Data Science: Visualisations in R - Assignment 1

dr. Mariken A.C.G. van der Velden

November 1, 2019

Instructions for the tutorial

Read the *entire* document that describes the assignment for this tutorial. Follow the indicated steps. Each assignment in the tutorial contains several excercises, divided into numbers and letters (1a, 1b, 1c, 2a, 2b, etc.).

Write down your answers with **the same number-letter combination** in a separate document (e.g., Google Doc), and submit this document to Canvas in a PDF format.

Note: this is the first time that this course has ever been taught. Because of rapidly changing techniques, the course tries to keep up/ reflect those changes. As a result, the assignments may contain errors or ambiguities. If in doubt, ask for clarification during the hall lectures or tutorials. All feedback on the assignments is greatly appreciated!

About these tutorials

Welcome to the first tutorial of the course Data Science: Visualisations in R. During six sessions you will be introduced to different ways of visualising data. The purpose of this is twofold. First of all you will be able to apply the techniques for your own research. For example, you can make 'pretty graphs' for your term papers. Secondly, and equally important, the aim of the course and the tutorials is to 'get your hands dirty' and prepare you for a field of work in which data visualizations play an increasingly important role.

The explosion of digital information and increasing efforts to digitise existing information sources has produced a deluge of data, such as digitised historical news archives, literature, policy and legal documents, political debates and millions of social media messages by politicians, journalists, and citizens. Visualizing the collected information allows you to explore and to learn about its structure. Good data visualisations enable you to communicate your ideas and findings. This is not only a valuable tool for scientists. Also outside of academia, there is clearly more demand for so-called data science skills. The aim of this course is to teach you this basic knowledge.

There are also some important topics that these tutorials will not cover. During the tutorials, we will mainly focus on small-scale in-memory datasets. This is the right place to start, because you cannot tackle big data questions unless you gain first expierence with some small data. Moreover, we will also not cover anything about *Python*, *Julia*, or any other programming languages for data science. We think that *R* is a good way to start your story into the data science visualization journey. Along the way, e.g. in your master, you have more opportunities to explore other programming languages to visualize (big) data.

Installing R and R Studio

During these tutorials, we will work with the software R. The overwhelming majority of the R community nowadays works with the interface *R Studio*. This is an integrated development environment, or IDE, for R.

To download R, go to CRAN, the *comprehensive R archive network*. CRAN is composed of a set of mirror servers distributed around the world and is used to distribute R and R packages. Please use the cloud mirror: *https://cloud.r-project.org*, which automatically figures out the mirror closest to you. Note: A new version of R comes out once a year, and there are 2-3 minor releases each year. It's a good idea to update regularly, despite the hassle that comes with updating.

Once R is installed, install R Studio from *http://www.rstudio.com/download*.

When you start R Studio, you will see four regions in the interface: 1) the code editor; 2) the R console; 3) the workspace and history; and 4) plots, files, packages and help. See e.g.p.36 of the book.

You will also need to install some packages. An R *package* is a collection of functions, data and documentation that extends the capabilities of base R. Using packages is key to the successfull use of R. The majority of packages we will work with in this course aer paert of the so-called tidyverse. The packages in the tidyverse share a common philosophy of data and R programming, and are designed to work together naturally.

You can install the complete tidyverse with a single line of code:

```
#install.packages("tidyverse") ## uncomment the line by deleting the '#'
```

You can type the line of code either in R Studio's console or code editor. For the former, you just have to press enter, for the latter you press Shift/enter (Windows) or command/enter (Mac). R will download the packages from CRAN and install them onto your computer.

You will not be able to use the functions, objects, and help files in a package until you load it with `library()`. Once you have installed a package, you can load it with the `library()` function:

```
library(tidyverse)
```

This tells you that tidyverse is loading the **gpglot2**, **tibble**, **tidyr**, **readr**, **purrr**, **dplyr**, **stringr**, and **forcats** packeges. These form the core of the tidyverse. Packages in the tidyverse cchange fairly frequently. You can see if updates are available, and optionally install them, by running `tidyverse_update()`.

When you will need other packages during the tutorials, this will always be stated explicitly in the assignment.

Running R Code

In the R console of R Studio, every line of code generally contains a command or instruction for the computer to do or calculate something (formally, such commands are often called statements). Like you learned in Python during the *Computational Thinking* course. In its simplest form, you can ask R to do simple sums, such as `2+2`:

```
2+2
```

```
## [1] 4
```

(note that the line `## [1] 4` is the output of this command: a single value 4)

Throughout the tutorials, we use a consistent set of conventions to refer to code:

- Functions are in a code font and followed by parentheses, like `sum()` or `mean()`. *Note*: If you like to explore a function use `?` in front of that function, e.g. `?sum`, and the explanation appears in the lower right panel of your R Studio.
- Other R objects (like data or function arguments) are in code font, without parentheses, like `x`.
- If we want to make clear what package an object comes from, we will use the package name followed by two colons, like `dplyr::mutate()` or `ggplot2:ggplot()`. This is also valid R code.

Data Visualization with ggplot2

During the course, we will learn you to visualize data using **ggplot2**. R has several systems for making graphs, but **ggplot2** is one of the most elegant and most versatile. **ggplot2** implements the *grammar of graphics*, a coherent systems for describing and building graphs. If you'd like to learn more about the theoretical underpinnings of **ggplot2**, I would recommend reading "A Layered Grammar of Graphics" (*http://vita.had.co.nz/papers/layered-grammar.pdf*)

The **layered grammar of graphics** tells us that a statistical graphic is a **mapping from data to aesthetic attributes** (colour, shape, size) **of geometric objects** (points, lines bars). In a formula, this looks like the following:

`Data + Aesthetic mappings + Layers (Geometric objects, Stats ) + Scales + Coordinate system + Faceting + Theme`

Key Components of that formula are: * **Data** = a data frame * A set of **aesthetic mappings** between variables in the data and visual properties (e.g. horizontal and vertical position, size, color and shape) * And at least one layer (describes how to render the observations; usually created with a **geom** function, e.g. `geom_point()`

Assignment 1a: Try a simple scatter plot

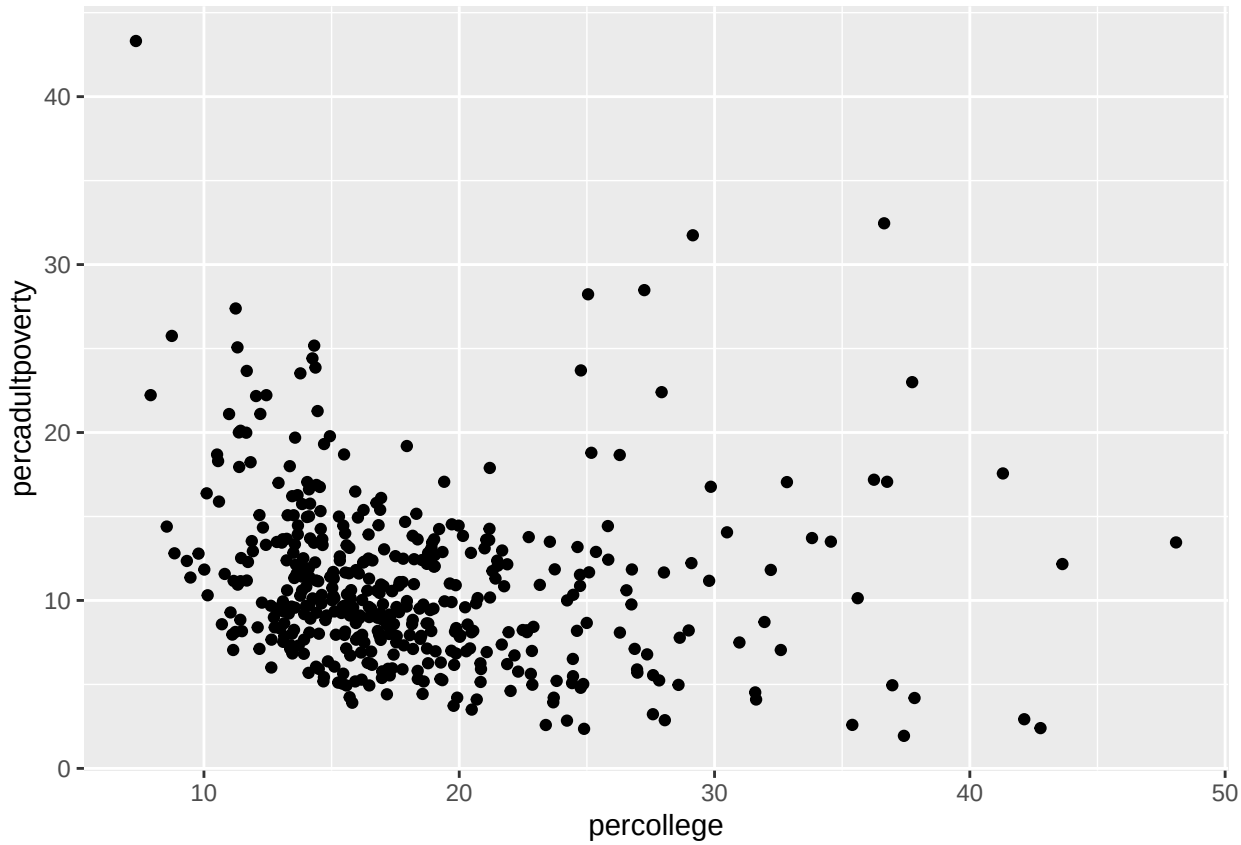
In this assignment, we will practice to get a little familiar with the above described key components of the grammar of graphics.

If you'd like to know whether areas in which more people with a college degree live are more likely to have low levels of poverty, we can use the `midwest` data to make a simple scatter plot, using the following key components:

- **Data** = `midwest`, you can inspect the data with either `head(midwest)` or `str(midwest)`
- **Aesthetic mappings** = percent college educated (variable `percollege` in `midwest`) mapped to x position, percent adult poverty (variable `percadultpoverty` in `midwest`) mapped to y position
- **Layer** = `geom_points()`

This leads to the following code:

```
ggplot(midwest, aes(x = percollege, y = percadultpoverty)) +
geom_point()
```




```
## There are two other ways to write the code for the same outcome.
#ggplot(midwest) +
#geom_point(aes(x = percollege, y = percadultpoverty))

#ggplot() +
#geom_point(data = midwest, aes(x = percollege, y = percadultpoverty))
```

1a) Replicate the codes and check whether you get three times the same graph. Pay attention, when `ggplot()` *is empty, you need to specify which data you use, using the* `data =` *statement in the layer* `geom_point()`.

1b) Explore the options of `geom_point()` *by adding a color to the points based on the values of* `percadultpoverty`. *Add only the used code into your assignment.*

1c) While the different colors for percent adult poverty (variable `percadultpoverty`*) give us un overview of the severity of adult poverty in the entire region the Midwest in the US. We might also want to know whether some states are worse of than others. Explore the options of* `geom_point()` *by adding a color to the points based on the values of* `state`. *Add only the used code into your assignment.*

1d) To combine both the variation in state and severity of adult povert, add a color for `percadultpoverty` *to the figure and shapes for the different states. Add the code to your assignment.*

1e) The figure created in 1e might become hard to read. Therefore, instead of using shapes for states, you can use `facet_grid` *to split the graph on state. Explore this command. You can also try out different fixed colors for* `percadultpoverty` *looking at http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf. Pay attention, when you want to fix a color, it needs to be outside the* `aes` *statement and in between "...". Add the code and the figure to your assignment. You can save the plot either by clicking on the "Export" button in the lower right panel or use* `ggsave()`.

Layers

There are many different layers one could choose from to visualize data. Next to a visual display, layers serve the purpose of giving a statistical summary of the data at hand. Furthermore, layers could be used to add additional metadata (e.g. annotations, maps, etc.). The geoms in `ggplot2` define the shape of the elements on the plot. The basic shapes one can choose from are points, lines, polygons, bars, texts; i.e. `geom_point()`, `geom_line()`, `geom_polygon()`, `geom_bar()`, and `geom_text()`. For statistical summaries of the data, one could use `geom_histogram()`, `geom_smooth()`, or `geom_density()`. For today's tutorial, we will work with the basic layers. Next week, we will dive into the statistical summary layers.

Assignment 2: Apply different basic layers

2a) create a data set using the following code

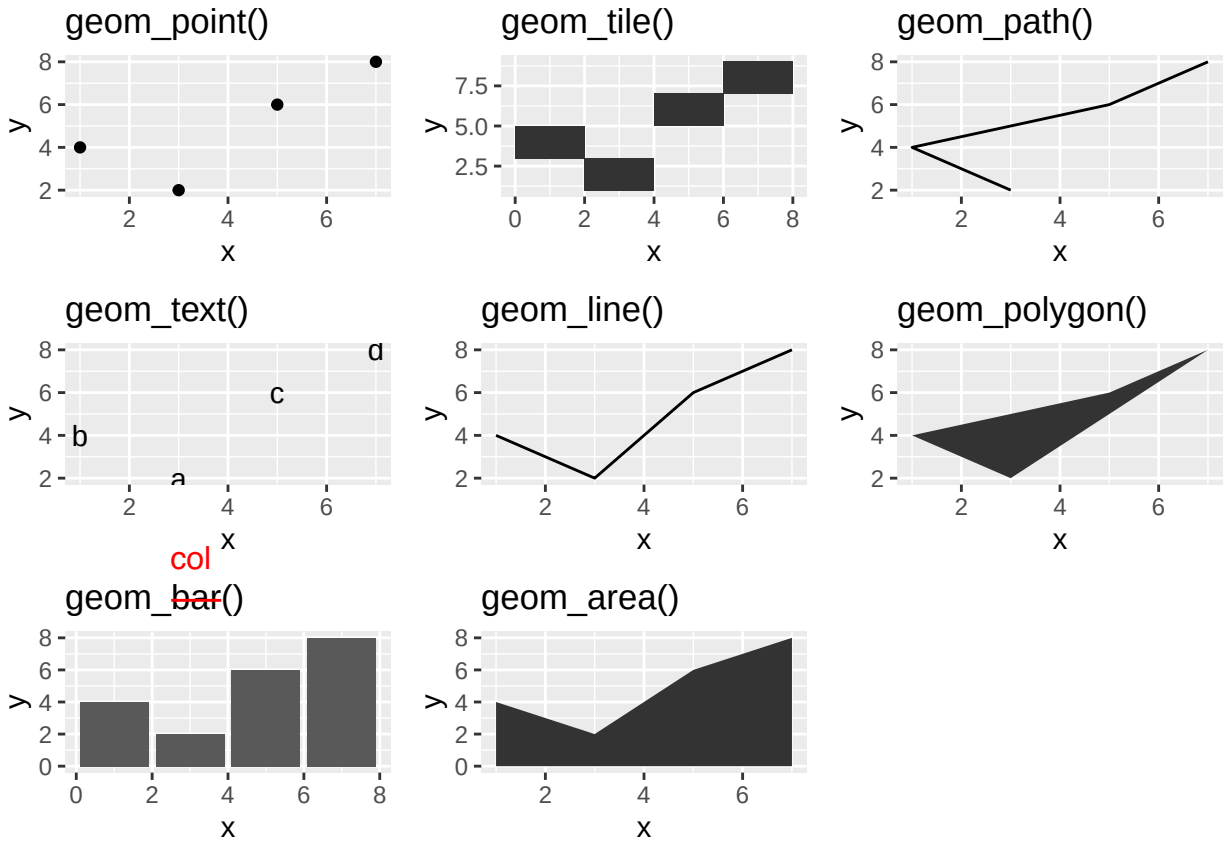
```
my_first_data_frame = data.frame(x = c(3,1,5,7),
                                 y = c(2,4,6,8),
                                 label = c("a", "b", "c", "d"))

my_first_data_frame
```

```
##   x y label
## 1 3 2     a
## 2 1 4     b
## 3 5 6     c
## 4 7 8     d
```

Do note that in R, in order to store an object, you always have to name it. The name itself doesn't matter, you can pick any name!

2b) The figure below uses `my_first_data_frame` *to display different layers. Replicate each figure. To add a title, use* `ggtitle()`*.To have all the graphs combined, have a look at http://www.cookbook-r.com/Graphs/Multiple_graphs_on_one_page_(ggplot2)/. Copy the code and add the combined graphs into your assignment.*

For a little help download the Data Visualization Cheat Sheet here: https://www.rstudio.com/wp-content/uploads/2016/11/ggplot2-cheatsheet-2.1.pdf



2c) Think of types of data for which each plot would be a good visualization.

Good Luck

The deadline for the assignment is **November 7, 10am**!

References

Healy, K. (2018). *Data visualization: a practical introduction*. Princeton University Press.

Wickham, H., & Grolemund, G. (2016). *R for data science: import, tidy, transform, visualize, and model data*. " O'Reilly Media, Inc."