Introduction to R, week 4

Indrek Seppo

Sept 24, 2021

HITSA Preparation of this course was supported by HITSA – Estonian Information Technology Foundation for Education.

Preparations

Lets load in tidyverse libraries and piaac data again (if you don't have it in your environment already), lets also read in some random data about animals:

```
library(tidyverse)
piaac <- read.csv("http://www.ut.ee/~iseppo/piaacENG.csv")
animals <- read.csv("http://www.ut.ee/~iseppo/animals.csv")
```

Save the animals and piaac dataset into our working directory for future use:

```
write_csv(animals, file="animals.csv")
write_csv(piaac, file="piaacENG.csv")
```

R markdown

Lets take a peek at R markdown. Create a new R markdown document by File -> New File -> R Markdown. Lets start with an html output at first. What you see, is an example document, that will give you a quick overview what you can do.

Markdown is a lightweight document language, which can be translated to different types of outputs – html, pdf, doc etc[1]. While there is no single markdown standard, it is pretty similar in wherever you encounter it, it is by no way R-specific.

[1] Some of the syntax can be output-specific though – you would generate a page break differently when opting for pdf-output than doc.

Markdown provides some simple syntax, for example:

```
# Heading 1
## Heading 2
### Heading 3
*italic text*
**bold text**
```

Bullet list:

```
* something
* other
* third
```

Numbered list:

```
1. something
2. other
3. third
```

You need to remember to add two linebreaks (so creating one empty line) to actually end a paragraph, start a list etc.

To add a footnote, use `^[myfootnote]`.

And you do not actually need much more to create a well-structured, readable document.

Lets try it!

To insert a code chunk press `Cntrl + Alt + i`, or use the green Insert New Codechunk pictogram from the top.

What you need to know in using R Markdown is, that it kind of runs in its own environment. Thus the packages you have loaded in your RStudio session will have to be loaded again, the data will have to be loaded again etc. You can run everything in the document until the current line by either `Cntrl+ALT P` or by clicking `Run all previous chunks` pictogram on a code chunk.

The `modify chunks options` is a particularly useful thing, you rarely need more (but at times you will). You usually do not want to show messages and warnings in your final document etc.

To add tables, try `kable()` from `knitr` package. This will produce tables in the markdown format, so would work in pdf, doc and html without any changes.

```
library(tidyverse)
library(knitr)
animals <- read_csv("animals.csv")
kable(animals)
```

But sometimes R will already produce readymade html or latex-code. To include this you should give a parameter `results=asis` to your code. For example – install and load the library **xtable**:

```
library(xtable)
options(xtable.comment = FALSE)
print(xtable(animals),type="html", include.rownames=FALSE)
```

For figures you have additional arguments – `fig.cap=`, `fig.height=`, `out.height=` etc. Lets try to add a figure – create a new code chunk and write the following inside it:

```
piaac <- read_csv("piaacENG.csv")
ggplot(piaac, aes(x=age, y=Income))+
  geom_smooth()
```

If you want to include pregenerated image, you can do it this way:

`![Caption for the picture.](/path/to/image.png)`

You can precompute some values in code chunks and use them in the text.

RStudio provides some additional RMarkdown templates. Check out the packages `tufte`, `rticles` and `rmdformats`. Some additional infor-

mation is available here: https://blog.rstudio.org/2016/03/21/r-markdown-custom-formats/. I am personally a fan of tufte-style.

And if you ever want to write a book with all the citations, cross-references etc, check out https://bookdown.org/yihui/bookdown/.

What we know about ggplot

A typical plot in the ggplot language looks something like this:

```
ggplot(data=mydataset, aes(x=firstvariable, y=secondvariable))+
  geom_something(aes(color=groupingvariable))+
  geom_otherthing(size=4)+
  facet_wrap(~othergroupingvariable)
```

If you want something at the graph to be connected to your data, you will need to put it into the **aes()**-call. If you want to tell it yourself (like the `size=4` here), it has to be outside of the `aes()`-call.

Your turn:

Lets use the `piaac` dataset[2] again. Lets remove everyone who does not know their education level:

[2] It was at http://www.ut.ee/~iseppo/piaacENG.csv.

```
piaac <- filter(piaac, !is.na(Education))
```

Then everyone who does not know if they have children or not:

```
piaac <- filter(piaac, !is.na(children))
```

1) I want you to do the graph on the side. Use `geom_bar()` (it only requires an x-aesthetic – you can see what it is from the graph) and facet by `Education`.

2) Now, lets plot some relationships in the data. This is geom_smooth() that I am using here. The variable names should be guessable from the graph.

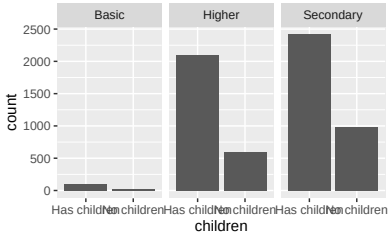


Figure 1: Solution to the first exercise

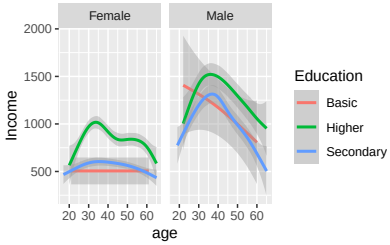


Figure 2: Second exercise

Reordering factor levels

We would probably want to see the education levels in correct order. This is easy:

```
library(forcats)
piaac$Education <- fct_relevel(piaac$Education, "Basic", "Secondary", "Higher")
```

Creating the graph again would now be in logical order:

```
ggplot(piaac, aes(x=children))+
  geom_bar()+
  facet_wrap(~Education)
```

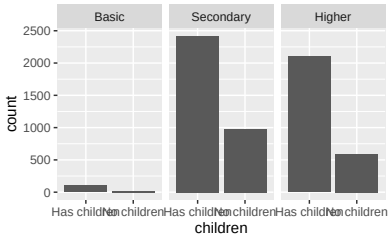


Figure 3: First exercise with correct ordering

This `forcats` package is the one-stop-solution for everything factor-related. All the commands start with *fct_*- just look at what it has to offer by typing *fct_* and pressing tabulator (or go to the packages pane and click on forcats). For example – would we like to change the factor levels from Low to "Low education" we would do it the following way:

```
piaac$Education<- fct_recode(piaac$Education, "Low education"="Basic", "Highschool"="Secondary")
levels(piaac$Education)
```

```
## [1] "Low education" "Highschool"    "Higher"
```

Wide and long data

Look at how the data from Eurostat (or World bank etc) looks like

```
library(eurostat)
silc <- get_eurostat("ilc_mdho05")
head(silc, n=3)
tmp<-label_eurostat(silc)
View(tmp)
```

This is called the long format. In practice we usually find data in the wide format:

```
##   animal length width age
## 1   bear     10     7   3
## 2    cat      4     3   5
```

But R likes to have data in the long format (or tidy data as it is called), where every row is a single measurement. The same data in the long format would be:

```
## # A tibble: 6 x 3
##   animal name   value
##   <chr>  <chr>  <dbl>
## 1 bear   length    10
## 2 bear   width      7
## 3 bear   age        3
## 4 cat    length     4
## 5 cat    width      3
## 6 cat    age        5
```

The most modern way to convert data between wide and long format (since couple of months) is: `pivot_wider()` and `pivot_longer()` in the package `tidyr`. Lets try it with the `animals` dataset:

```
animals <- read.csv("http://www.ut.ee/~iseppo/animals.csv")
```

Take a look at the help file of `pivot_longer()`. This is a rather powerful function with a nr of arguments, but the main ones are: `data` – which dataset to convert, `cols` – which columns to pivot into longer format, `names_to` – what should be the name of the new variable containing these values, `values_to` – what should be the name of the new variable containing the values which are currently inside the matrix.

Lets try it:

```
animals_long <- pivot_longer(animals, cols=c("length", "width", "age"),
                             names_to="measurement", values_to="value")
```

If you have data in long format, then you can widen it with `pivot_wider()`. Again, lets look at the help file: `?pivot_wider()`. It requires the data frame on which to operate, then the `id_cols` – these need to identify each observation uniquely (you will notice if there are problems with this), by default you do not have to put anything there, all of the other columns you are not touching, will just be left in place. What is important for our simple use cases are `names_from` and `values_from` parameters stating which columns should be moved to variable names and how to populate the data matrix below it.

Lets take a look:

```
pivot_wider(data=animals_long, names_from="measurement", values_from="value")
```

```
## # A tibble: 2 x 4
##   animal length width   age
##   <chr>   <int> <int> <int>
## 1 bear       10     7     3
## 2 cat         4     3     5
```

And it is done!

There is also a nice vignette about these functions available here: https://tidyr.tidyverse.org/articles/pivot.html

Your turn:

- Create a new dataset from the animals dataframe, so that it would look like this (note which columns are pivoted to longer format):

```
##   animal age measurement value
## 1   bear   3      length    10
```

```
## 2    cat   5      length     4
## 3   bear   3       width     7
## 4    cat   5       width     3
```

- Reload the `gdp` dataset from `http://www.ut.ee/~iseppo/gdpestimate.csv`. Convert the variable `date` to be a date variable using package lubridate (remember – it had `ymd()`, `dmy()` etc in it, so look at how the dates look in this dataset and convert them using appropriate function). It has variables `firstEstimate` and `latestEstimate`, Create a new dataset, so that we would have a measurement-value pair in this – so one column for `measurement` (for each row there would then be a value of either `firstEstimate` or `latestEstimate`) and another for `value` (the numerical estimate in question).

When plotting it, we could now use the measurement variable to automatically create the following plots:

This was more or less possible without converting the data too, but this would not have been:

- Try to recreate the faceted graph.

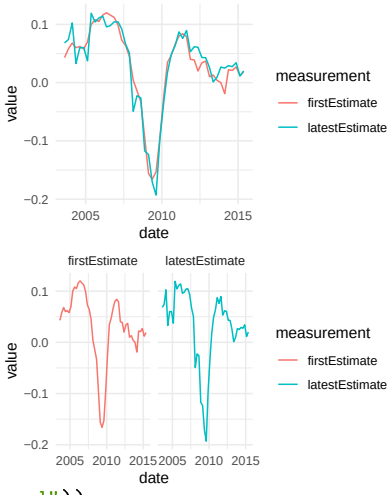


Figure 4: This what you need to recreate

Lets download gdp data from World bank and choose three countries (You can use whichever countries you want):

```
gdp.pc <- read.csv("http://www.ut.ee/~iseppo/gdppc.csv")
gdp.pca <- subset(gdp.pc, country%in%c("World", "Estonia", "Finland"))
```

1) see if you can spread this data (`gdp.pca`) so that there will be columns representing years and rows reperesenting countries. The value should be the numerical value of the indicator. Oh, and first remove the `date_ct`-variable by assigning NULL to it. Note: the `date` variable consists of the year number – this is what you want to become column name now.

dplyr – a grammar of data manipulation

dplyr, again authored by Hadley Wickham, is pretty much all you need in your initial data manipulation – this is the step that actually takes most of the time in most of the data projects, and it makes it much easier. Sometimes all you need at all.

We will look at six basic verbs in dplyr:

- select(): for subsetting variables (columns)
- filter(): for subsetting rows
- mutate(): add new columns
- group_by(): define groups

- summarise(): create summary statistics by group
- arrange(): re-order the rows

"And then" operator %>% aka piping in R

%>% is not in fact an operator from **dplyr**, but comes from **magrittr**, but **dplyr** (as every other Hadleys recent package) supports and promotes it.

Take the **dplyr** command **top_n()**, which finds x rows from the dataset where some variable has the highest values. To find the five people with highest wages in the dataset[3]:

[3] This is actually not so great idea even with the public use datasets – exceptional datapoints should be examined, but privacy matters and our professional ethics should only let us do what is needed for an analysis.

```
library(dplyr)
top_n(piaac, 5, Income)
```

But we can also rewrite it in this way:

```
piaac %>%
  top_n(5, Income)
```

Seems pointless at first – why would anyone convert a nice oneliner to a longer expression, but we will see that it can get elegant when we add yet more lines.

Selection of variables, **select()**

select() helps us to select the variables from the data frame. We used `dataframe[c("variable1", "variable2")]` before, in the dplyr we can do it in the following way:

```
piaac$logincome<-log(piaac$Income)
piaac.small <- piaac %>%
  select(gender, logincome, Education)
head(piaac.small, n=3)
```

```
##   gender logincome  Education
## 1   Male        NA Highschool
## 2 Female  6.619678 Highschool
## 3 Female        NA Highschool
```

Doesn't seem like much, but **select()** offers us a plethora of additional flexibility – take a look at the *Data Wrangling cheat sheet*, *Subset Variables (Columns)*. We can, for example select all the variables from one to another:

```
piaac.small <- piaac %>%
  select(children:Income)
head(piaac.small, n=3)
```

Or all the variables that start with “e”, and then add “seqid”:

```
piaac.small <- piaac %>%
  select(starts_with("e"), seqid)
head(piaac.small, n=3)
```

Your turn:

- Create a new dataframe, containing all the variables that start with “empl” and also all the variables which end with “n”. You will find how to filter out the variables ending with something from the cheat sheet.

summarize()

Let me introduce you to the function **summarize()**. It, well, summarizes data and creates a data frame of these summaries.

```
piaac %>%
  summarize(averagewage=mean(Income, na.rm=TRUE), n=n())
```

```
##   averagewage    n
## 1    899.6363 6202
```

Note that we can add multiple new summarizing variables, separated by a comma.

```
piaac %>%
  summarize(meanincome=mean(Income, na.rm=TRUE), medianincome=median(Income, na.rm=TRUE))
```

```
##   meanincome medianincome
## 1   899.6363     725.0342
```

Your turn:

- Use summarize to find max and min income (Income) for the people in the sample.

Grouping

Nothing very special until now. Lets add grouping. This is done with **group_by()** function.

Look at the following command:

```
table1 <- piaac %>%
  group_by(gender, Education) %>%
  summarize(meanwage=mean(Income, na.rm=TRUE), totalwage=sum(Income, na.rm=TRUE))
table1
```

This is how we rock!

filter() – subsetting rows

In the base-R we used [] or the `subset()`-function to subset datasets. In the dplyr there is a `filter()` function, which works very much like `subset()`:

```
newtable <- piaac %>%
  filter(Income>700)
summary(newtable)
```

We can add the conditions with & (logical AND) and | (logical OR) operators (actually a regular comma - , – works as logical AND, all the conditions have to be met):

```
newtable <- piaac %>%
 filter(Income > 700, gender=="Male")
summary(newtable)
```

We can now add this all together

Your turn:
Read the following carefully. This is written in normal language, so it is not a step-by-step guide. You'll have to put some thought into what to do first, what after that etc.

- Create a new dataframe containing only people of the opposite sex, whose Health is either excellent or very good (check how these are spelled in the data – capital letters matter) and numeracy score (Numeracy) is at least 300. What is their average monthly income (Income)?

arrange()

Or look at **arrange()**. It is a very simple function - it arranges the dataset according to some variable (in ascending order):

```
gdp3 <- gdp.pc %>%
  filter(date=="2015")%>%
  arrange(value) %>%
```

```
  select(value, country) #we will only save these variables

head(gdp3)
```

To do it the other way around, use **desc()** (descending)

```
gdp3<-gdp.pc %>%
  filter(date=="2015")%>%
  arrange(desc(value)) %>%
  select(value, country)

head(gdp3, n=4)
```

you can add many variables into `arrange()`, the dataset will be arranged according to first one, if they are equal, then second one etc.

Creating new variables: **mutate()**

mutate() helps to add new variables. In its simplest form:

```
piaac <- piaac %>%
  mutate(sumOfSkills=Numeracy+Literacy)
```

But there are a lot of additional features (take a look at *Make New Variables* part of *Data Wrangling* cheatsheet. We can find in which percentile someone is:

```
piaac <- piaac %>%
  mutate(numeracypc=percent_rank(Numeracy))
```

Combining mutate with **group_by()**, we can find the percentiles inside the group, or we can divide someones wage by groups average etc

```
newtable <- piaac2 %>%
  group_by(gender)%>%
  mutate(relincome = Income/mean(Income))
```

You will now have relative income to towards the total mean, but to group mean.

Your turn:

- find the average wage (dont forget to ignore the NA-s) and numeracy scores by gender and studyarea (variable `studyarea`) for those whose age is over 30, arrange it by the numeracy score, write it to a new data object and write this data object out as a .csv file (`readr::write_csv` in actually part of the tidyverse, so you can just add it as a final row using '%>%'.

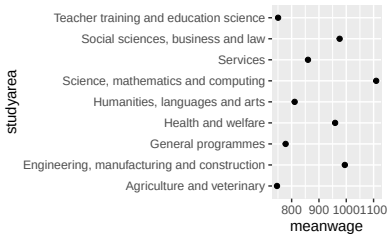
- add a new variable to piaac dataset, so that it would equal Literacy/Numeracy.

Arranging factor levels by another variable.

Imagine you want to show the average wages by study area. You would immidately compute these with some confidence intervals, you can even arrange them, but this will not help for the graph:

```
avwage <- piaac %>%
  group_by(studyarea)%>%
  summarize(meanwage=mean(Income, na.rm=TRUE))%>%
  arrange(meanwage)
```

```
ggplot(avwage, aes(x=meanwage, y=studyarea))+
  geom_point()
```

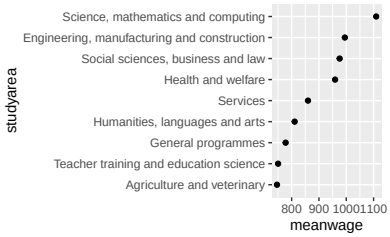


They are in some reverse alphabetical order! Not pretty at all. Lets change the order of studyarea so that it will be ordered by the size of meanwage variable. The forcats package comes to the rescue again.

```
library(forcats)
avwage$studyarea<-fct_reorder(avwage$studyarea, avwage$meanwage)
```

and do the graph again:

```
ggplot(avwage, aes(x=meanwage, y=studyarea))+
  geom_point()
```



Much better, we should just get rid of the NA-s again.

Your turn:

- Remove everyone who does not have Health-indicator (it is NA) from piaac. Find the median and average wage (`Income`) by `Health` from piaac, write it to a new dataset called `mwpiaac`. This should look smth like this:

```
## # A tibble: 2 x 3
##   Health    median average
##   <chr>      <dbl>   <dbl>
## 1 Excellent   815.   1031.
## 2 Fair        600     741.
```

- Now you have a dataset, where there are variables gender, Health, median and average. Use pivot_longer() to pivot the latest variables to a longer format, so that you would end up with smth like this (I named this `mwpiaaclong` as it is in long format now):

```
## # A tibble: 2 x 3
##   Health    name    value
##   <chr>     <chr>   <dbl>
## 1 Excellent median   815.
## 2 Excellent average 1031.
```

- Try to create the following graph on the side. I am using `shape=21` and `size=3` here and filling the point with the measurment-variable.

- Order the Health-variable to be in normal order – Poor-Fair-Good-Very good-Excellent, using `forcats::fct_relevel()` and make the graph again.