot using the same `ggplot()` code as above. Thus, the same `ggplot()` code with a changed dataset (remember, that we made a new ordering of the levels of the variable `machineName`) will give a new plot:

```
ggplot(downloads, aes(x = machineName, y = size/106)) +
  geom_col()
```



Daily summary statistics

Next we want to visualize the number and the total size of the downloads done each date for each of the 6 machines. For later usage we also compute the cumulated number of downloads within each of the 6 machines over the dates. We do this via the following steps:

1. Using `group_by()` we group the dataset by both `machineName` and `date`.
2. Using `summarize()` we count the number and the total size of the downloads for within each machine and date.
3. Using `mutate()` we cumulate the number of downloads over the dates within the machines. We remark that `cumsum` makes the cumulative sum over the innermost grouping variable, which is `date`.

```
daily_downloads <- downloads %>%  
  group_by(machineName, date) %>%  
  summarize(dl_count = n(), size_mb = sum(size)/106) %>%  
  mutate(total_dl_count = cumsum(dl_count))
```

```
`summarise()` has regrouped the output.
i Summaries were computed grouped by machineName and date.
i Output is grouped by machineName.
i Use `summarise(.groups = "drop_last")` to silence this message.
i Use `summarise(.by = c(machineName, date))` for per-operation grouping
  (`?dplyr::dplyr_by`) instead.
```

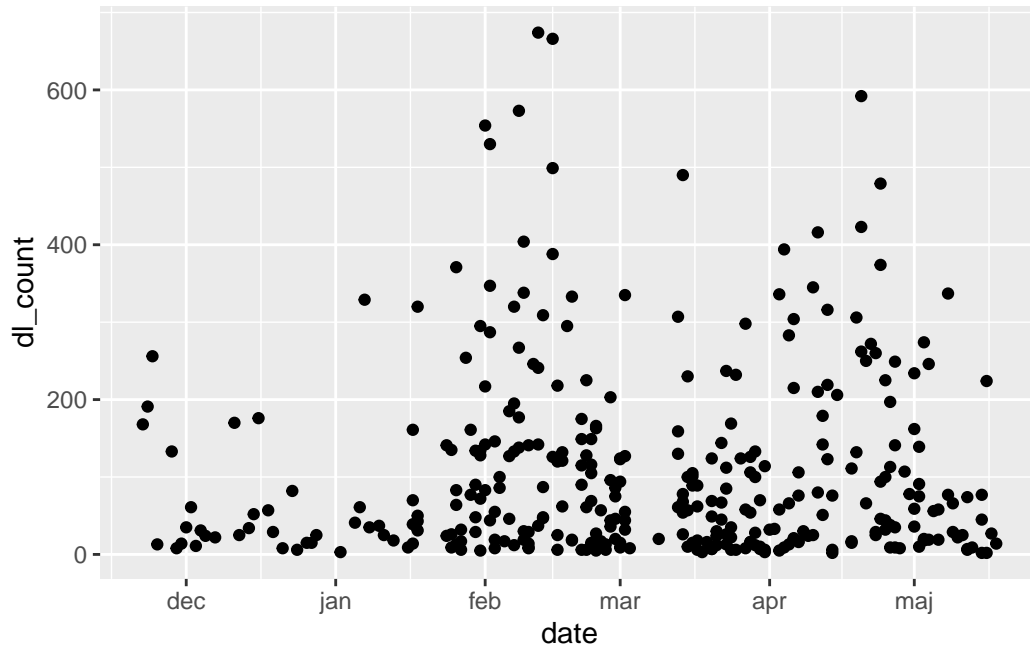
daily_downloads

```
# A tibble: 337 x 5
# Groups:   machineName [5]
  machineName date          dl_count size_mb total_dl_count
  <fct>      <dtm>          <int>   <dbl>         <int>
1 pluto      1995-01-18 00:00:00      50  0.141           50
2 pluto      1995-01-24 00:00:00     141  5.38           191
3 pluto      1995-01-25 00:00:00      26  0.0986          217
4 pluto      1995-01-26 00:00:00     371  6.44           588
5 pluto      1995-01-27 00:00:00      32  0.130           620
6 pluto      1995-01-29 00:00:00      77  0.915           697
7 pluto      1995-01-30 00:00:00      48  0.281           745
8 pluto      1995-01-31 00:00:00     128  1.71           873
9 pluto      1995-02-01 00:00:00     142  1.22          1015
10 pluto      1995-02-02 00:00:00     347  2.44          1362
# i 327 more rows
```

A simple scatter plot

To make a scatter plot we use `geom_point()`. We save the ggplot-description in the variable `p`, so that it is easy to try out different layout features on the same plot. Please note that this, of course, will overwrite the previous content of `p` (which happened to be the description of the bar charts). Thus, after executing the following R chunk, `p` will contain the description of a scatter plot.

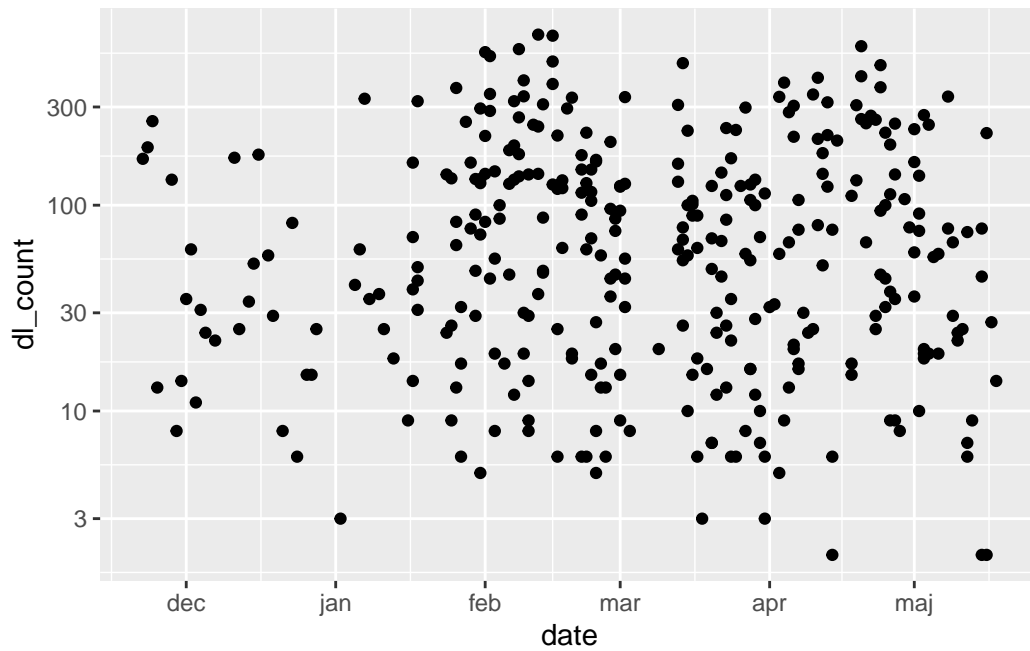
```
p <- ggplot(daily_downloads, aes(x = date, y = dl_count)) +
  geom_point()
p
```



Plotting on the log-scale

To change the y-axis to be logarithmic we add `scale_y_log10()`. Please note, that for visualizations we often prefer the base-10 logarithm (whereas statisticians often use the natural logarithm for modeling). However, for some applications the natural logarithm or the base-2 logarithm might be the preferable choice.

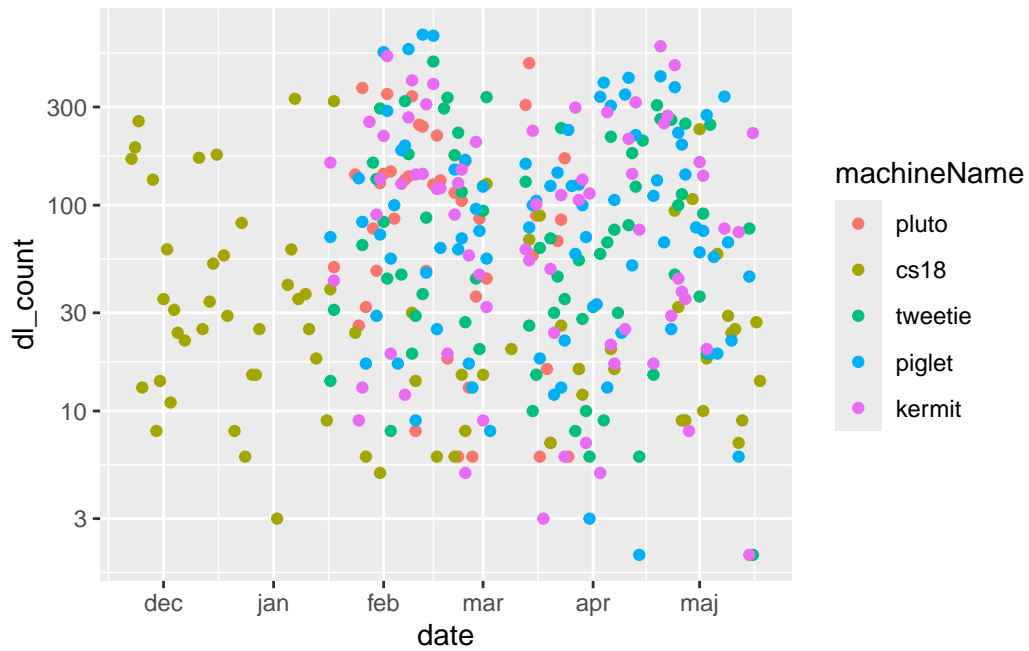
```
p <- p + scale_y_log10()
p
```



Points colored by machine

Remember, that `p` presently encodes a scatterplot, which is made using `geom_point()`. To color the points according to `machineName` we simply add this as an aesthetic.

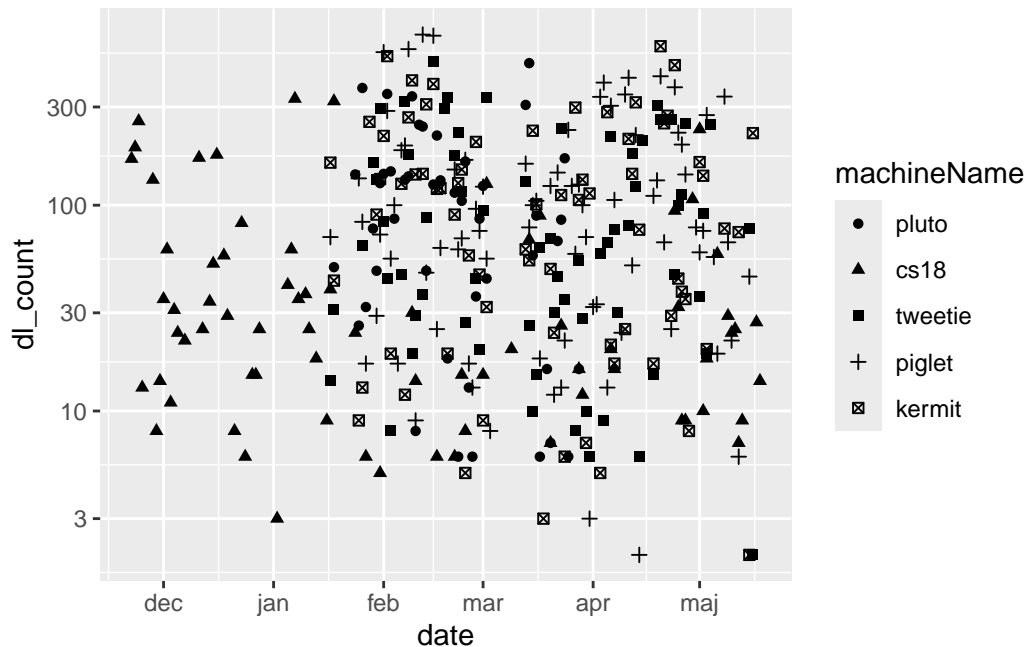
```
p + aes(color = machineName)
```



Points shaped by machine

Alternatively, we can visualize the five different machines by different plotting symbol. This is done by adding a *shape* aesthetic instead.

```
p + aes(shape = machineName)
```

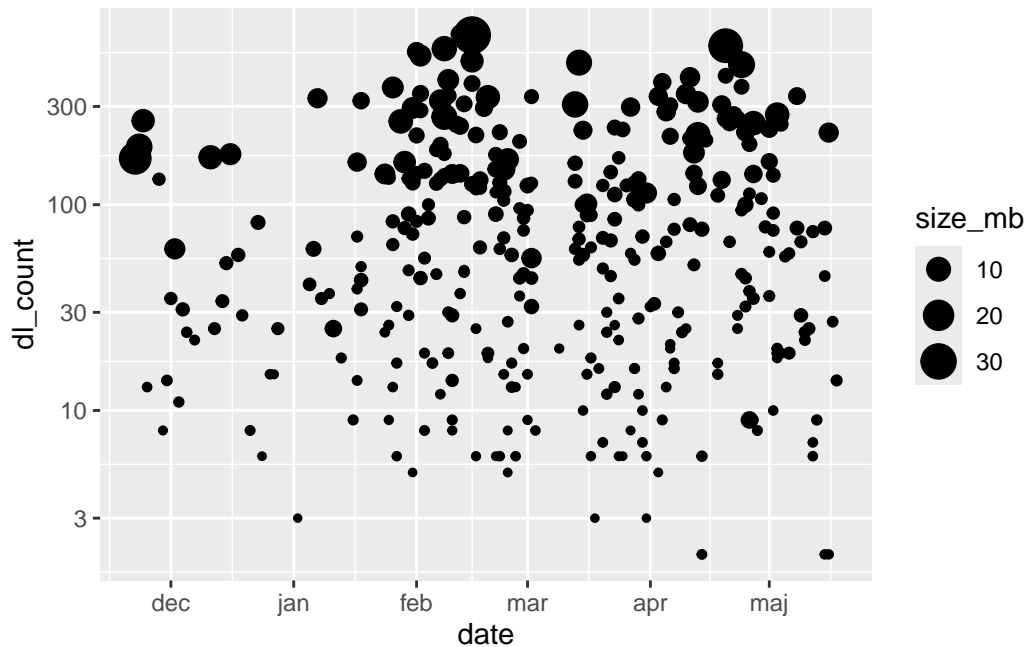
Please note that *ggplot2* only contains six different plotting symbols. If you use the *shape* aesthetic on a categorical variable with more than 6 levels, then you get the following error message:

```
> p + aes(shape = factor(size_mb))
Warning messages:
1: The shape palette can deal with a maximum of 6 discrete values because more than 6 becomes
discriminate; you have 337. Consider specifying shapes manually if you must have them.
2: Removed 331 rows containing missing values (geom_point).
```

Bubble plot

The *bubble plot* is an extension of the classical *scatter plot*. Recall, that a scatter plot displays two numerical variables via the x-axis and the y-axis. The idea of the bubble plot is to visualize a third numerical variable by letting it encode the size of the points. The best human perception of size is achieved when the area (and not the diameter, say) of the points is taken to be proportional to this third numerical variable. This is exactly what is achieved by the *size* aesthetic:

```
p + aes(size = size_mb)
```

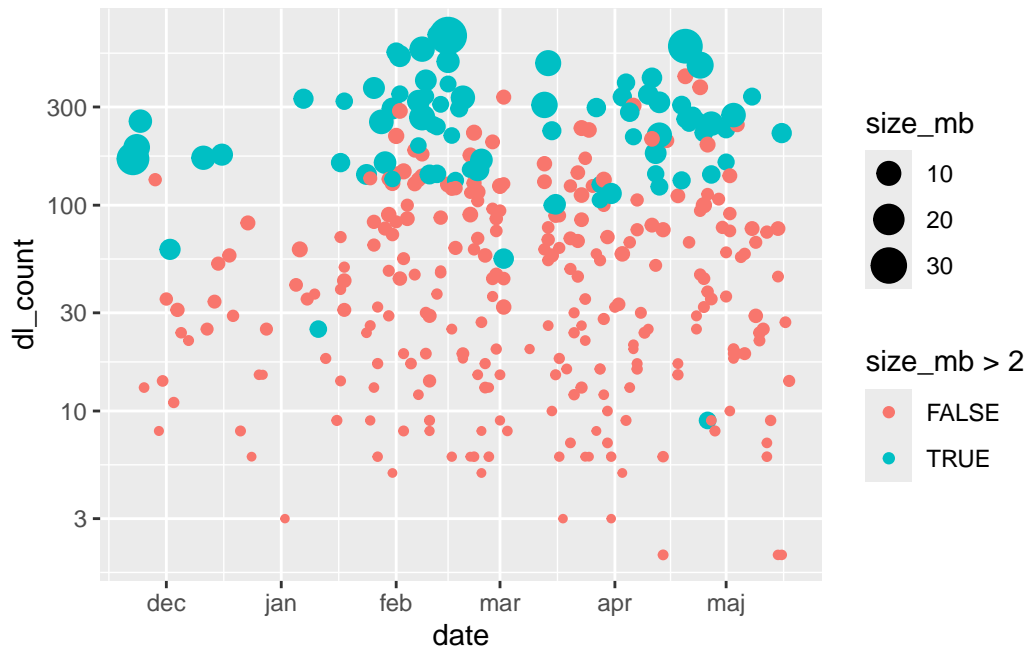


Please note, that “*size*” appearing on the left hand side is the name of the aesthetic known by **ggplot2**, whereas “*size_mb*” on the right hand side is the name of the variable inside the tibble **downloads**. It does not come as a surprise that the total download size is often large (large points) when the number of downloaded packages is large (large value on y-axis).

Points colored by download size

The ideas exemplified above may be combined. E.g. to make a stronger visualization of the daily total download size we may choose to make a combined usage of the *size* and the *color* aesthetic.

```
p + aes(size = size_mb, color = size_mb > 2)
```

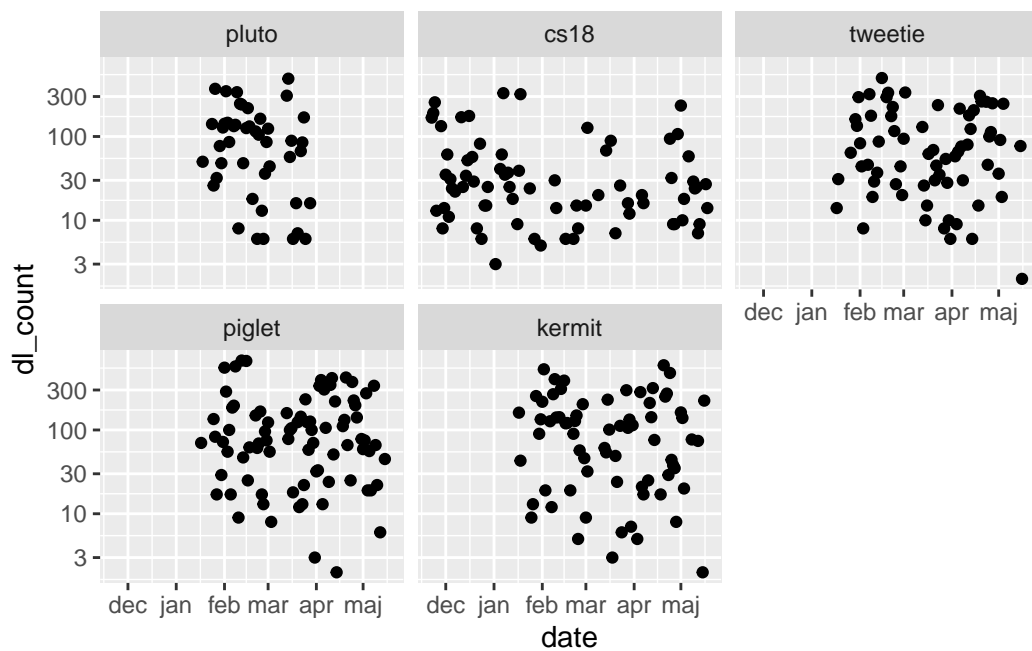


From the inserted legend in the right panel of the plot it should be clear how the colors are to be interpreted.

Faceting

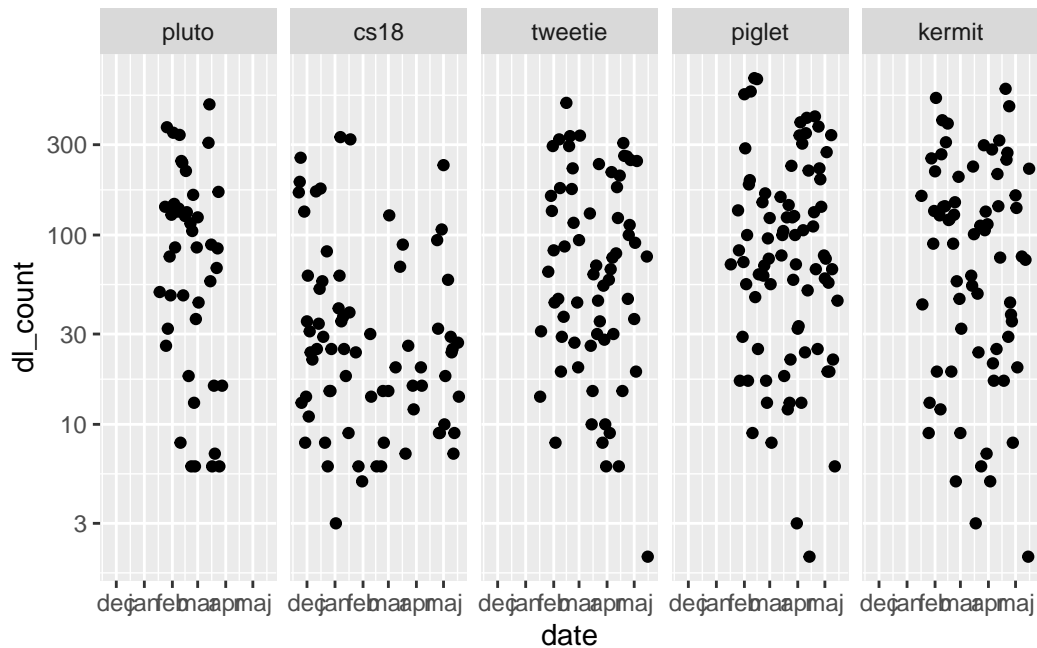
Faceting is a clever way of visualizing additional categorical variables (one, two, or even more of them) by dividing the plot into several panels (either along the rows, the columns, or both the row and the columns). Here is an example:

```
p + facet_wrap(~machineName)
```



And here is another one:

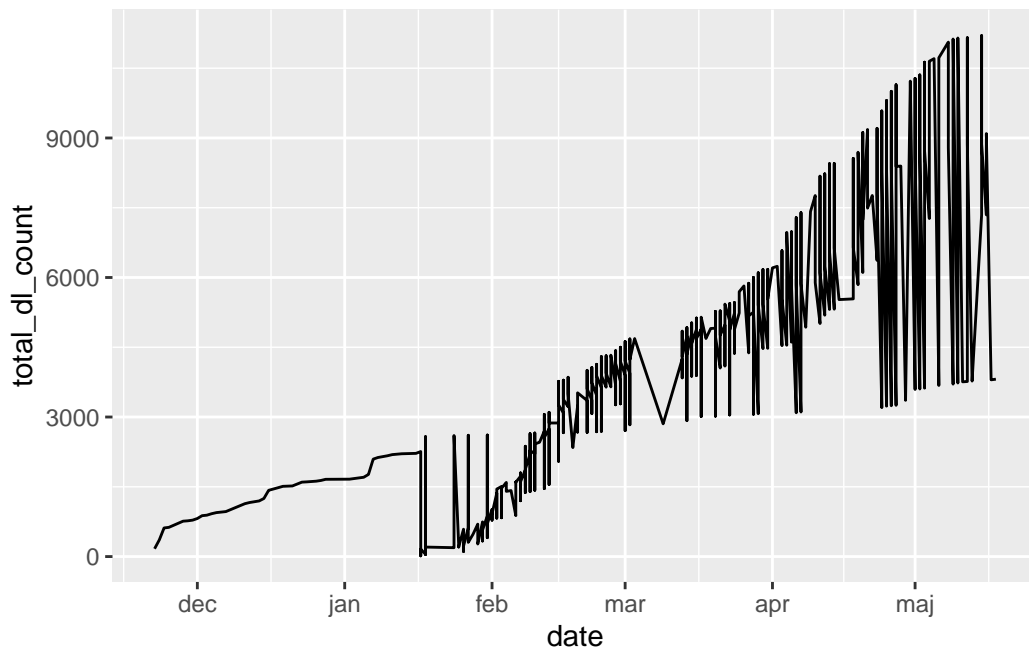
```
p + facet_grid(. ~ machineName)
```



Cumulated total download size over the dates within machines

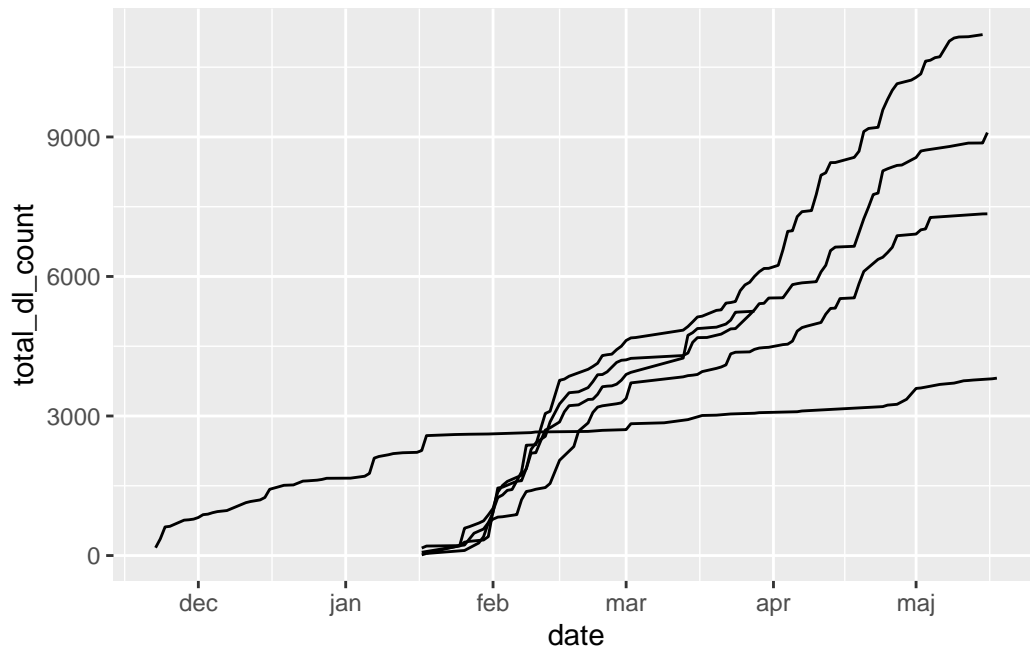
The geom called `geom_line()` is used to insert lines. In the code below you see a first attempt to write R code that visualizes the cumulated total download size over the dates.

```
ggplot(daily_downloads, aes(x = date, y = total_dl_count)) +  
  geom_line()
```



However, in the code above we forgot to make the lines *within the machines*. This may be achieved by adding a *group* aesthetic as shown in the following code.

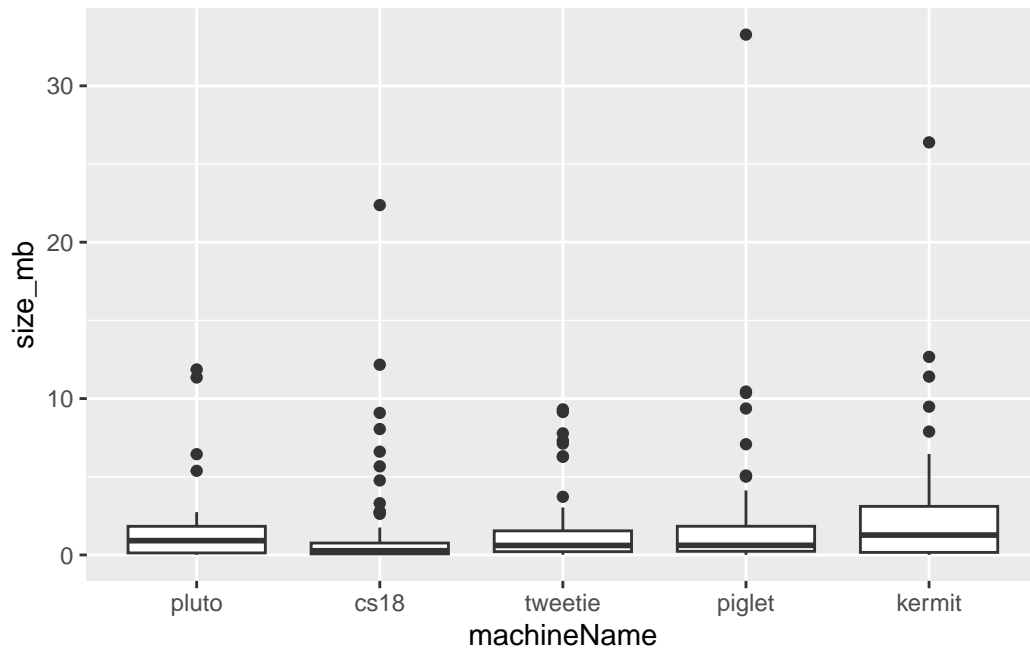
```
ggplot(daily_downloads, aes(x = date, y = total_dl_count)) +  
  geom_line(aes(group = machineName))
```



A box plot

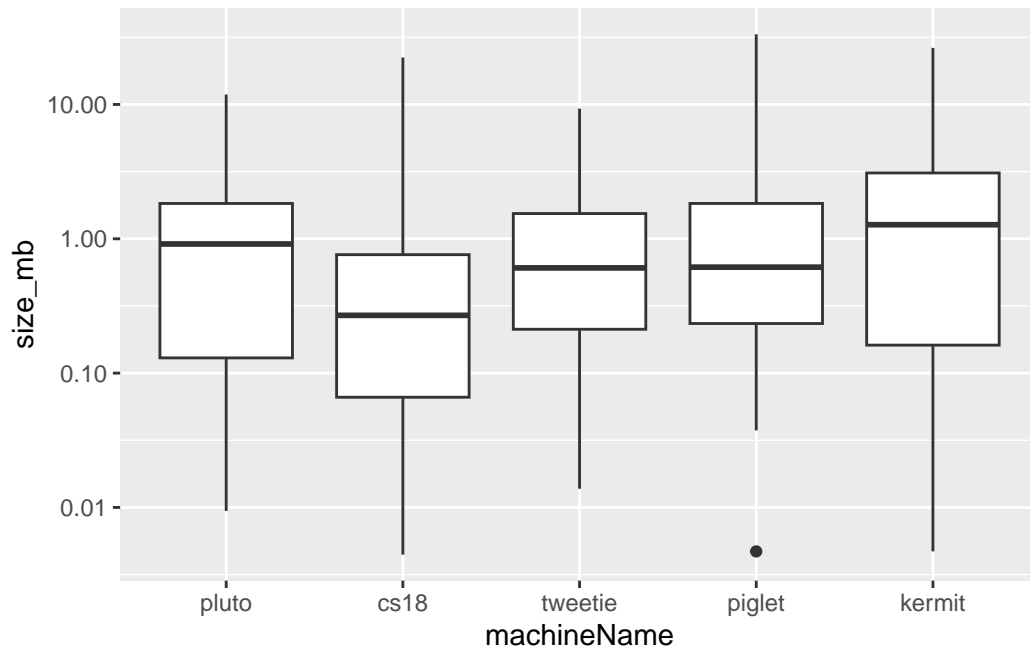
Let's also try to make a boxplot...

```
p <- ggplot(daily_downloads, aes(x = machineName, y = size_mb)) + geom_boxplot()  
p
```



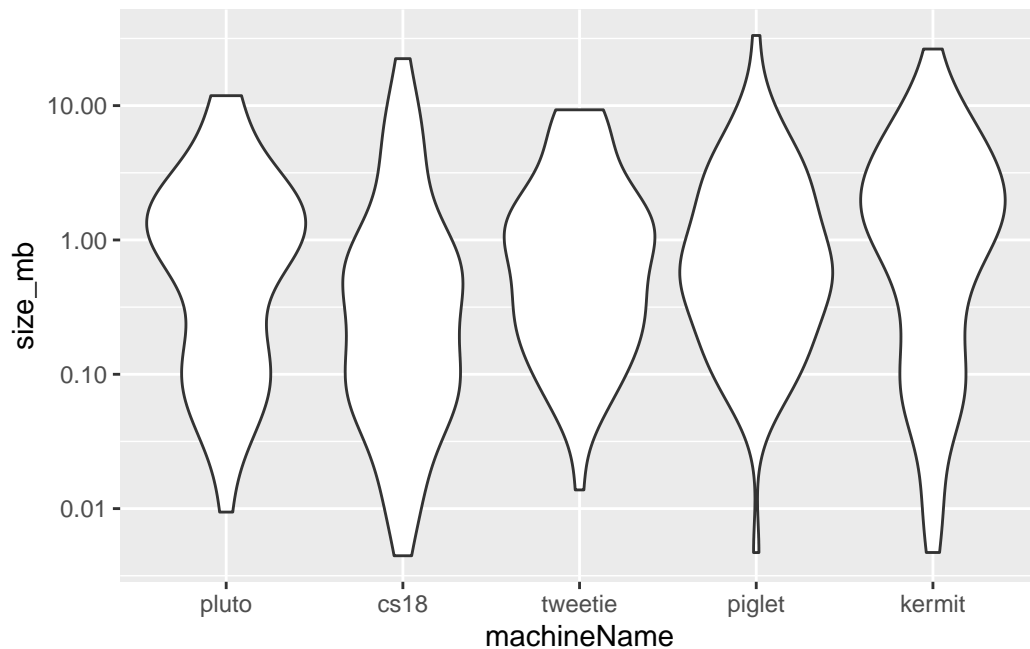
But perhaps the boxplot is more informative on the log-scale? Let's try it out!

```
p + scale_y_log10()
```



An alternative to the classical boxplot is the so-called *violin plot*, which show the full probability distribution:

```
ggplot(daily_downloads, aes(x = machineName, y = size_mb)) + geom_violin() + scale_y_log10()
```



Saving plots

ggplots easily can be printed to a graphical device. The following code saves the most recently produced plot to a pdf-file called `violin.pdf`. Please note that `ggsave()` guesses the graphics format from the file name:

```
ggsave("violin.pdf")
```

Saving 5.5 x 3.5 in image

If a plot has been saved as an object, like `p` above, then it can be saved with `ggsave()` even if is not printed on the screen. Moreover, the size of the image can be changed if needed, as this command shows:


```
ggsave("p-plot.pdf", p, width=10, height=5)
```

Conclusion

1. The `ggplot2` package is a powerful and versatile tool for making plots.
2. We think, that the generated plots are beautiful.
3. As far as we know some plots, e.g. using *faceting*, in practice are only producible in `ggplot2`.
4. Learning the syntax needed for making specific plots is a challenge. The best way (the only way!?) to learn is to practice. You may start by solving the exercise sheet. After you have gotten used to the basic ideas you can find a lot of help on the internet. We remark, that there have been several updates of the `ggplot2` package over the recent years, and some of the old entries that pop up when you google a `ggplot2`-issue might be outdated.
5. Not all things can be made in `ggplot2`! For a geometrical object to be available it need to have a syntactical description, and it needs to be implemented. Other things require computer-hacks to be made (e.g. using different orderings of categorical variables in faceted plots).
6. Many geoms make statistical computations before the actual plot is made. This can be very practical. However, occasionally the computation done isn't what you perhaps wanted. As e.g. was the case with `geom_col(position = "dodge")`. When in doubt, it can be more safe to do the needed computations manually (i.e. via `tidyverse` and `dplyr`).

End of presentation.