Intro to R days 5 and 6

Indrek Seppo and Nicolas Reigl

March 2-3, 2017

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

From the previous sessions

dplyr and piping

We went through the basics of data manipulation with dplyr last time. Recall that in the tidyverse – modern R – there was an "and then" operator: `%>%`. This sends the result of the last command to be the first argument of the next command. So a typical workflow would be something like this:

```
newdataset <- initialdata %>%
  filter(variable>4, !is.na(othervariable))%>%
  group_by(variable1, variable2)%>%
  summarize(meanofvar=mean(variable1, na.rm=TRUE), othervalue=myfunction(variable4, variable5))
```

The central package of `tidyverse` is `dplyr`, providing a grammar of data manipulation. The main verbs of dplyr's grammar of data manipulation were:

1* **select()** for selecting variables, you can use helper functions like **starts_with()**, or select consequetive variables with `: variable1:othervariable`. Separate variables (or criterions) by commas.

Your turn: YOUR TURN

1. Read the original piaac data in again

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

2. Create a new dataset `smallpiaac`, which contains only the variables `sequid`, `age`, `pvlit1`, `pvnum1`, `pvpsl1`.

Then there was **filter()** for subsetting the dataset. You can separate different conditions by commas, for example `sex=="Male", age > 29`. A useful way to select all the values in a set is to use `%in%` operator: `education %in% c("high", "medium")`.

Your turn: YOUR TURN

1. overwrite piaac with a dataset derived from piaac where the NA-s in either edlevel3, studyarea or health variable are removed. Remember – you have to use `is.na()` for this (actually `!is.na()`).
2. Create a new dataset `goodhealth`, which contains only those rows from piaac where health is either "Excellent" or "Very good".

And there was a powerful combination of: * **group_by()** – creates groups in the data by any number of variables * **summarize()** – computes some summarizing statistics of the variable (by groups if **group_by()** was used previously).

Your turn: YOUR TURN

1. Create a new dataset which would include average wage (average of variable `earnmth`) per education level (variable `edlevel3`) and `gender`. Call the newly created variable `averagewage`. I think you should also use the `na.rm=T` while finding the averages.
2. Lets make it bit more difficult – add 25.th and 75.th percentiles of the wages to the previous `summarize()` call. You can find the percentiles using function `quantile()`, which needs a second argument as well: the percentile (probs=0...1, so 25.th percentile would be 0.25). You should just add this `probs=` to the quantile call after the variable you want to find the quantile of (and you will also need a third parameter telling it to not take into account the NA values – `na.rm=T`. Call the new variables `pc25` and `pc75`. Then arrange the result by the variable `averagewage`, which you have just created.

The result should look smth like this:

```
## # A tibble: 3 x 5
## # Groups:   edlevel3 [3]
##   edlevel3 gender averagewage  pc25  pc75
##   <fct>    <fct>        <dbl> <dbl> <dbl>
## 1 Low      Female         506   302   545
## 2 Medium   Female         553   350   644
## 3 High     Female         865   526  1013
```

There was also **mutate** – to add variables to the dataframe:

```
piaac <- piaac %>%
  mutate(relativewage = earnmth / mean(earnmth, na.rm=T))
```

Note that the `mean(earnmnth)` would here find the average earnings over the entire dataset. But mutate also works with `group_by()` – if you use `group_by()` before it, it would take the average over this group.

A quick comment – you would like to add `ungroup()` at the end as the groups would be saved to the dataset otherwise and later on this could cause some unexpected behaviour.

Couple of additions:

- `summarize()` can only deal with functions which return a single value. There is another verb called `do()`, which can deal with functions which return a data frame. We will not touch it further here though.
- there is a verb called `rename()`, which renames the variables.

```
library(dplyr)

piaac.tmp <- piaac %>%
  rename(personId=seqid)
```

Lets try it. Lets create a new dataset called numeracyaverages, so that it would contain mean numeracy levels by gender and studyarea:

```
numeracyaverages <- piaac %>%
  group_by(gender, studyarea)%>%
  summarize(meannum=mean(pvnum1, na.rm=TRUE))
```

Conflicts in packages

Install and load the **plyr** package. Note the warnings, they are easy to miss. Now run the previous command again:

```
numeracyaverages <- piaac %>%
  group_by(gender, studyarea)%>%
  summarize(meannum=mean(pvnum1, na.rm=TRUE))
```

See, what happens? We dont have them grouped anymore, as we are now using the functions from `plyr`, not `dplyr`. The only way out is to unload `dplyr` and then load it again (just typing `library(dplyr)` will not help, unloading is neccesary!). There are two ways to do it – either uncheck dplyr from the `Packages` tab and check it again[1], or write:

[1] Sometimes you'd have to do it two times.

```
detach("package:dplyr", unload=TRUE)
```

One way to solve this problem is to use the package name with the function name (and if you are using a lot of conflicting packages, you learn to do this):

```
numeracyaverages<-piaac %>%
  dplyr::group_by(gender, studyarea)%>%
  dplyr::summarize(meannum=mean(pvnum1, na.rm=TRUE))
```

Your turn:

YOUR TURN

- Change the variable names in `piaac` to be more pleasing to the eye and graphs[2]:

[2] You can do it all in one `rename()` call, just remember to overwrite your previous dataset with the new one.

  - pvlit1 to Literacy
  - pvnum1 to Numeracy
  - pvpsl1 to Problem solving skills[3]
  - earnmth to Income
  - edlevel3 to Education
  - health to Health

- Remove all the lines where either Health, Education or Numeracy is missing[4].
- This is now very tricky stuff: find the mean, lower and upper confidence interval for the variable numeracy by Education level. Use the function `CI()` from package **Rmisc** for this. NB! You do not have this package installed yet. What is worse – it will load the package **plyr** when you read it in. And this will f up your `dplyr`. You now need to detach dplyr and then reattach it! One way to do this is unmark it in the Packages pane and then mark it again. Another thing - `CI()` will return 3 values, but summarize can only deal with 1. What should we do? Take a look at what `CI()` returns, by running `CI(piaac$numeracy)` – it returns a vector of three. How to access a single value? Just add `[1]` or `[2]` or `[3]` at the end of the call like this: `lower=CI(numeracy)[3]`. Write the resulting dataframe in a data object called **averages**.

  This is what you should have as a result:

[3] You will need to use the quatation marks to quote the name if there are spaces in it!

[4] Remember, variable!=NA will not work, you would want to use the `!is.na()` function.

```
averages
```

```
## # A tibble: 3 x 4
##   Education upper  mean lower
##   <fct>     <dbl> <dbl> <dbl>
## 1 High        291   290   288
## 2 Low         254   246   237
## 3 Medium      270   269   267
```

ggplot2

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.

Lets create a new variable in the piaac dataset to study the relationship between working and numeracy. We have a variable called `empl_status` there, that can have three values

```
unique(piaac$empl_status)
```

```
## [1] Out of the labour force Employed                Unemployed
## [4] Not known
## Levels: Employed Not known Out of the labour force Unemployed
```

Lets create a new variable that is TRUE when this variable is `Employed` and `FALSE` if it something else:

```
piaac$employed <- piaac$empl_status=="Employed"
summary(piaac$employed)
```

```
##    Mode   FALSE    TRUE
## logical    1410    4794
```

And now lets convert it to numeric for graphing purposes. R will automatically convert TRUE-s to 1 and FALSE-s to 0:

```
piaac$employed <- as.numeric(piaac$employed)
```

Your turn:

To study a relationship between two continuous variables, you would usually use `geom_smooth()` – it gives you smoothed conditional means by default.

1) Create a graph showing the relationship between Numeracy and Literacy. Connect x to Numeracy and y to Literacy. Remember to use `aes()` for this, as you are connecting aesthetics to variables in your data! Show points using geom_point() and the relationship using geom_smooth().

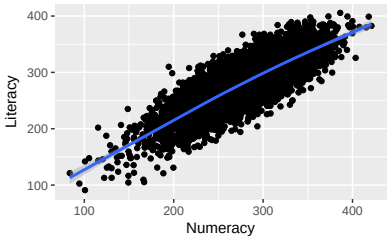


Figure 1: Expected result of the first exercise

2) Take a look if there is a noticable difference of this relationship by education level by connecting the color to education level variable in geom_smooth(). Again, dont forget the `aes()`.

3) You can graph binary outcomes the same way – geom_smooth() gives you the expected value, whic would be the percentage of ones. Lets see if there is a connection between numeracy and the probability of being employed (we just created the variable). Recreate the graph on the sideline.

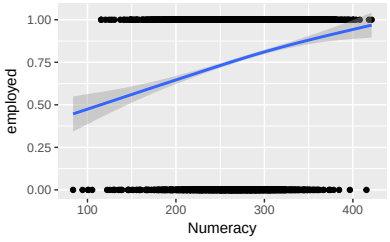


Figure 2: Expected result of ex 3

4) Ok, but maybe this is somewhat explained by the education level? Take a look by connecting color in the `geom_smooth()` to education.

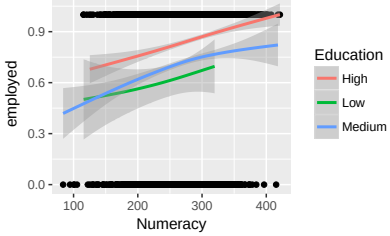


Figure 3: Expected result of ex 4

5) And the same stuff once more. Create a graph showing the relationship between age and employment, showing the difference between men and women by education level (result is on the side). You see that it goes a bit crazy here for the people with low education – it does not know that the probabilities can not go over or below 1 and 0, it overfits etc. But you will get the idea for High and Medium levels of education.

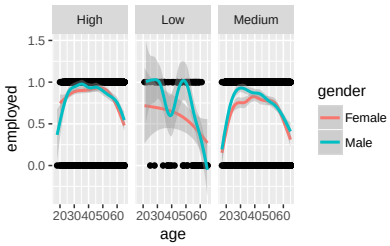


Figure 4: Expected result of ex 5

`geom_smooth()` actually allows you tou specify the functional relationship. Use `method="lm"` to get a linear model, or specify the function yourself. For example, if I would like to see how would the second order polynom perform here, I could give it an additional parameter of `formula = y ~ x^2` (but you would need to use the method="lm" aswell).

```
ggplot(piaac, aes(x=age, y=employed))+
  geom_point()+
  geom_smooth(aes(color=gender))+
  geom_smooth(aes(color=gender), method="lm",  formula= y~poly(x,2))+
  facet_wrap(~Education+gender)
```

Factors

We learned that for dealing with factors there is a package called `forcats`. To manually change the order of factor levels, you can use `fct_relevel()`.

For example: the current order of the `piaac$Health` is:

```
levels(piaac$Health)
```

```
## [1] "Excellent" "Fair"      "Good"      "Poor"      "Very good"
```

Lets change it:

```
library(forcats)
piaac <- piaac %>%
  mutate(Health = fct_relevel(Health, "Poor", "Fair", "Good", "Very good", "Excellent"))
levels(piaac$Health)
```

```
## [1] "Poor"      "Fair"      "Good"      "Very good" "Excellent"
```

But you could also have done it with a one-liner:

```
piaac$Health = fct_relevel(piaac$Health, "Poor", "Fair", "Good", "Very good", "Excellent")
```

To change the factor levels, you can use `fct_recode()`:

```
levels(piaac$children)
```

```
## [1] "No"  "Yes"
```

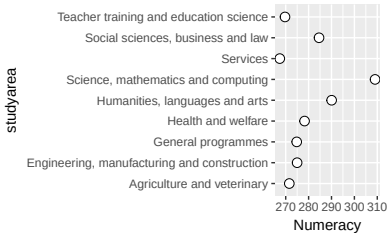
```
piaac <- piaac %>%
  mutate(children=fct_recode(children, "Has children"= "Yes", "No children"="No"))
levels(piaac$children)
```

```
## [1] "No children"  "Has children"
```

Third thing you would do quite a lot is order the level of factors by some variable. Lets create again a simple summarizing table of average numeracy scores by study area:

```
numscores <- piaac %>%
  group_by(studyarea)%>%
  summarize(Numeracy = mean(Numeracy))

ggplot(numscores, aes(x=Numeracy, y=studyarea))+
  geom_point(shape=21, size=3, fill="white")
```



These are not in a nice order at all. What can we do to plot them so that the highest would be on top and the lowest in the bottom? We can change the order of level manually, but there is an easier way. We can use `fct_reorder()` from `forcats`:

```
numscores$studyarea <- fct_reorder(numscores$studyarea, numscores$Numeracy)
```

Now try to run the plot again!

Your turn:

- Take the piaac dataset and change the levels of factor in Education variable so that "High" will become "Higher", "Low" to "Basic" and "Medium" to "Secondary". Use the **fct_recode()** function from the **forcats** package. The example from the help section will show you how to do it.
- Change the order of Education factor levels so that it will be Basic, Secondary, Higher. Use the **fct_relevel()** function from the forcats package to do it. The example in the help section will show you how to do it.

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[5]. While there is no single

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

markdown standard, it is pretty similar in wherever you encounter it, it is by no way R-specific.

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.

The only thing somewhat tedious is to generate tables, but luckily R provides a number of ways to do it for us.

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.

```
kable(averages)
```

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(averages),type="html", include.rownames=FALSE)
```

For figures you have additional arguments – `fig.cap=`, `fig.height=`, `out.height=` etc.

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

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

RStudio provides some additional RMarkdown templates. Check out the packages `tufte`[6], `rticles` and `rmdformats`. Some additional information is available here: https://blog.rstudio.org/2016/03/21/r-markdown-custom-formats/.

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

[6] Or `tint` (Tint Is Not Tufte) – a slightly modified version of tufte this handout is written in.

Lists

List is a data structure that we will meet quite often. Data frame consists of vectors of the same length, list can include whatever.

For example[7]:

[7] Note that we give names to list elements. R will not do it automatically, as it does for data frames.

```
name<-"Masha"
surname<-"Mishka"
length<-1.65
peripherals=data.frame(nr.of.fingers=c(5,5), nr.of.toes=c(5,5))
mashadata <- list(first.name=name, surname=surname, length=length, peripherals=peripherals)
class(mashadata)
```

```
## [1] "list"
```

```
mashadata #vaatame sinna sisse:
```

```
## $first.name
## [1] "Masha"
##
## $surname
## [1] "Mishka"
##
## $length
## [1] 1.65
##
## $peripherals
##   nr.of.fingers nr.of.toes
## 1             5          5
## 2             5          5
```

Lists can contain lists that can contain lists etc.

To address a single element of list we can use the $-sign:

```
mashadata$first.name
```

```
## [1] "Masha"
```

Or double straight brackets (the name of the element has to be quoted then):

```
mashadata[["first.name"]]
```

```
## [1] "Masha"
```

You can in fact use single straight brackets, but they will return a list, not the initial data type (the data.frame, which we included in the list), which you will want to get in practice.

```
class(mashadata["peripherals"])
```

```
## [1] "list"
```

```
class(mashadata[["peripherals"]])
```

```
## [1] "data.frame"
```

If we want to access the vector **nr.of.fingers** from the data frame **peripherals** from the list **mashadata**, then we can do it in the following way:

```
mashadata[["peripherals"]]$nr.of.fingers
```

```
## [1] 5 5
```

but not like this:

```
mashadata["peripherals"]$nr.of.fingers
```

```
## NULL
```

There is one thing to note when dealing with lists. Data frame will not let you include two columns with the same name. List would not care less:

```
list2<-list(a=3, b=4, a=5)
list2$a #Not even a warning!
```

```
## [1] 3
```

The reason we pay so much attention to lists, is that a lot of the results R will give you, are in the form of lists. Take t.test() – t test compares two samples and asks if it is statistically viable that they come from the same population, that the differences could be just thanks to sampling error. If we only give it one vector as an input, it will tell us whether the mean is statistically different from 0.

Lets do a t-test for **Literacy**-variable (measuring literacy levels) in our **piaac** dataset[8]:

[8] Note that we are using the piaac data incorrectly here! In reality there are ten different plausible values given for literacy levels, as there is considerable uncertainty involved. Do do it correctly you should use all the information – this is possible with a package called **svyPVpack**. Neither do we use the weights here.

```
t.test(piaac$Literacy)
```

```
##
##  One Sample t-test
##
## data:  piaac$Literacy
## t = 511.9, df = 6203, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  278.3766 280.5169
## sample estimates:
## mean of x
##  279.4468
```

It will give us 95% confidence intervals for the population. Meaning – considering this sample, believing that this is a random sample, we would believe that the correct average for the whole population should be inside this interval with quite a high certainty. The average for the sample is 275,64, we believe, that the average for the whole population should fall somewhere between 275...277.

`t.test()` shows us the results, but if we save it to a data object, we will find out, that it actually produces a list. Lets do it:

```
result <- t.test(piaac$Literacy)
class(result)
```

```
## [1] "htest"
```

```
typeof(result)
```

```
## [1] "list"
```

```
str(result)
```

```
## List of 9
##  $ statistic  : Named num 512
##   ..- attr(*, "names")= chr "t"
##  $ parameter  : Named num 6203
##   ..- attr(*, "names")= chr "df"
##  $ p.value    : num 0
##  $ conf.int   : atomic [1:2] 278 281
##   ..- attr(*, "conf.level")= num 0.95
##  $ estimate   : Named num 279
##   ..- attr(*, "names")= chr "mean of x"
##  $ null.value : Named num 0
##   ..- attr(*, "names")= chr "mean"
##  $ alternative: chr "two.sided"
```

```
##  $ method     : chr "One Sample t-test"
##  $ data.name  : chr "piaac$Literacy"
##  - attr(*, "class")= chr "htest"
```

Your turn:

- confidence intervals are inside this list as a vector called **conf.int**. Can you access it?

1) create a variable called **lower**, and assign it the lower value of conf.int (the first value in this vector)
2) create a variable called **upper**, and assign it the upper value of conf.int (the second value in this vector)

Functions

Writing a custom function or command in R is extremely simple. Take a look at the example:

```
timesthree<-function(x){ #we name the function and say that its input will be named x
  result <- x * 3  #we do some stuff
  return(result) #we will return the answer
}
```

You now need to make R aware of the new function – just select it and click Run (or cntrl+enter).

Lets find out how it works:

```
timesthree(4)
```

```
## [1] 12
```

```
timesthree(c(3, 4, 5))
```

```
## [1]  9 12 15
```

We know that functions could have default values. This is how to specify the defaults:

```
multiplydivide<-function(x, multiplier=3, divisor=1){
  result <- x * multiplier / divisor  #use the input
  return(result) #return the result
}
```

If we will not give any values for **multiplier** or **divisor**, then it will use the default values. But we can change them:

```
multiplydivide(x=4)
```

```
## [1] 12
```

```
multiplydivide(x=4, multiplier=1, divisor=2)
```

```
## [1] 2
```

Couple of things to consider when writing functions – think through what kind of data structure do you give the function as an input. Is it a data frame? Is it a vector? This is the object that will be called x now, you will need to address it appropriately.

Minimal about conditional statements and control flows in R

You can write conditional statements in R in the following way

```
if (condition) {
  whattodo
} else {
    dosomethingelse #else part is not required
}
```

For example[9]:

[9] You do not need to use the square brackets, if the `whattodo` is a single line. For more lines you need the brackets.

```
nameOfTheBear<-"George" #let us have a bear named George

if(nameOfTheBear=="George") friendOfABear<-"Tom"
if(nameOfTheBear=="Bill") friendOfABear<-"Mary"

friendOfABear #who is the friend of the bear?
```

```
## [1] "Tom"
```

Another useful feature is a for cycle[10]:

[10] Note that we are using `in` here, not `%in%`!

```
for (i in c(1:10)){
cat(i)
}
```

```
## 12345678910
```

`cat()` will write the object into a standard output – into console. You notice that it will write the stuff without any whitespace. Lets change it:

```
for (i in c(1:10)){
cat(paste(i, "\n", sep=""))
}
```

What I did here was paste'ing multiple strings together, and telling it to not leave any spaces between them. This is a useful command, lets try it again:

```
paste("Ahha", "uhhuu")
```

```
## [1] "Ahha uhhuu"
```

You can go through the values of some variables (all the unique values of factor are given by `levels(df$factorvar)`, all the unique values of a character-type variable can be accessed by `unique(df$textvar)`:

```
for (i in levels(piaac$gender)){
  cat(i)
  tmp <- filter(piaac, gender==i)
  averagewage <- mean(tmp$earnmth, na.rm=T)
  cat("\nAverage wage of", i, "is", averagewage, "\n")
}
```

```
## Female
```

```
## Warning in mean.default(tmp$earnmth, na.rm = T): argument is not numeric or
## logical: returning NA
```

```
##
## Average wage of Female is NA
## Male
```

```
## Warning in mean.default(tmp$earnmth, na.rm = T): argument is not numeric or
## logical: returning NA
```

```
##
## Average wage of Male is NA
```

To learn more about control flow in R take a look at the following free tutorial: https://www.datacamp.com/courses/intermediate-r.

YOUR TURN

Your turn:

1) create a function called **subtract**, which takes as an input two data objects (for examplem *a* and *b*) and subtracts one from the other.

2) create a function called **findlower**, which takes in a vector, does a t-test and returns the lower bound. First of all look at what `t.test(piaac$pvnum1)` gives you. Write out the command to get this lower bound out of this. Then replace `piaac$pvnum` with `x` or whatever you are using as an input varible for the function and return the answer.

3) create a function called **findupper**, which takes in a vector, does a t-test and returns the upper bound.

4) **Only for people who are too quick:** Rewrite the previous function, that found the upper bound so, that you can give it additional parameter specifing which bound to return (by default "lower"), and if it is "lower", then it returns the lower bound, if it is "upper", it returns the upper bound.

You can use your newly created functions in `dplyr` `summarize` calls. Lets try it:

```
piaac %>%
  group_by(gender)%>%
  summarize(lowernum=findlower(pvnum1), averagenum=mean(pvnum1), uppernum=findupper(pvnum1))
```

Joining data

In the real life you do not have data in one dataset. You have multiple datasets you would like to join with each other. `dplyr` offers functions for just that. We have `left_join()`, `right_join()`, `inner_join()`, `full_join()` `anti_join()` and `semi_join()`. Lets take a look at them. Lets create couple of small datasets:

```
animallength <- data.frame(animal=c("cat", "dog", "elephant"), length=c(10, 20, 50))
animalwidth <- data.frame(animal=c("cat", "dog", "bear"), width=c(5, 15, 10))
```

And lets try it:

```
animals <- left_join(animallength, animalwidth)
animals
```

```
##     animal length width
## 1      cat     10     5
## 2      dog     20    15
## 3 elephant     50    NA
```

It took all the animals from the first dataset (the dataset on the left) and added the values from the second dataset to these. If it could not find a matching animal from the second dataset, it added a `NA`.

By default it tries to merge the datasets by the variables present in both datasets. In case the variable names are different in different datasets (*e.g.* Company in the first one and Firm in the second one) you can specify this with a parameter `by=c("Company", "Firm")`.

The `right_join()` would take all the animals in the second dataset and try to add columns from the first dataset to these[11].

'full_join() would retain all the animals from both dataset:

```
full_join(animallength, animalwidth)
```

```
##     animal length width
## 1      cat     10     5
## 2      dog     20    15
## 3 elephant     50    NA
## 4     bear     NA    10
```

And `inner_join()` would only leave in the animals which are present on both datasets:

```
inner_join(animallength, animalwidth)
```

```
##   animal length width
## 1    cat     10     5
## 2    dog     20    15
```

`anti_join()` would compare the datasets and remove everything that is present in both dataset, leaving only the rows which are in the first dataset and not in the second one (you'll actually find use to this – this is a quick way for testing whether there are some values in one dataset that are missing in another).

```
anti_join(animallength, animalwidth)
```

```
##     animal length
## 1 elephant     50
```

```
semi_join(animallength, animalwidth)
```

```
##   animal length
## 1    cat     10
## 2    dog     20
```

[11] Why would you need both `left_join()` and `right_join()` if you could just switch the arguments? The reason is the piping in tidyverse – if you use `%>%` the result of the previous lines of code will be presented as the first argument to `right_join()` or `left_join()`.

YOUR TURN

Your turn:

First read in the the following datasets: `http://www.ut.ee/~iseppo/gdppercap.csv` and `http://www.ut.ee/~iseppo/population.csv`. But use `read_csv()` from `readr` package. We have used base-R-s `read.csv()` mostly in the class[12], but readr is in fact a better option. It does some things differently from `read_csv()`: it does not convert text to factors automatically (you have to do it yourself manually by `df$variable <- as.factor(df$variable)`),

[12] The reason being that at one point `readr` had a bug and did not work well with internet addresses – this is fixed now.

it retains the original column names if they contain spaces or other special characters (`read.csv()` replaces them with points), it even tries to parse the dates and convert them to date-type variables. It also reads zipped files and it is much quicker than `read.csv()`. Note that expects the files to be in UTF-8 encoding by default.

- take a look at these files – which variables do they contain? Now join them so that all the countries present in either of the datset would be included. For how many countries is there no data of GDP per capita for this year?

You might also want to add some datasets "on top of each other". Base-R offers a function called `rbind()` for this (and `cbind` to bind together columns), dplyr has `bind_rows()` – a somewhat more intelligent version. Download the following files:

Read in the following files:

```
q1.2017 <- read_csv("http://www.ut.ee/~iseppo/q12017.gz")
q2.2017 <- read_csv("http://www.ut.ee/~iseppo/q22017.gz")
q3.2017 <- read_csv("http://www.ut.ee/~iseppo/q32017.gz")
```

Take a look at them. Oh, they are in Estonian! These are all the business entities in Estonia detailing how many people they employed, how much did thay pay VAT and how much employment taxes. Estonian tax and customs board gives this kind of data out every quarter and you can download it from their website.

What could we do with this data? Lets add it together to one big dataframe! But to be able to tell the difference – from which quarter was the data, we will need to add the information to the datasets:

```
q1.2017$quarter <- "q1.2017"
q2.2017$quarter <- "q2.2017"
q3.2017$quarter <- "q3.2017"
```

Now we can create a single dataset:

```
tax.2017 <- bind_rows(q1.2017, q2.2017, q3.2017)
dim(tax.2017)
```

```
## [1] 361453     11
```

Your turn: YOUR TURN

We have been using `group_by()` and `summarize()` a lot.

- Can you find the nr of employees (sum of employees) by county and quarter using these two verbs? Use `na.rm=T` in the `sum()` as a parameter

```
## # A tibble: 3 x 3
## # Groups:   county [1]
##   county        quarter nrofemployees
##   <chr>         <chr>           <int>
## 1 Harju maakond q1.2017        396744
## 2 Harju maakond q2.2017        404940
## 3 Harju maakond q3.2017        403618
```

We were using `spread()` from the package `tidyr` to convert tidy data to wide format. Load the package `tidyr`, look at the help text of spread and spread the data so that the name of the quarter becomes a new variable and the `nrofemployees` will be the thing that will be presented in the cells. Remember, you had to specify the `key` (which variable contents will now be turned to new variables) and the `value`. The result should look something like this:

```
## # A tibble: 3 x 4
## # Groups:   county [3]
##   county           q1.2017 q2.2017 q3.2017
##   <chr>              <int>   <int>   <int>
## 1 Harju maakond     396744  404940  403618
## 2 Hiiu maakond        2912    3075    2911
## 3 Ida-Viru maakond   45841   46326   46635
```

- Create a new variable in this dataset that you just created called change, and make it so that it would be the difference between `q3.2017` and `q1.2017`. Arrange the dataset by this variable. Which counties created the most jobs and which ones the least?

An overly quick intro to regressions in R

First lets reread the piaac dataset without the modifications we have made into it:

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

So you want to run some regressions. How to do it in R? We will use a regular least squares regression here, but everything else would be using more or less the same syntax.

First specify a regression model:

```
piaac$logincome <- log(piaac$earnmth)
model1 <- lm(logincome ~ edlevel3 + gender, weights=weights, data=piaac)
model1
```

```
##
## Call:
## lm(formula = logincome ~ edlevel3 + gender, data = piaac, weights = weights)
##
## Coefficients:
##    (Intercept)     edlevel3Low  edlevel3Medium      genderMale
##         6.5561         -0.5063         -0.3571          0.5209
```

The `model1` is not terribly informative, showing only the model and the point estimates. Lets try `summary()`:

```
summary(model1)
```

```
##
## Call:
## lm(formula = logincome ~ edlevel3 + gender, data = piaac, weights = weights)
##
## Weighted Residuals:
##     Min      1Q  Median      3Q     Max
## -44.092  -3.718   0.209   3.916  28.711
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)     6.55606    0.01635  400.90   <2e-16 ***
## edlevel3Low    -0.50626    0.03338  -15.17   <2e-16 ***
## edlevel3Medium -0.35714    0.02087  -17.12   <2e-16 ***
## genderMale      0.52091    0.01988   26.20   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.765 on 4057 degrees of freedom
##   (3571 observations deleted due to missingness)
## Multiple R-squared:  0.1861, Adjusted R-squared:  0.1855
## F-statistic: 309.2 on 3 and 4057 DF,  p-value: < 2.2e-16
```

This is slightly better – you get a traditional regression table with standard errors and p-values etc. As we were using logincome as the dependent variable we can interpret the coefficients roughly as percentages[13].

This results dataobject that we just wrote the regression results into, is again a list, and we can get everything out of this list (take a look at `str(model1)`). You will find coefficients there, residuals, fitted values etc. So to access some coefficients (e.g. for automatic inclusion in our report) we can call:

[13] Economists always tend to interpret them as rough percentages, but be aware if there are anyone from mathematics close to you. For small values of these coefficients they are indeed very close to percentages (so if a coefficient 0.1 for genderMale would have meant that men earn ~10% more). The precise nr would be `exp(0.1)-1=0.1051`, which is around 10%. But `exp(0.3)-1`is already 0.35 and `exp(0.5)-1` is `0.65` (so you should say 65% instead of 50% here). Handle this with some care when the coefficient is higher than 0.2 or so, or someone will remark it at your presentation.

```
model1$coefficients["edlevel3Medium"]
```

```
## edlevel3Medium
##     -0.3571447
```

There is actually a package called `broom` in the tidyverse that makes this much easier creating a regular dataframe of the coefficients:

```
library(broom)
model1overview <- tidy(model1)
model1overview
```

```
##             term   estimate  std.error statistic       p.value
## 1    (Intercept)  6.5560627 0.01635338 400.89967  0.000000e+00
## 2    edlevel3Low -0.5062603 0.03337649 -15.16817  1.369427e-50
## 3 edlevel3Medium -0.3571447 0.02086588 -17.11620  1.810446e-63
## 4     genderMale  0.5209098 0.01987817  26.20512 5.694088e-140
```

Another function `glance()` gives you a dataframe of some parameters helping you to understand how precise the model is with this data.

```
glance(model1)
```

```
##   r.squared adj.r.squared    sigma statistic       p.value df    logLik
## 1 0.1860817     0.1854798 6.764605  309.1765 9.008321e-181  4 -3863.257
##        AIC      BIC deviance df.residual
## 1 7736.515 7768.061 185647.9        4057
```

From the model object you can access fitted values, but residuals are usually more interesting. Lets add the residuals to the original dataframe. To do this we can use `augment()` function from broom[14]:

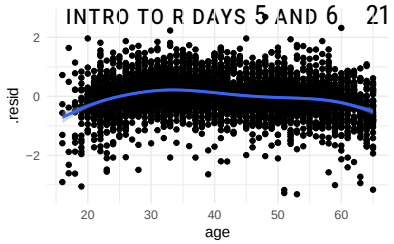
[14] There is an alternative – `residuals()` in base R would provide us the vector of residuals, augment() will work on many types of models but not all, the `residuals()` is usually implemented everywhere.

```
piaacRes <- augment(model1, data=piaac)
```

Note that the resulting piaacRes dataset is smaller than piaac – all the values which had NA-s in one of the regression variables are removed from here (you can add `na.action=na.exclude` as a parameter to the regression call leave the NA-s in).

Why is this important? You can now do some quick analysis of residuals – is there heteroscedasticity in the data? Just plot the residuals against other variables. Would adding some additional variables help? Lets try it and plot the residuals against age:

```
library(ggplot2)
ggplot(piaacRes, aes(x=age, y=.resid))+
  geom_point()+
  geom_smooth()+
  theme_minimal()
```



There is clearly a relationship, but it does not seem to be a linear one. Lets add a polynomial of age to the model:

```
model2 <- lm(logincome ~ edlevel3 + gender + poly(age,2), weights=weights, data=piaac)
```

We could have added `+age +age^2` as well – this would not have changed any other coefficients, but would mess up prediction intervals (why? Because R would treat `age` and `age^2` as separate independent variables while they are clearly not). You wouldn't usually care about prediction intervals, so many people are just adding the `age` and `age^2` to the model and so can you.

If you want to add cross-effects, use `*`:

```
model3 <- lm(logincome ~ edlevel3*gender + poly(age,2), weights=weights, data=piaac)
```

And to change the base-value of a nominal variable, make the value you want to use as a base to be the first in the factor:

```
library(forcats)
piaac$edlevel3<-fct_relevel(piaac$edlevel3, "Secondary")

## Warning: Unknown levels in 'f': Secondary

model4 <- lm(logincome ~ edlevel3*gender + poly(age,2), weights=weights, data=piaac)
```

Predictions at specified values

You can easily plot the predicted values of a regression by using a `predict()` function. You need to give the dataframe newdata to it, specifying for which values you want to get the predictions. You can add the initial dataset (but you would also get the predicted results for the initial dataset with a function fitted(modelname)). To create a dataset of all possible values, use `expand.grid()`:

```
mydata <- expand.grid(edlevel3=levels(piaac$edlevel3), gender=levels(piaac$gender), age=c(20:65))
mydata$prediction <- predict(model3, newdata=mydata)
ggplot(mydata, aes(x=age, y=prediction))+
  geom_point()+
  facet_wrap(~edlevel3)+
  theme_minimal()
```

If you want to futher visualize your regressions, take a look at package `visreg` http://pbreheny.github.io/visreg/. If you wan to have nicely outputed regression tables, package `stargazer` should help you out: https://www.jakeruss.com/cheatsheets/stargazer/.