Intro to R, days 3 & 4

Indrek Seppo and Nicolas Reigl

Feb 16, 2017

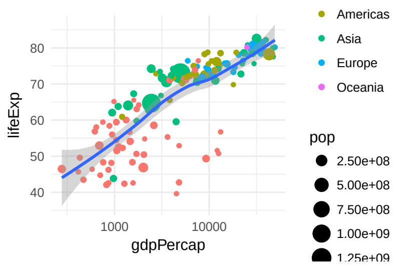
Preparation of this course was supported by HITSA – Estonian Information Technology Foundation for Education.

continuing with ggplot2

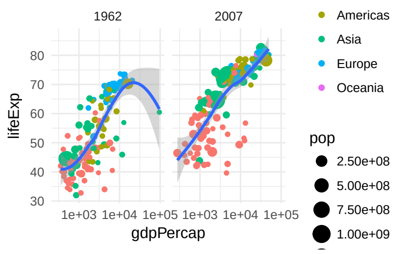
Your turn:

1) Lets recreate one of the classic visualisations. Install a package called `gapminder` and load it in. You can now use a dataset called gapminder (try it: `summary(gapminder)`).

- take a subset of this data leaving only year 2007 in.
- create a graph where gdpPercap is mapped to x, lifeExp is mapped to y and add points as a geom.
- map the size of the points to variable `pop`
- map the color of the points to variable `continent`
- add two more lines: `scale_x_log()` – this will change the x-scale to be logarithmic, and `theme_minimal()`



2) Now recreate the previous graph with years 1962 and 2007 (so make a new subset of the data – the year can now be either 1962 or 2007). Use previous graph as a starting point and add faceting to it using `year` as a faceting variable.



Scales

x ja y

Everything that is connected with aesthetic, is connected with a scale - x, y, size, color etc. All the scales have some common features – e.g **breaks** and **labels**. Lets try it:

```
ggplot(data = piaac, aes(x = edlevel3, y=logincome)) +
  geom_jitter(alpha=0.3)+
  theme_minimal()+
  scale_y_continuous(breaks=c(0,2,3), labels=c("small", "average", "big"))
```

R will automatically add some space in the beginning and end of the **x** and **y** scale. We can control it with a parameter **expand=c(0.1, 0.1)**, with the first number showing how many percantages and the other number how much in the nominal terms is added.

NB! There is one particular thing you should never use: you can set limits in **scale_x_continuous()**, but do not do it ever. It will just cut all the data above and below the limit and the smoothers etc will then be meaningless. Use **coord_cartesian()** to zoom into where you want to zoom in.

We were dealing with the continuos scales up until now, but there are others – **scale_x_discrete()** for discrete x-axis, **scale_x_date()** and **scale_x_datetime()**, to show the dates with some convenience() **date_breaks="1 months"**, will show all the months etc).

Colorscales and legends

GGplot adds the colors automatically, but we can tell it what kind of colors to use. There are two different scales again - for continuous and discrete data. You can see all the available scales bywriting `scale_color` (või `scale_fill`) into the console and hit the tabulator.

One very usable color scale is **scale_color_brewer()** (if your data is discrete). Take a look at the colorbrewer2 website: http://colorbrewer2.org/. You can choose the pallettes that are color-blind safe etc.

```
ggplot(data = piaac, aes(x = edcat6))+
  geom_bar(aes(fill=edcat6))+
  scale_color_brewer("Education level", palette="Pastel1")
```

Note, that *color brewer* will usually give you up to 8 different colors

The place to go if you have any questions about the legends is: http://www.cookbook-r.com/Graphs/Legends_(ggplot2)/.

You will usually have the legend automagically, the legend can be removed with the function `guides()`:

```
ggplot(data = piaac, aes(x = EDCAT6))+
  geom_bar(aes(fill=EDCAT6))+
  scale_color_brewer(palette="Pastel1")+
 guides(fill=FALSE)
```

Also take a look at the viridis color scheme: https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html.

Manual scales

We will sometimes want to predefine which colors to use. E.g. if we want to use the same colors in the subsequent graphs.

We will need named vectors for this:

```
ggplot(data = piaac, aes(x = edlevel3))+
  geom_bar(aes(fill=edlevel3))+
  scale_fill_manual(values=c("Low"="red",
                             "Medium"="blue",
                             "High"="orange"))
```

You can use names, you can use RGB-codes etc. And it is exactly the same with other manual scales – if you want to predefine point shapes, you would use `scale_shape_manual()`

Your turn:
We downloaded the gdp per capita PPP data before by:

```
gdp.pc <- wb(indicator = "NY.GDP.PCAP.PP.KD", POSIXct = TRUE)
```

1) Select three "countries" – the world, Estonia and your native country from the data (using `subset()` or `filter()`).

2) create a graph where x-axis is the date (date_ct), y-axis is the value, and countries are grouped by color

3) create a second graph where each country has its own facet.

Statistics in ggplot

There is one more concept in grammar of graphics, called statistics. We do not need to specify these often, as each geom has its default statistics and it usually just works, but there are couple of cases were we need them.

Lets look at the help file of geom_bar(). It will tell us that the default statistics used is count (stat="count"). What this means is that by default it will count all the cases as we saw previously:

```
ggplot(piaac, aes(x = edlevel3)) + geom_bar()
```

But what if we do not want to count the ocurrences, but have the following data precomputed for us:

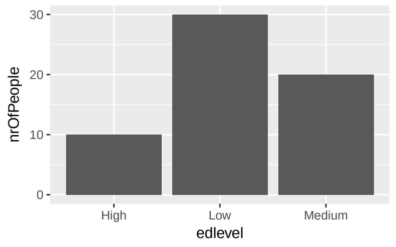
```
averages <- data.frame(edlevel = c("High", "Medium", "Low"), nrOfPeople = c(10,
    20, 30))
averages

##   edlevel nrOfPeople
## 1    High         10
## 2  Medium         20
## 3     Low         30
```

Imagine we want the bars to represent the number. The way we used geom_bar() previously wont work:

```
ggplot(averages, aes(x=edlevel))+
  geom_bar()
```

What we need is to change the stat argument like this:

```
ggplot(averages, aes(x=edlevel, y=nrOfPeople))+
  geom_bar(stat="identity")
```



Or take geom_boxplot as an example – by default it uses stat="boxplot" – finds the medians and quartiles by itself from the data. But if you happen to have agregated data, you can use your own values.

Lets load a dataset called wagebirthyear:

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

Take a look at this data. It includes percentiles of monthly salaries earned in 2011 by birth year. Lets first create a variable age into this dataset:

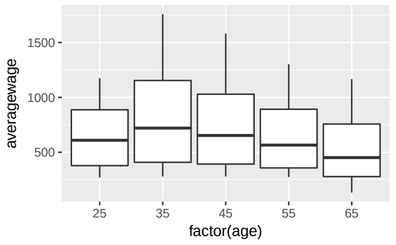
```
earnings$age <- earnings$year - earnings$birthyear
```

and lets take some random five ages from there:

```
toplot <- subset(earnings, age %in% c(25, 35, 45, 55, 65))
```

Lets now create a box-and-whisker plot from this so that the whiskers would denote the 10-th and 90-th percentile (Note that I changed the age to be interpreted as a factor variable.):

```
ggplot(toplot, aes(x=factor(age)))+
geom_boxplot(aes(y=averagewage, lower=pc25,
middle=medianwage, upper=pc75,
ymin=pc10, ymax=pc90), stat="identity")

## Warning: Ignoring unknown aesthetics: y
```



Your turn:

I would not graph this kind of data the way we just did. Lets see if you manage to make the following graph. The grey area shows 25.-th to 75.-th percentile for each age group. So 25% of the people are below the grey area (for their age), 25% are above it and 50% are inside the area. The dataset used is `earnings`, x is connected to `age` variable in the dataset, the grey area is done by `geom_ribbon()` which requires `x`, `ymin` and `ymax` to be set as aesthetics. I have also told it with additional parameters that `fill="grey"` and `alpha=0.3`. Then there are two lines – average and median. For the average I have set the color to be blue, for the median I have set the color to be orange. At the very end I have added theme_minimal().

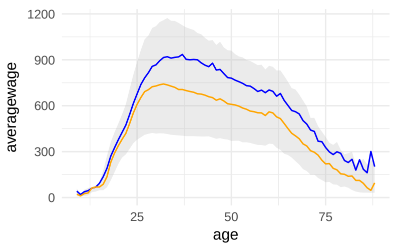


Figure 1: Expected result

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[1] again. Lets remove everyone who does not know their education level:

[1] It was at http://www.ut.ee/~iseppo/piaacest.csv.

```
piaac <- read.csv("http://www.ut.ee/~iseppo/piaacest.csv", fileEncoding = "utf-8")
piaac <- subset(piaac, !is.na(edlevel3))
```

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

```
piaac <- subset(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 `edlevel3`.

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.

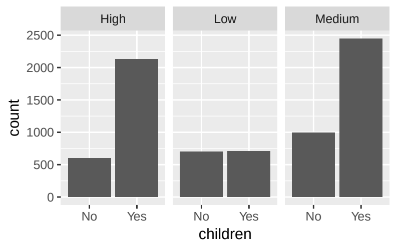


Figure 2: Solution to the first exercise

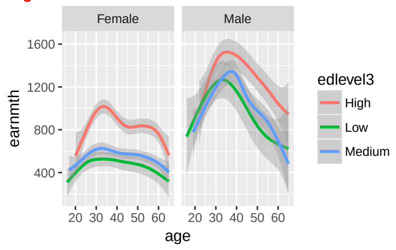


Figure 3: Second exercise

Reordering factor levels

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

```
library(forcats)
piaac$edlevel3 <- fct_relevel(piaac$edlevel3, "Low", "Medium", "High")
```

Creating the graph again would now be in logical order:

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

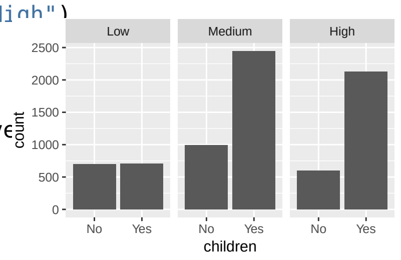


Figure 4: First exercise with correct ordering

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$edlevel3 <- fct_recode(piaac$edlevel3, ‘Low education‘ = "Low", Highschool = "Medium")
levels(piaac$edlevel3)
```

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

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[2]:

[2] 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, earnmth)
```

But we can also rewrite it in this way:

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

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$logincome<-log(piaac$earnmth)
piaac.small <- piaac %>%
  select(gender, logincome, edlevel3)
head(piaac.small, n=3)
```

```
##   gender logincome   edlevel3
## 1   Male        NA Highschool
## 2 Female  6.619678 Highschool
## 3 Female        NA Highschool
```

Doesnt 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:earnmth)
head(piaac.small, n=3)
```

Or all the variables that start with "pv", and then add "seqid":

```
piaac.small <- piaac %>%
  select(starts_with("pv"), 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 1. 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(earnmth, na.rm=TRUE), n=n())
```

```
##   averagewage    n
## 1    885.3164 7572
```

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

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

```
##   meanincome medianincome
## 1   885.3164          700
```

Your turn:

- Use summarize to find max and min income (earnmth) 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, edlevel3) %>% summarize(meanwage = mean(earnmth,
    na.rm = TRUE), totalwage = sum(earnmth, 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(earnmth>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(earnmth > 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 (pvnum1) is at least 300. What is their average monthly income (earnmth)?

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 = pvnum1 + pvlit1)
```

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(pvnum1))
```

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 = earnmth/mean(earnmth))
```

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 `pvlit1/pvnum1`.

Wide and long data

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

```
library(eurostat)
gdpPerCap <- get_eurostat("tec00114")
head(gdpPerCap, n = 3)
tmp <- label_eurostat(gdpPerCap)
View(tmp)
```

This is called the long or tidy format. R loves tidy data, and most of the data sources provide it. In practice we usually find data in the wide format:

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

The same data in the long format (tidy data) would be:

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

Youll need a package **tidyr** to convert between the formats and the two main functions for this are: **`gather()`** ja **`spread()`**. Lets install it and read it in:

```
library(tidyr)
```

From wide to long format – gather()

We can get from wide to long format with **`gather()`**. Lets take a look at the help:

```
'?'(gather)
```

We need data, parameters “key=”, “value=” and the columns we wish to gather. Lets look at an example:

```
animals <- data.frame(animal = c("bear", "cat"), length = c(10, 4), width = c(7,
    3), age = c(3, 5))
animals
```

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

```
animals.long <- gather(data = animals, key = "measurement", value = "value",
    length, width, age)
animals.long
```

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

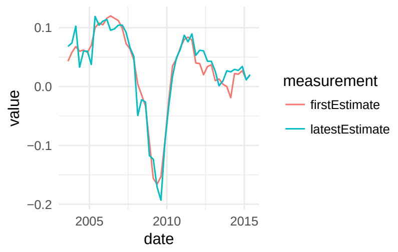
Your turn:

- Create a new dataset from the animals dataframe, so that it would look like this (note which variables are gathered):

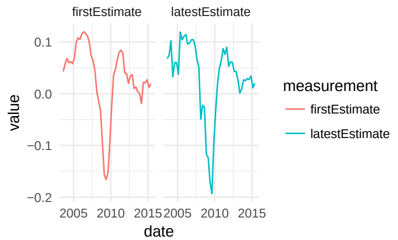
```
##   animal age measurement value
## 1   bear   3      length    10
## 2    cat   5      length     4
## 3   bear   3       width     7
## 4    cat   5       width     3
```

- 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.

From long to wide format – spread()

Lets look at the helpfile of **spread()**: `?spread`. It wants a dataset, *key*-variable and *value*-variable.

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

And we are back to where we started.
We will use spread to present data in the tables.

Your turn:
Lets download again the gdp data from World bank (You can use whichever countries you want):

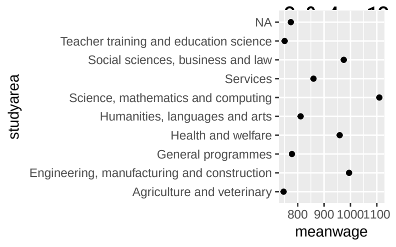
```
library(wbstats)
gdp.pc <- wb(indicator = "NY.GDP.PCAP.PP.KD")
gdp.pca <- subset(gdp.pc, country %in% c("World", "Estonia", "Finland"))
```

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.

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(earnmth, na.rm=TRUE))%>%
  arrange(meanwage)

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

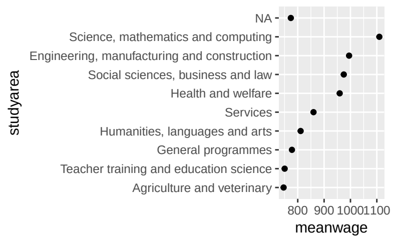


They are in some reverse alphabetical order! Not pretty at all. Lets change the order of studyarea so that it will be ordered by the size of meanwage variable. The forcats package comes to the rescue again.

```
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:

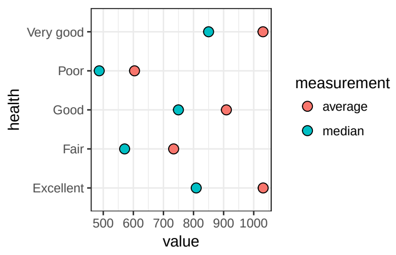
- Remove everyone who does not have health-indicator (it is NA) from piaac. Find the median and average wage (`earnmth`) by `health` from piaac, write it to a new dataset called `mwpiaac`. This should look smth like this:

```
## # A tibble: 2 x 3
##   health    median average
##   <fct>      <dbl>   <dbl>
## 1 Excellent    809    1031
## 2 Fair         571     734
```

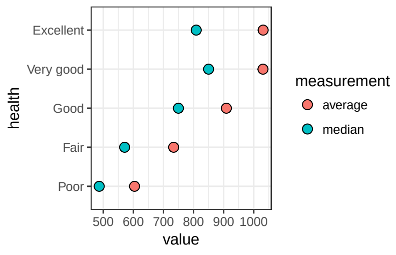
- Now you have a dataset, where there are variables gender, health, median and average. Gather the latest variables together, so that you would end up with smth like this (I named this `mwpiaaclong` as it is in long format now):

```
## # A tibble: 2 x 3
##   health    measurement value
##   <fct>     <chr>       <dbl>
## 1 Excellent median        809
## 2 Fair      median        571
```

- 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.



Additional materials

Formatting tables in R

If you want to get some tables out of R to use them in MS Word, it would usually require to first produce the table in html, and then import it into Word. One way is to use package z-table. You can find an introduction from: `https://cran.rstudio.com/web/packages/ztable/vignettes/ztable.html` (google: ztable vignette).

To write straight to Office formats, you might want to look at a new interesting package **ReporteRs**.

Another solution is **stargazer**, which offers us a command named **stargazer()**. Install it and load it in.

```
library(stargazer)
```

Lets look at the help `?stargazer`. More parameters than one can count! It can output a number of different type tables. If you send a *data.frame* to it, it will make you an overview table:

```
stargazer(piaac, digits = 1, header = FALSE)
```

Statistic	N	Mean	St. Dev.	Min	Max
X	7,572	3,814.9	2,203.6	1	7,632
seqid	7,572	3,814.5	2,203.7	1	7,632
age	7,572	40.9	14.3	16	65
earnhr	4,052	5.9	27.6	0.2	1,621.8
pvlit1	7,572	275.8	44.1	91.0	405.6
pvnum1	7,572	272.2	45.0	83.7	421.2
pvpsl1	5,231	275.7	42.4	112.6	413.2
earnmth	4,058	885.3	687.3	20.8	7,222.2
weights	7,572	117.7	24.2	68.3	227.6
logincome	4,058	6.6	0.7	3.0	8.9

By default it produces latex tables – this is for those of you who use LaTeX. But you can tell it to produce html-tables (**type="html"**) and to write it in a file (**out="failinimi.html"**). You can open the file in MS Word then and copy it to your document.

```
stargazer(piaac, type = "html", title = "My table", iqr = TRUE, out = "table1.html",
    digits = 1)
```

I have added the parameter **iqr=TRUE** to add the 25.th and 75.-th percentile into the table.

If we tell it that we do not need a summary (**summary=FALSE**), the table itself will be produced:

```
stargazer(piaac[1:3, 1:5], summary = FALSE, rownames = FALSE, digits = 0,
    header = FALSE)
```

X	seqid	age	gender	empl_status
1	1,512	64	Male	Out of the labour force
2	7,513	30	Female	Employed
3	4,593	61	Female	Out of the labour force

And if you want to produce some nice regression tables, you can again use stargazer:

```
piaac$logincome <- log(piaac$earnhr)
regression1 <- lm(logincome ~ pvnum1 + age, data = piaac)
regression2 <- lm(logincome ~ pvnum1 + pvlit1 + age, data = piaac)
```

```
stargazer(regression1, regression2)
```

% Table created by stargazer v.5.2 by Marek Hlavac, Harvard University. E-mail: hlavac at fas.harvard.edu % Date and time: R, veebr 16, 2018 - 13:27:33

	Dependent variable:	
	logincome	
	(1)	(2)
pvnum1	0.004^{***}	0.005^{***}
	(0.0002)	(0.0004)
pvlit1		-0.001^{***}
		(0.0004)
age	-0.004^{***}	-0.005^{***}
	(0.001)	(0.001)
Constant	0.563^{***}	0.666^{***}
	(0.072)	(0.077)
Observations	4,052	4,052
R^2	0.086	0.089
Adjusted R^2	0.085	0.088
Residual Std. Error	0.594 (df = 4049)	0.593 (df = 4048)
F Statistic	189.522^{***} (df = 2; 4049)	131.437^{***} (df = 3; 4048)
Note:		$^{*}p<0.1$; $^{**}p<0.05$; $^{***}p<0.01$