Foundamentals of R for univariate and bivariate descriptive analysis

S.I.De.S Basic course in R for Historical Demography

Saverio Minardi

2023-06

Contents

Introduction

This document is an introduction to the basics of programming for data analysis in R. It will go through the various necessary steps of any data analysis project in R. Starting from data acquisition, data inspection, data cleaning, and basic univariate and bivariate analysis as well as their optimal graphical representation.

R is a programming language and software environment primarily used for statistical computing and graphics. The basic version of R provides a wide variety of statistical and graphical techniques, these are called built in functions. The built-in function in the R programming language is the functions that are already existing or pre-defined within an R framework.

The present document will almost exclusively use built in functions.

However, it is important to know that one of the key strengths of R is its extensive collection of packages. R packages are sets of functions, data, and documentation that extend the capabilities of the base R system. These packages are developed by the R community and cover various areas of data analysis and specialized domains, such as bioinformatics, finance, and social sciences.

Install and load packages

Even thought we will not use packages in this session, knowing how to install and load a package is one of the most important tasks in R.

Installing and *loading* a package in R are the two necessary steps for utilizing the functions provided by a specific package.

To install a package in R, you need to download and install it from a repository. To install a package, you can use the *install.packages()* function followed by the name of the package. For example, to install the "tidyverse" package, you would write:

```
install.packages("tidyverse")
```

This command will download the package files from the CRAN repository and install them on your computer. You typically only need to install a package once, unless you want to update it to a newer version.

Once a package is installed, you need to load it into your current R session to access its functions, data, and other resources. To load a package, you can use the *library()* or *require()* function followed by the name of the package. For example, to load the "ggplot2" package, you would run:

```
library("tidyverse")
```

Import data

The first step to any data analysis is to import the data frame we want to analyse in our R environment. Data are usually stored in a folder of our computer. We call the folder in our computer where we store data, files, scripts, and other resources, a directory.

Set the working directory

Before we start working we want to inform R that all the files we need are stored in this directory and that all the output it will produced will be saved in that folder. We do this through the command:

```
setwd("C:/Users/PC/Documents/Sides_R_intro")
```

Inside the brackets is the path to the folder in my computer where the data are stored. You should write in there the path of the folder in your computer.

You can check to which folder your directory is currently set at any time by typing:

```
getwd()
```

```
## [1] "C:/Users/save_/Dropbox/PC/Documents/Sides_R_intro"
```

Once we have set the directory we can import our data.

Import data

There are different commands for importing the data depending on the format in which the data are stored. In this case our data are stored as a ".csv" and the command to import them is read.csv ("name_of_the_file.csv").

We also want to store the data as an object. In this case we will call it "cens".

The overall command to import the data is:

```
cens <- read.csv("Data/ipums_cens_subs.csv")
```

Now in the R environment you can see an object named "cens". Those are our data.

As mentioned, the command depends on the format of the files. Here are other common examples:

- *read.csv()* for .csv;
- *read.dta()* for .dta which are STATA type of data;
- *read.xlsx()* for .xlsx ;

These data are a random sample of 20000 individuals from the full population of New York in 1880 extracted from the full count censuses provided by IPUMS. (*Steven Ruggles, Sarah Flood, Matthew Sobek, Danika Brockman, Grace Cooper, Stephanie Richards, and Megan Schouweiler. IPUMS USA: Version 13.0 [dataset]. Minneapolis, MN: IPUMS, 2023. https://doi.org/10.18128/D010.V13.0).*

The full data frame are accessible upon request at https://usa.ipums.org/usa/

The attached data file is intended only for assignments as part of Sides R course. Individuals are not to redistribute the data without permission.

Inspect data

The first thing we want to do when we import the data is to have a general look at them to asses their structure and whether they have been imported correctly.

There are several ways to do this. The first and most general is to simply *browse* them and visualize as a spreadsheet:

```
View(cens)
```

Views() shows the full data frame in a separate window but does not produce any output.

A second way is *str()* which prints a compact description of the number of rows, columns, and all variables contained in the data with their variable type and first observations .

```
str(cens)
```

```
## 'data.frame':    20000 obs. of  10 variables:
##  $ id      : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ serial  : int  6444975 6314961 6273065 6373486 6303493 6289586 6386406 6449576 6350576 6405714 ..
##  $ relate  : int  3 7 10 1 1 12 12 10 3 3 ...
##  $ sex     : int  1 2 1 1 1 2 2 1 2 1 ...
##  $ age     : int  17 30 8 35 63 70 54 35 4 2 ...
##  $ birth_pl: chr  "US" "Ireland" "US" "Ireland" ...
##  $ nchild  : int  0 0 0 3 0 0 0 0 0 0 ...
##  $ eldch   : int  99 99 99 9 99 99 99 99 99 99 ...
##  $ marst   : chr  "Single" "Single" "Single" "Married" ...
##  $ fams    : int  7 7 10 5 2 1 1 7 3 8 ...
```

A third way is *head()*. The *head()* function shows the first few rows of your data frame. It allows you to quickly examine the data and get a sense of its contents. By default, it displays the first six rows, but you can specify the number of rows to display with the option *n()*. For example:

```
head(cens, n=10)
```

```
##    id  serial relate sex age birth_pl nchild eldch            marst fams
## 1   1 6444975      3   1  17       US      0    99           Single    7
## 2   2 6314961      7   2  30  Ireland      0    99           Single    7
## 3   3 6273065     10   1   8       US      0    99           Single   10
## 4   4 6373486      1   1  35  Ireland      3     9          Married    5
## 5   5 6303493      1   1  63  Germany      0    99 Divorced/widowed    2
## 6   6 6289586     12   2  70       UK      0    99 Divorced/widowed    1
## 7   7 6386406     12   2  54       US      0    99           Single    1
## 8   8 6449576     10   1  35  Ireland      0    99           Single    7
## 9   9 6350576      3   2   4       US      0    99           Single    3
## 10 10 6405714      3   1   2       US      0    99           Single    8
```

Finally, if we only want to know the names of all variables in our data, we simply write:

```
names(cens)
```

```
##  [1] "id"       "serial"   "relate"   "sex"      "age"      "birth_pl"
##  [7] "nchild"   "eldch"    "marst"    "fams"
```

In this case the variables represent:

- "id" : unique identification number for each individual
- "serial": household identification number

- "relate": relationship to the household head
- "sex": sex of the respondent
- "age": age of the respondent
- "birth_pl": birth place of the respondent
- "nchild": number of children of the respondent
- "eldch": respondendt's eldest child in the household
- "marst": marital status
- "fams": size of the family

Data cleaning

Most data is not provided in the precise format you need for your analysis. Before analyzing the data, we usually need to prepare them by structuring, cleaning, adding features and so on... Here we will see some of the most common commands.

Rename variables

Variables are not always provided with the most intuitive names. There are more ways to changing their names.

"*colnames()*" refers to the names of all variables in a data frame:

```
colnames(cens)
```

```
##  [1] "id"       "serial"   "relate"   "sex"      "age"      "birth_pl"
##  [7] "nchild"   "eldch"    "marst"    "fams"
```

We can access and change them in many ways.

First, we can refer to the column number we want to change using [ ].

In R, the square brackets [ ] are used to subset or extract elements from vectors, matrices, or data frames. The [ ] notation allows you to specify one or more indices or conditions to retrieve specific elements or subsets of your data. For example cens[1] is the first column of the data frame "cens". *colnames(cens)[1]* is the first column names. So we can change it by typing:

```
colnames(cens)[1] <- "person_id"
```

Second, we can refer to the existing column name:

```
colnames(cens)[colnames(cens) == "person_id"] <- "id"
```

Access variables in a data frame

To access variables in a data frame in R, you can use the $ operator. The $ operator is used by specifyng the variable name directly after a data frame name, separated by the $ symbol.

In other words, the $ symbol means that we are referring to a vector within a data frame. It is necessary to distinguish variables in a data frame from any other object in the environment.

For example, if we want to tabulate the variable "sex" contained in the data "cens" we write:

```
table(cens$sex, useNA = "always")
```

```
##
##     1     2  <NA>
##  9882 10118     0
```

Replace values

Sometimes, data frame contain missing values, outliers, or erroneous entries.

There is more than one way to replace specific values of a variable in R. The most straightforward using built in commands uses [ ] to refer to those values we wish to change. For instance, in the variable "eldch" the value 99 is the indication of "no children". We might want to assign a missing value (NA) to those whit 99. We can do so in this way:

```
cens$eldch[cens$eldch == 99] <- NA
```

We can also use the *ifelse* command:

```
cens$eldch <- ifelse(cens$eldch == 99, # If 99 is found in the ver eldch
                     NA,               # Replalce with NA
                     cens$eldch)       # Else replace with existing eldch value
```

If the first condition within the brackets is satisfied (*cens$eldch == 99)*, than the value is replaced with the first term after the comma (*NA*), if it is not then it is replaced with the third entry after the comma ("eldch").

Create a new variable

To create a new variable, we simply name it and assign the values that we wish. For instance to create a new variable of all NAs we write:

```
cens$new_var <- NA
```

We can use any function we want to create new variables.

Drop variables

To remove variables from the data frame we can use the function *subset()*.

```
cens <- subset(cens, select = -c(new_var))
```

The code above assign to the data frame "cens" a new version of the data frame without the indicated variable.

Alternatively, we can write:

```
cens$new_var<-NULL
```

Drop observations

To drop all females (keep all males):

```
cens_males <-cens[cens$sex==1,]
```

Here we created a new data frame named "cens_males" which is equal to "cens" less all the rows which do not have a value of 1 for the variable "sex". We put a "," to indicate that we are referring to rows.

Missing data

We can use the *is.na()* function to identify data with "NA" values. The following example demonstrates how the function works.

The output of *is.na()* is "TRUE" or "FALSE" to designate whether each observation is an "NA" value. If the inpute of *is.na()* is a vector, such as a variable, *is.na()* returns a vector equal to the length of the variable indicating T (true) if the observation is missing and F (false) if it's not.

By reporting it in a table we can see the number of missing values:

```
table(is.na(cens$eldch))
```

```
##
## FALSE  TRUE
##  5996 14004
```

If we want to quiclky check for missing in all variables, we can compute the sum of all missing by column:

```
colSums(is.na(cens))
```

```
##       id   serial   relate      sex      age birth_pl   nchild    eldch
##        0        0        0        0        0        0        0    14004
##    marst     fams
##        0        0
```

or the share, by computing the mean by column (T and F are treated as 1 and 0)

```
colMeans(is.na(cens))
```

```
##       id   serial   relate      sex      age birth_pl   nchild    eldch
##   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.7002
##    marst     fams
##   0.0000   0.0000
```

Univariate descriptive analysis

A description of all quantitative variables in the data frame can be done using *summary()*:

```
summary(cens)
```

```
##        id            serial            relate            sex
##  Min.   :    1   Min.   :6198393   Min.   : 1.000   Min.   :1.000
##  1st Qu.: 5001   1st Qu.:6268782   1st Qu.: 2.000   1st Qu.:1.000
##  Median :10000   Median :6335148   Median : 3.000   Median :2.000
##  Mean   :10000   Mean   :6336679   Mean   : 3.618   Mean   :1.506
##  3rd Qu.:15000   3rd Qu.:6401622   3rd Qu.: 3.000   3rd Qu.:2.000
##  Max.   :20000   Max.   :6477378   Max.   :13.000   Max.   :2.000
##
##       age           birth_pl            nchild           eldch
##  Min.   :  0.00   Length:20000       Min.   :0.0000   Min.   : 0.00
##  1st Qu.: 11.00   Class :character   1st Qu.:0.0000   1st Qu.: 7.00
##  Median : 24.00   Mode  :character   Median :0.0000   Median :14.00
##  Mean   : 25.62                      Mean   :0.8757   Mean   :15.18
##  3rd Qu.: 38.00                      3rd Qu.:1.0000   3rd Qu.:22.00
##  Max.   :116.00                      Max.   :9.0000   Max.   :68.00
##                                                       NA's   :14004
##     marst                fams
##  Length:20000       Min.   : 1.000
##  Class :character   1st Qu.: 3.000
##  Mode  :character   Median : 5.000
##                     Mean   : 5.251
##                     3rd Qu.: 7.000
##                     Max.   :47.000
##
```

Summary also works for single variables or list of variables:

```
summary(cens$eldch)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    0.00    7.00   14.00   15.18   22.00   68.00   14004
```

Alternatively you can use functions to estimate the measure you are interested in.The most common functions are:

Measures of central tendency

```
mean(cens$age)
```

```
## [1] 25.619
```

```
median(cens$age)
```

```
## [1] 24
```

Measures of dispersion

```
sd(cens$age)
```

```
## [1] 17.51085
```

```
var(cens$age)
```

```
## [1] 306.6298
```

```
range(cens$age)
```

```
## [1]   0 116
```

```
min(cens$age)
```

```
## [1] 0
```

```
max(cens$age)
```

```
## [1] 116
```

You can store any of these values in an object:

```
mn_age<-mean(cens$age)
sd_age<-sd(cens$age)
```

and use it for operations or creating new variables. For instance, we can create the age standardized:

```
cens$age_stz<- (cens$age - mn_age)/sd_age
summary(cens$age_stz)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -1.46304 -0.83485 -0.09246  0.00000  0.70705  5.16143
```

```
sd(cens$age_stz)
```

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

We can also combine any of these output to produce descriptive tables

```
summary_table <- data.frame(
  Variable = c("age", "nchild"),
  Mean = c(mean(cens$age), mean(cens$nchild)),
  Median = c(median(cens$age), median(cens$nchild)),
  SD = c(sd(cens$age), sd(cens$nchild))
)

# Print the summary table
print(summary_table)
```

```
##   Variable     Mean Median        SD
## 1      age 25.61900     24 17.510847
## 2   nchild  0.87565      0  1.671983
```

We can export this table directly to an excel file or a word file:

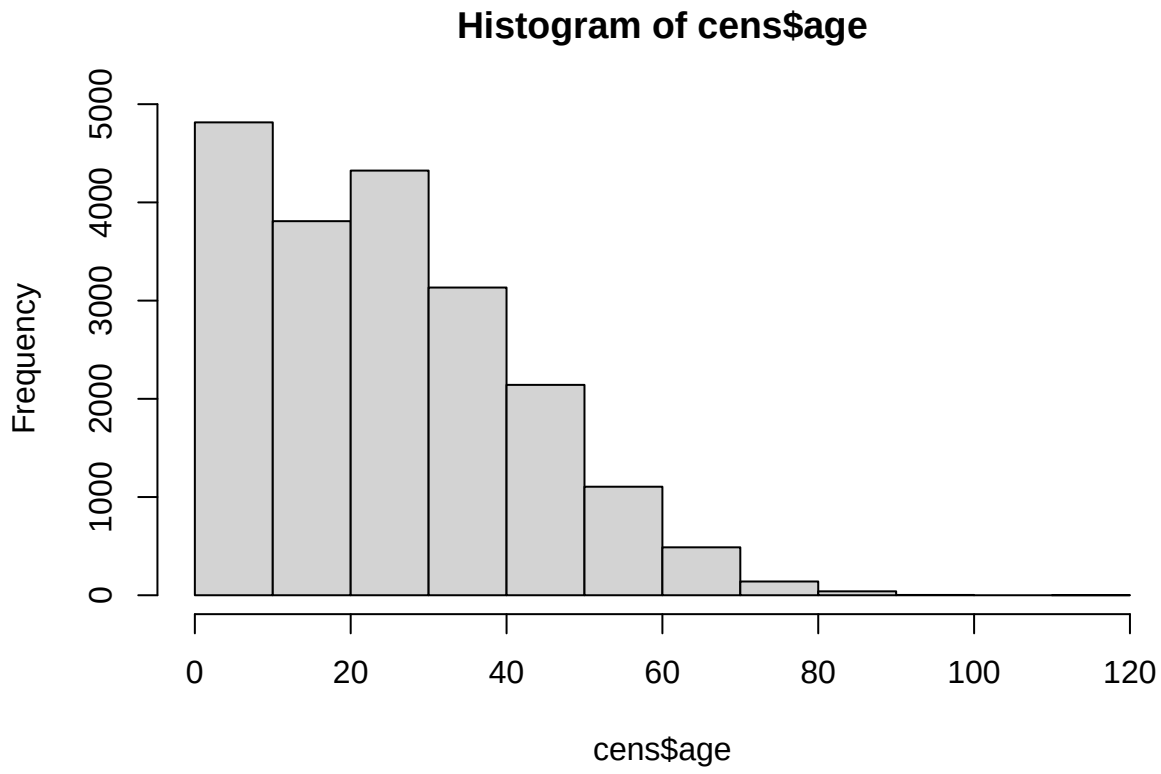
```
library(writexl)
library(officer)

# Export to Excel
write_xlsx(summary_table, "summary_table.xlsx")

# Export to Word
doc <- read_docx()
doc <- body_add_table(doc, summary_table)
print(doc, target = "summary_table.docx")
```

This is just one way to export descriptive statistics many packages offer other solutions.

Graphical representations for numerical variables

Histogram:

```
hist(cens$age)
```

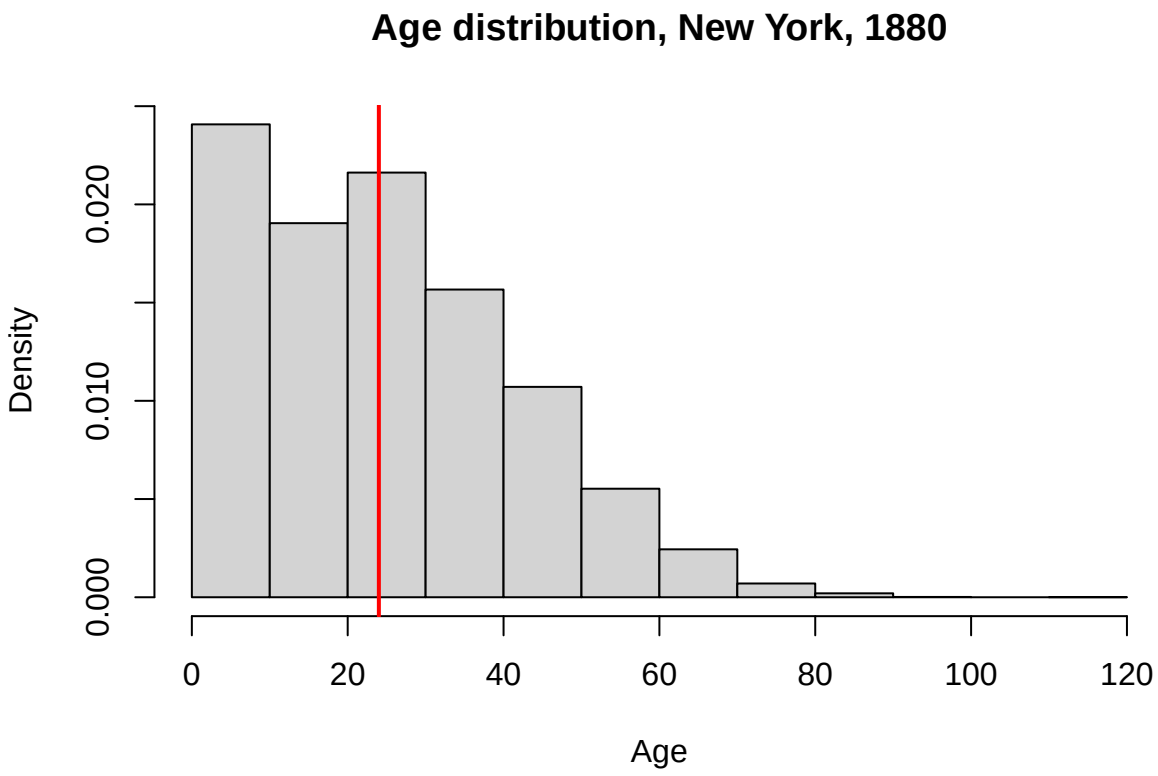


We can adjust titles and labels of the graph:

```
hist(cens$age, xlab="Age", main="Age distribution, New York, 1880", freq = FALSE)
```

And we can add graphic elements, such as a line indicating the median or any other value:

```
hist(cens$age, xlab="Age", main="Age distribution, New York, 1880", freq = FALSE)
abline(v = median(cens$age), col = "red", lwd = 2)
```



The graph cab be stored in an object and we can easily export it:

```
#Set size of the image and text in pixels
png("hist.png", width = 1200, height = 800, pointsize = 20)

#Plot
hist(cens$age,
     xlab="Age",
     main="Age distribution, New York, 1880",
     freq = FALSE)

#Add reference line for media
abline(v = median(cens$age), col = "red", lwd = 2)

# Close the device
dev.off()
```

```
## pdf
##   2
```

Box-plot

```
boxplot(cens$age,
     ylab="Age",
```

```
     main="Age distribution, New York, 1880",
     freq = FALSE)
```

Qualitative variables

The summary statistics that we have seen above are for continuous variables. In case our variables are categorical, we want to tabulate the frequency and percentages of each category.

First, we must table the frequency:

```
table(cens$birth_pl)
```

```
##
## Germany Ireland   Italy  Others      UK      US
##    2728    3269     197    1163     607   12036
```

If we want to include missing values as a category in our table we can do so with the option useNA, like this:

```
table(cens$birth_pl, useNA="always")
```

```
##
## Germany Ireland   Italy  Others      UK      US    <NA>
##    2728    3269     197    1163     607   12036       0
```

Once again we can store the output of our command in an object

```
tab_mig<-table(cens$birth_pl)
```

If we want to tabulate the proportion of each category, we can do it with "prop.table":

```
prop.table(tab_mig)
```

```
##
## Germany Ireland   Italy  Others      UK      US
## 0.13640 0.16345 0.00985 0.05815 0.03035 0.60180
```

```
tab_mig_prop<-prop.table(tab_mig)
```

We can combine the two to produce a final table with absolute and relative frequencies:

```
freq_mig<-cbind(tab_mig, tab_mig_prop)
colnames(freq_mig)<-c("N", "Share")
print(freq_mig)
```

```
##             N   Share
## Germany  2728 0.13640
## Ireland  3269 0.16345
## Italy     197 0.00985
## Others   1163 0.05815
## UK        607 0.03035
## US      12036 0.60180
```

Plot qualitative variables

The most common way to plot a qualitative variable is a barplot. To do so we must first produce the frequency or percentage table:

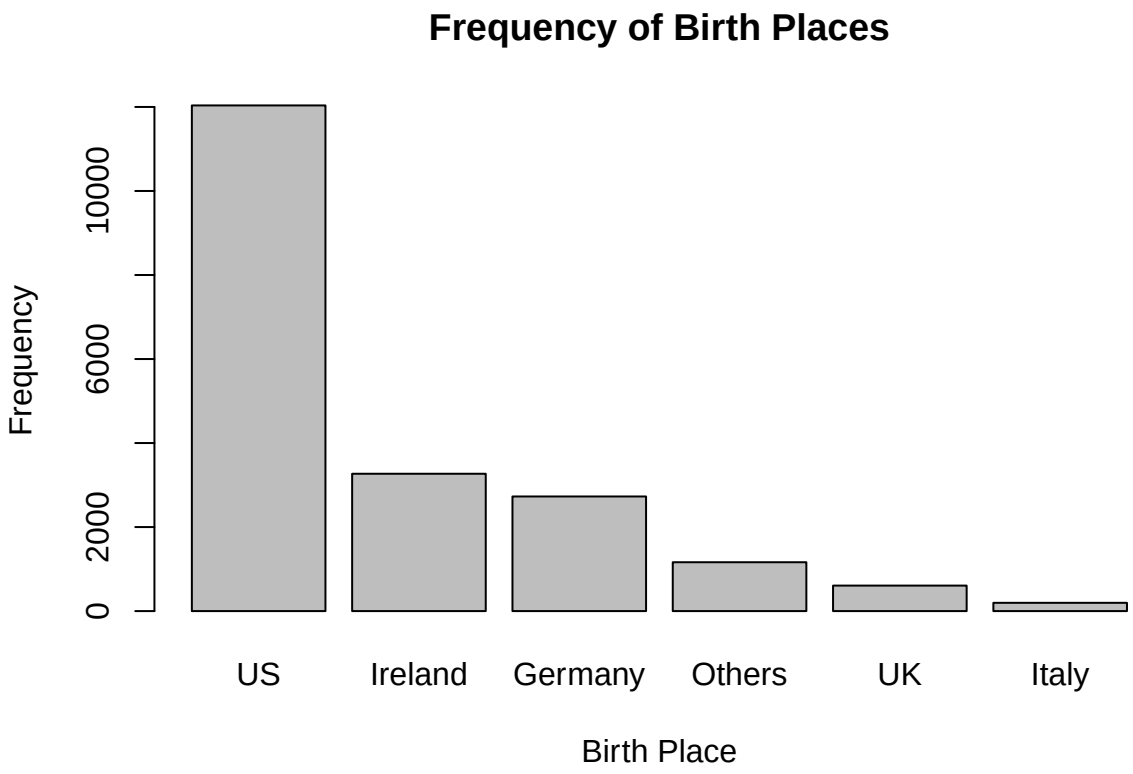
```
freq <- table(cens$birth_pl)
```

Second, if we want the bars sorted based on the frequency we must sort the table based on that value. We can do it using the function *sort()* creating a new sorted table names*:*

```
sorted_freq <- sort(freq, decreasing = TRUE)
```

Finally, we can plot with the command *barplot()*. We can add details with the various options:

```
barplot(sorted_freq,
        xlab = "Birth Place",
        ylab = "Frequency",
        main = "Frequency of Birth Places")
```



Two-way descriptive analysis

Categorical and continuous variables

Let's descriptively investigate the relationship between country of origin and family structures. We use two variables that we have already checked before: "birth_pl" and "nchild".

We want to compute the mean number of children for each origin group but first we want to keep only females 18 to 30 years old.

First we select the sample for our analysis:

```
cens_fems<-cens[cens$sex==2 & cens$age>=18 & cens$age<=30,]
```

Second, we computer the group means with the command *aggregate()*. Through *aggregate()* we create a new object names mean group which computes the mean of "nchild" across categories of "birth_pl" from the data frame "cens_fems".

```
#Compute group means
mean_by_group <- aggregate(nchild ~ birth_pl, data = cens_fems, FUN = mean)
```

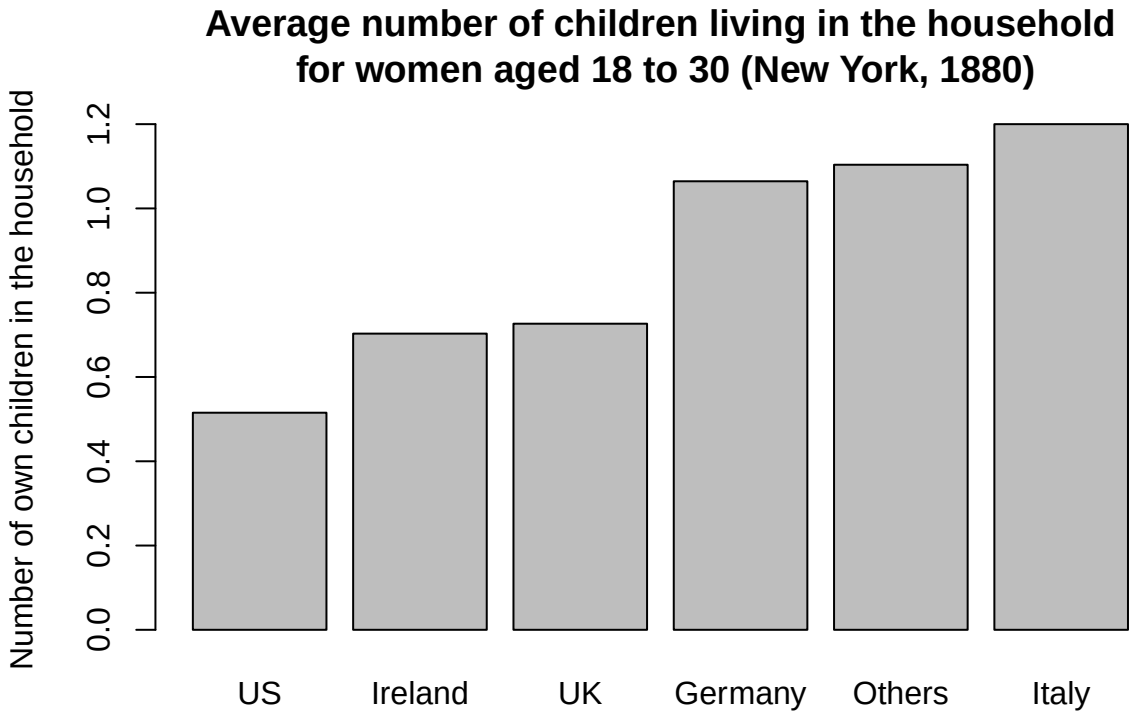
Finally, we sort the categories based on the value of interest and we print it:

```r
mean_by_group<- mean_by_group[order(mean_by_group$nchild), ]

print(mean_by_group)
```

```
##   birth_pl    nchild
## 6       US 0.5151689
## 2  Ireland 0.7028862
## 5       UK 0.7264151
## 1  Germany 1.0644172
## 4   Others 1.1036585
## 3    Italy 1.2000000
```

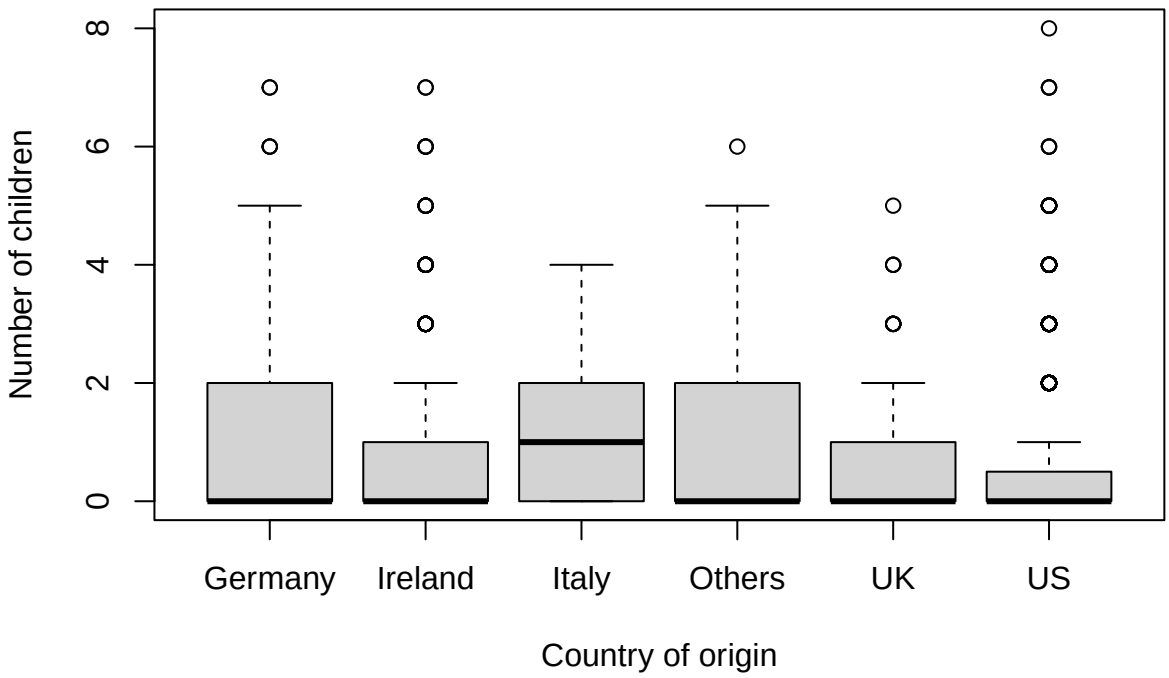
We can plot this relationship with a simple barplot:

```r
barplot(names.arg = mean_by_group$birth_pl,
        mean_by_group$nchild,
        ylab = "Number of own children in the household",
        main = "Average number of children living in the household\n for women aged 18 to 30 (New York,
```



Or with a two-way boxplot from the micro data

```r
boxplot(cens_fems$nchild ~ cens_fems$birth_pl,
        ylab = "Number of children",
        xlab = "Country of origin")
```



Categorical and categorical variables

Let's investigate the relationship between country of origin and marital status among men. Country of origins is a categorical variable taking four possible categories, marital status three: divorced/widowed, married, single.In the case of two categorical variables we use two-ways contingency tables.

First we limit our sample to males aged 30 to 50 years of age:

```
cens_mal<-cens[cens$sex==1 & cens$age>=30 & cens$age<=50,]
```

Second, we plot the two-way absolute frequency table. We can do this with the command table:

```
table(cens_mal$marst, cens_mal$birth_pl)
```

```
##
##                   Germany Ireland Italy Others  UK  US
##   Divorced/widowed      25      41     3     13   5  40
##   Married              645     583    41    226  96 606
##   Single               112     181    13     62  37 258
```

We can percentualise this in any direction using the command *prop.table()*. The option *margin* indicate whether the percentualisation has to be done by column, rows, or over the total. *margin=1* indicates rows and *margin=2* indicate columns.

```
prop.table(table(cens_mal$marst, cens_mal$birth_pl), margin=2)
```

```
##
##                       Germany    Ireland      Italy     Others         UK
##   Divorced/widowed 0.03196931 0.05093168 0.05263158 0.04318937 0.03623188
##   Married          0.82480818 0.72422360 0.71929825 0.75083056 0.69565217
##   Single           0.14322251 0.22484472 0.22807018 0.20598007 0.26811594
##
##                           US
##   Divorced/widowed 0.04424779
##   Married          0.67035398
##   Single           0.28539823
```

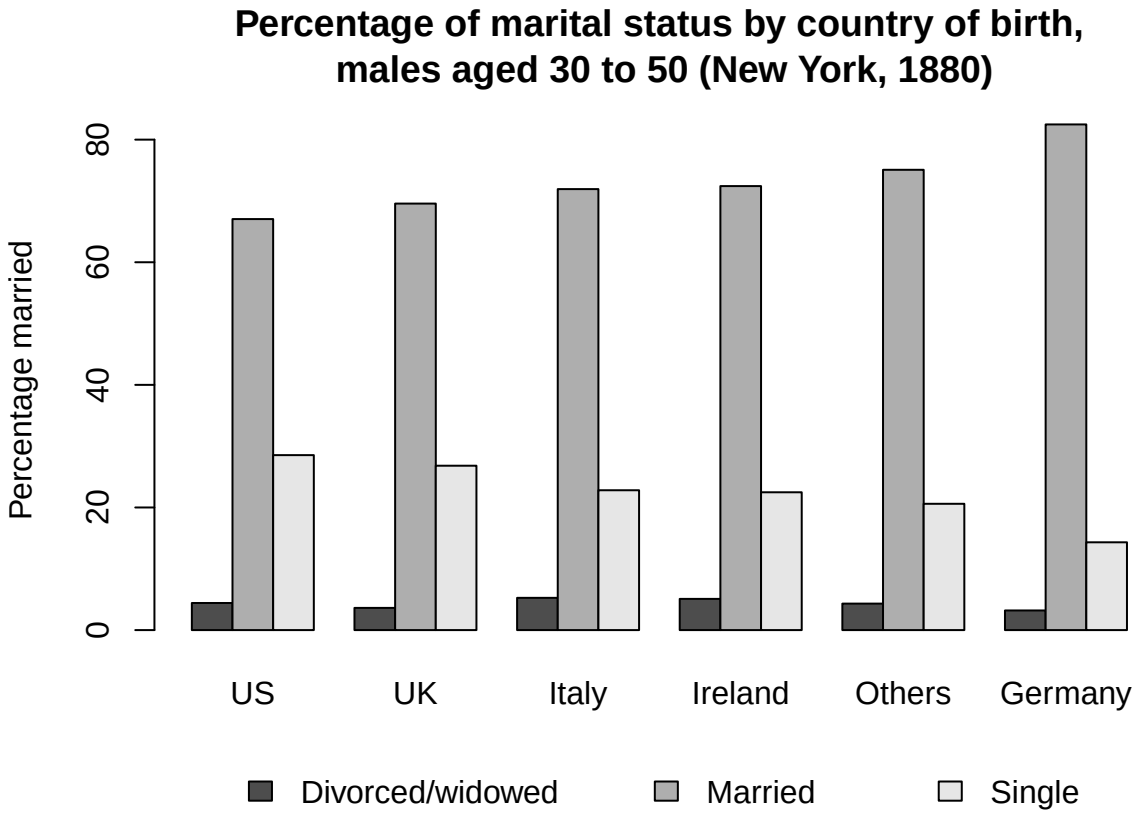
We can round the values, transform in percentages, and sort:

```
cont_table<-prop.table(table(cens_mal$marst, cens_mal$birth_pl), margin=2)

cont_table<-round(cont_table*100, 2)

cont_table <- cont_table[, order(cont_table["Married", ])]

cont_table
```

```
##
##                       US    UK Italy Ireland Others Germany
##   Divorced/widowed  4.42  3.62  5.26    5.09   4.32    3.20
##   Married          67.04 69.57 71.93   72.42  75.08   82.48
##   Single           28.54 26.81 22.81   22.48  20.60   14.32
```

Finally we can barplot:

```
# Plot the contingency table
barplot(cont_table, beside = TRUE, legend = TRUE,
        xlab = "",
        ylab = "Percentage married",
        main = "Percentage of marital status by country of birth,\n males aged 30 to 50 (New York, 1880)
       args.legend = list(xpd = TRUE,
                          x = 26,
                          y = -20,
                          bty= "n",
                          horiz=T,
                          text.width = c(6, 5, 5)))
```



Continuous and continuous variables

Let's try a different data frame. The Swiss data contains information on Swiss Fertility and Socioeconomic Indicators for 47 french speaking provinces (1888). It is found in the data frames R package. You can load it in R by issuing the following command at the console *data("swiss")*.

```
data("swiss")
```

As we have done before we can check the structure:

```
str(swiss)
```

```
## 'data.frame':    47 obs. of  6 variables:
##  $ Fertility       : num  80.2 83.1 92.5 85.8 76.9 76.1 83.8 92.4 82.4 82.9 ...
##  $ Agriculture     : num  17 45.1 39.7 36.5 43.5 35.3 70.2 67.8 53.3 45.2 ...
##  $ Examination     : int  15 6 5 12 17 9 16 14 12 16 ...
##  $ Education       : int  12 9 5 7 15 7 7 8 7 13 ...
##  $ Catholic        : num  9.96 84.84 93.4 33.77 5.16 ...
##  $ Infant.Mortality: num  22.2 22.2 20.2 20.3 20.6 26.6 23.6 24.9 21 24.4 ...
```

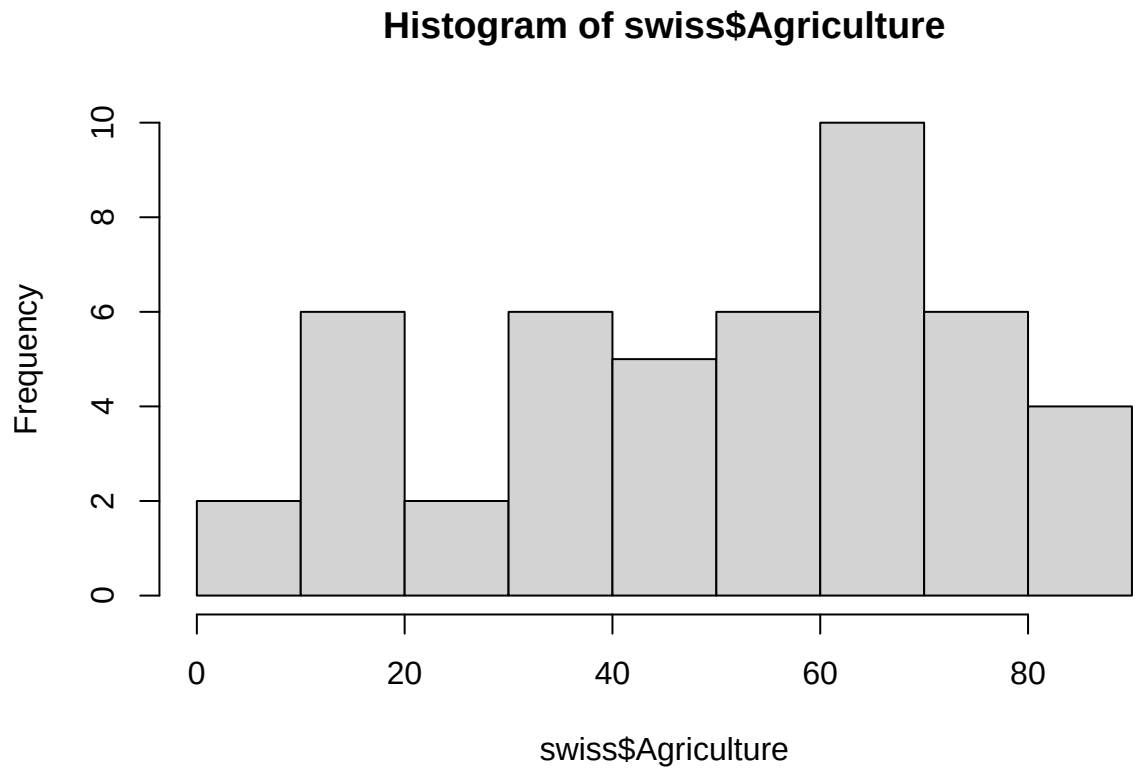
And the distribution of the variables:

```
summary(swiss)
```

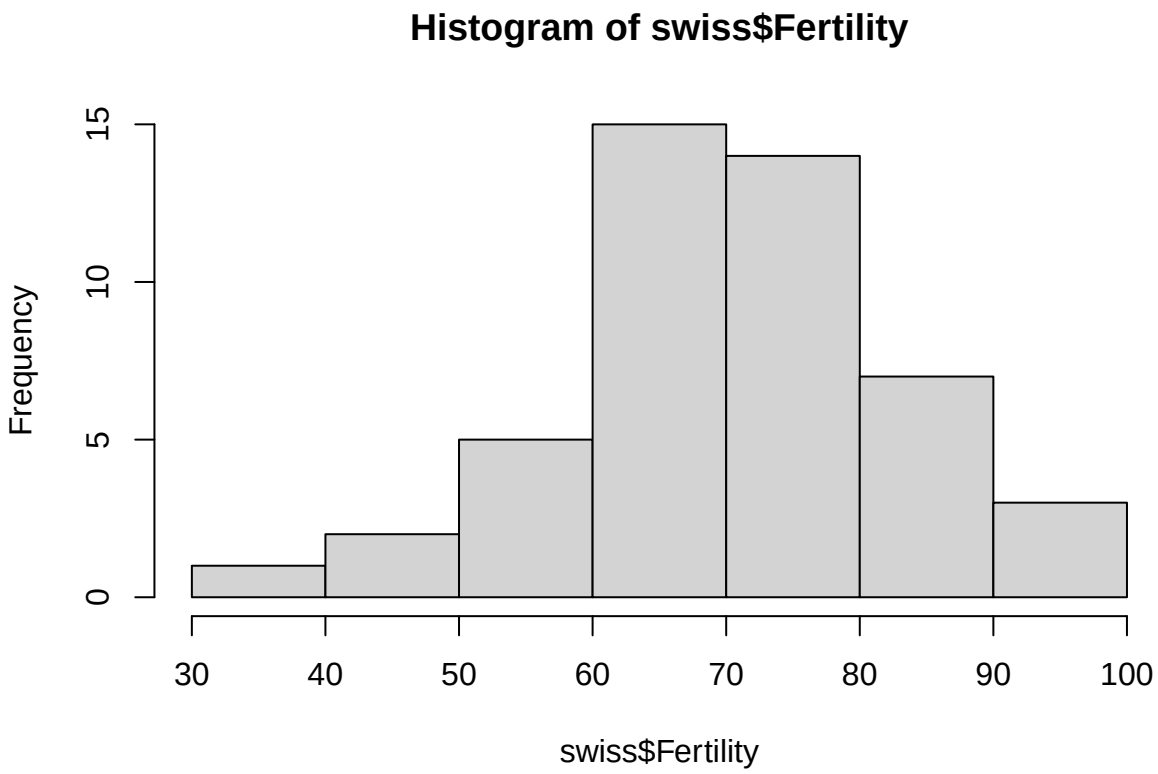
```
##    Fertility      Agriculture     Examination      Education
##  Min.   :35.00   Min.   : 1.20   Min.   : 3.00   Min.   : 1.00
##  1st Qu.:64.70   1st Qu.:35.90   1st Qu.:12.00   1st Qu.: 6.00
##  Median :70.40   Median :54.10   Median :16.00   Median : 8.00
##  Mean   :70.14   Mean   :50.66   Mean   :16.49   Mean   :10.98
##  3rd Qu.:78.45   3rd Qu.:67.65   3rd Qu.:22.00   3rd Qu.:12.00
##  Max.   :92.50   Max.   :89.70   Max.   :37.00   Max.   :53.00
##     Catholic       Infant.Mortality
##  Min.   :  2.150   Min.   :10.80
##  1st Qu.:  5.195   1st Qu.:18.15
##  Median : 15.140   Median :20.00
##  Mean   : 41.144   Mean   :19.94
##  3rd Qu.: 93.125   3rd Qu.:21.70
##  Max.   :100.000   Max.   :26.60
```

We can plot the single variables of interest:

```
hist(swiss$Agriculture)
```




```
hist(swiss$Fertility)
```



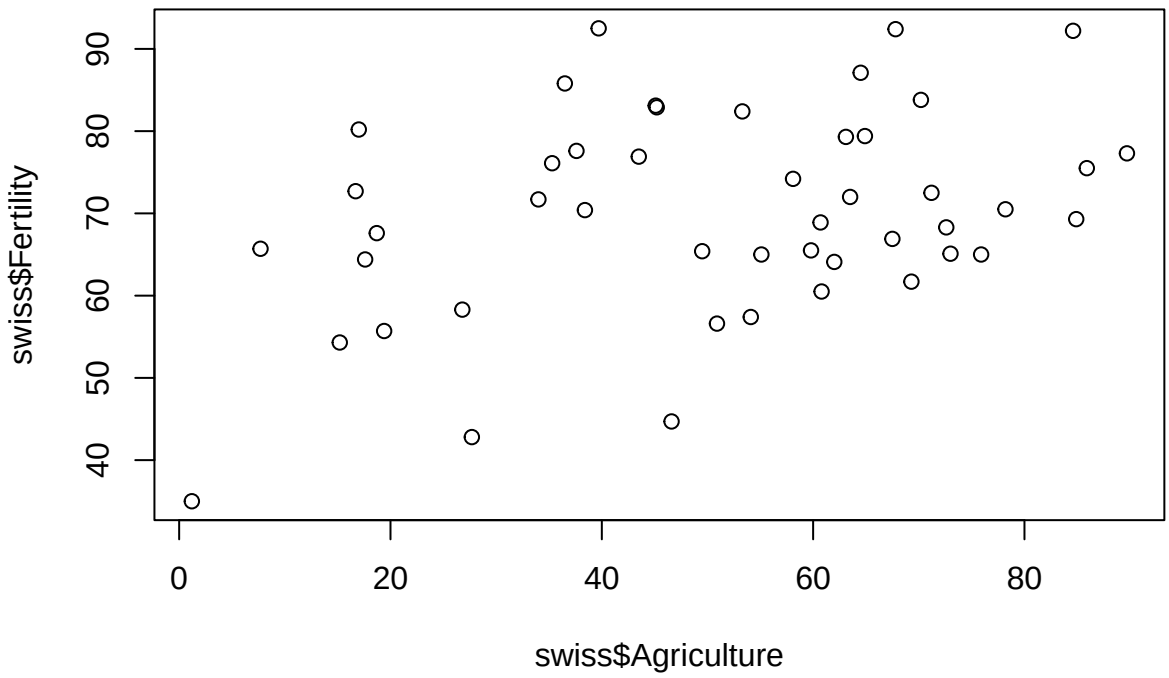
Given that we are interested in the relationship between modernization and fertility we check the correlation between people in agricolture and fertility rate with the command *cor():*

```
cor(swiss$Agriculture, swiss$Fertility, method="pearson" )
```

```
## [1] 0.3530792
```

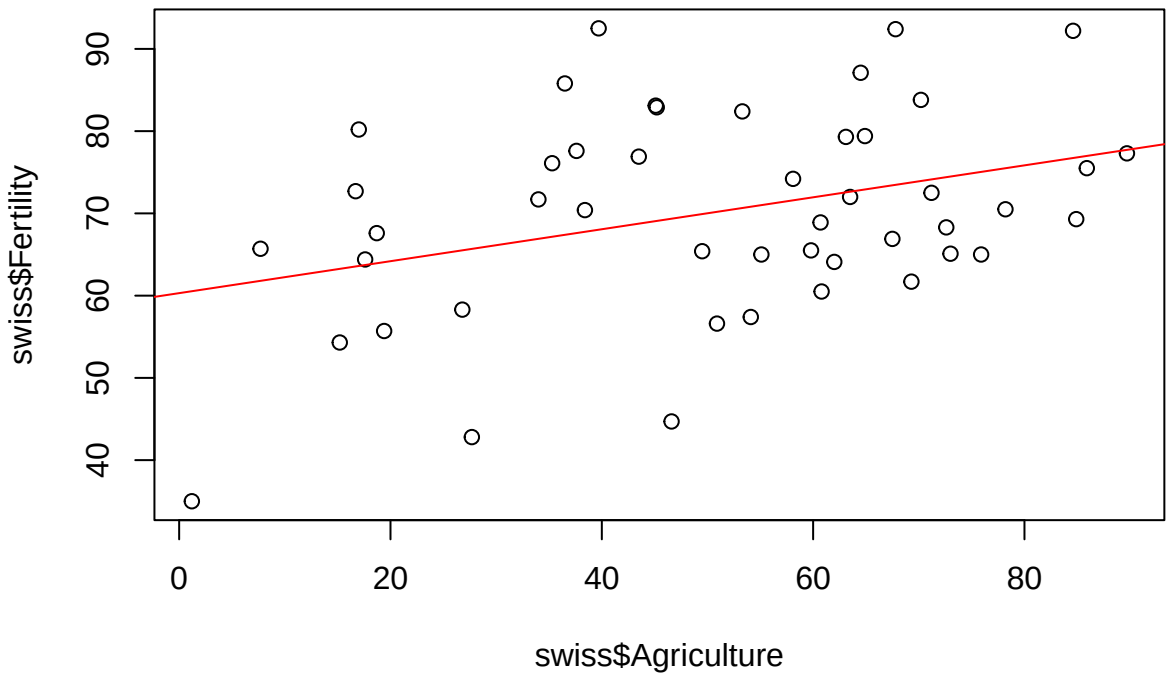
We can create a scatter-plot with the command *plot():*

```
plot(swiss$Agriculture, swiss$Fertility)
```



And even ad a fitted line:

```
plot(swiss$Agriculture, swiss$Fertility)
# Add a fitted line
fit <- lm(swiss$Fertility ~ swiss$Agriculture)  # Fit a linear regression model
abline(fit, col = "red")
```



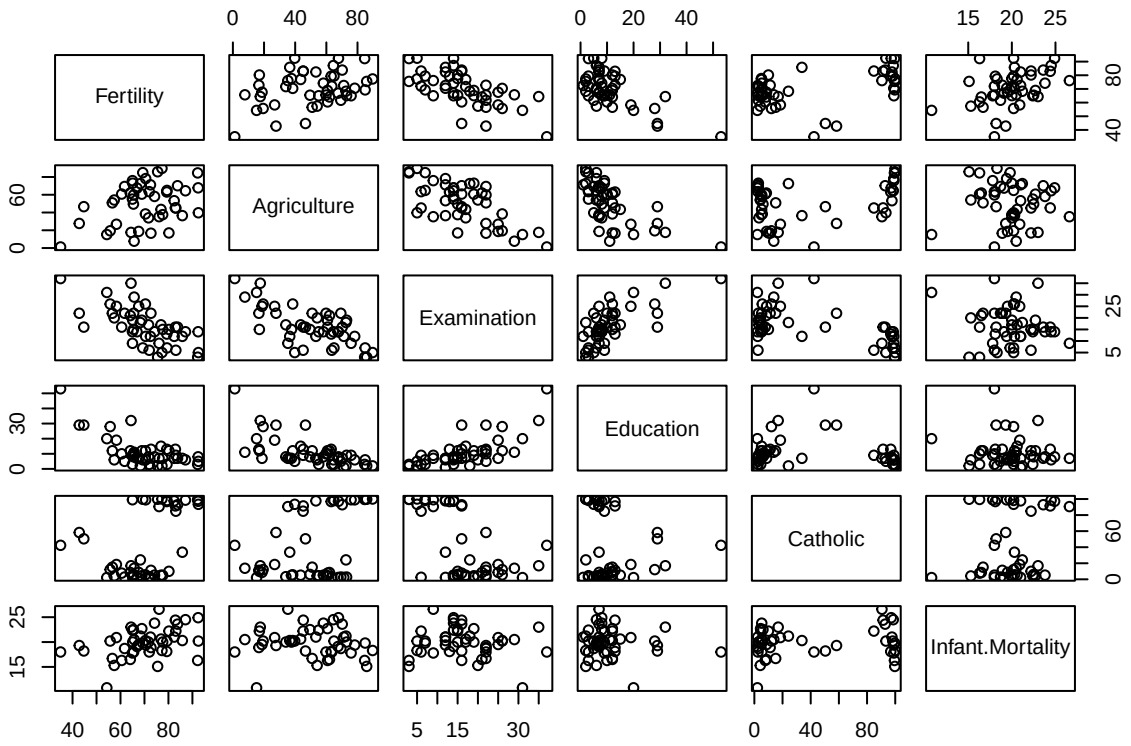
We can also check the full correlation matrix of all variables in the data frame using cor():

```
cor(swiss)
```

```
##                  Fertility Agriculture Examination   Education   Catholic
## Fertility         1.0000000  0.35307918  -0.6458827 -0.66378886  0.4636847
## Agriculture       0.3530792  1.00000000  -0.6865422 -0.63952252  0.4010951
## Examination      -0.6458827 -0.68654221   1.0000000  0.69841530 -0.5727418
## Education        -0.6637889 -0.63952252   0.6984153  1.00000000 -0.1538589
## Catholic          0.4636847  0.40109505  -0.5727418 -0.15385892  1.0000000
## Infant.Mortality  0.4165560 -0.06085861  -0.1140216 -0.09932185  0.1754959
##                  Infant.Mortality
## Fertility              0.41655603
## Agriculture           -0.06085861
## Examination           -0.11402160
## Education             -0.09932185
## Catholic               0.17549591
## Infant.Mortality       1.00000000
```

Finally, if we want to check the relationship between multiple numerical variables we can include a matrix in the command plot and it will return a scatter plot for each possible combination:

```
plot(swiss)
```

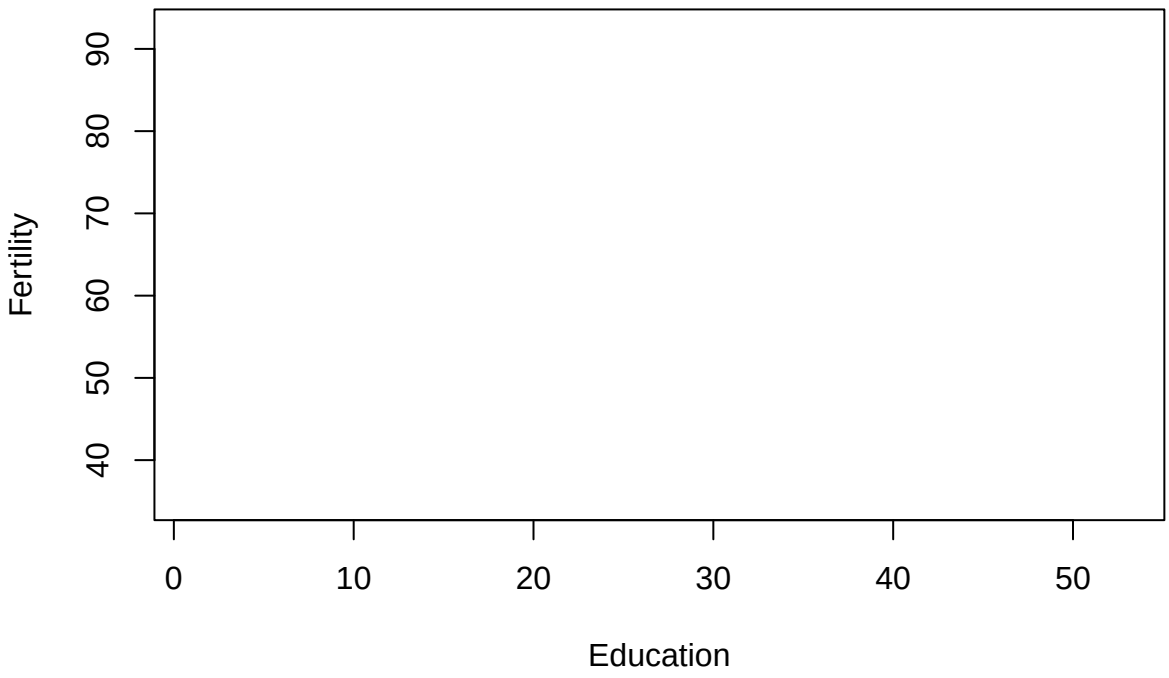


If we want to plot by categories we need to combine plots. For instance, let's investigate the relationship between education and fertility in catholic and protestant communities. First we generate a variable indicating the village is predominantly catholic or protestant:

```
swiss$relig<-ifelse(swiss$Catholic>=50, "Catholic", "Protestant")
```

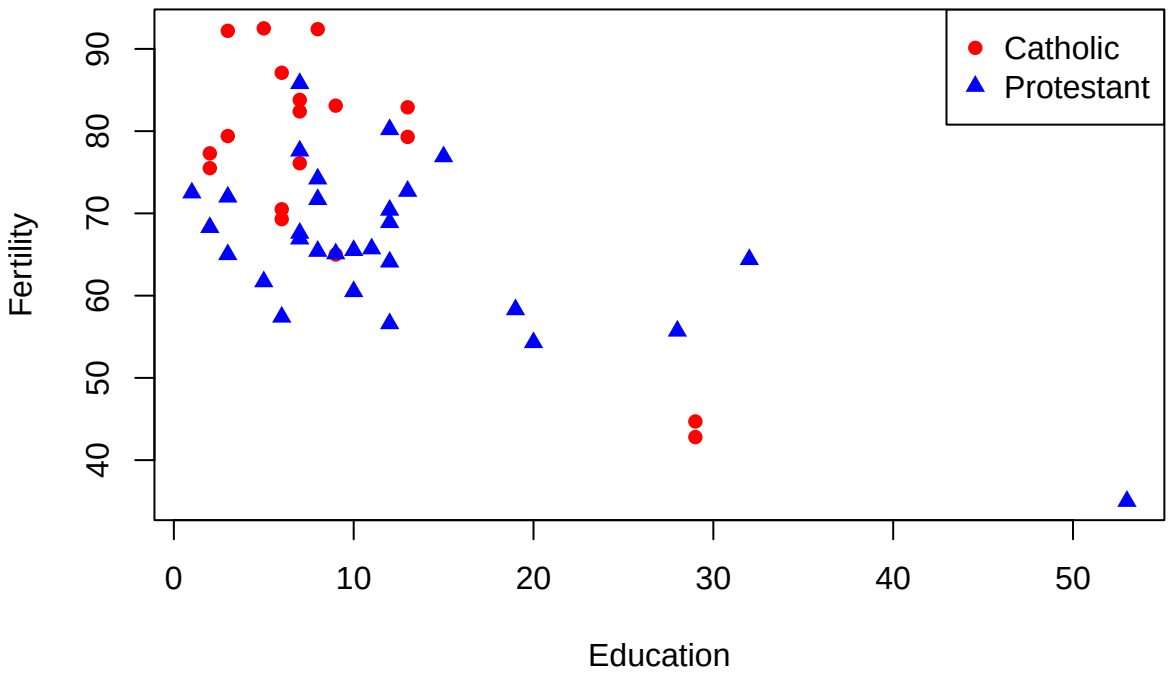
To plot by categories we first set and empty plot with axis ranges equal to the ranges of the two variables of interest:

```
plot(0, xlim = range(swiss$Education), ylim = range(swiss$Fertility),
        xlab = "Education", ylab = "Fertility")
```



Second, we fill the plot with the point separately for the two groups:

```
plot(0, xlim = range(swiss$Education), ylim = range(swiss$Fertility),
         xlab = "Education", ylab = "Fertility")

# Plot the scatterplot for the catholic in red
 points(swiss$Education[swiss$relig == "Catholic"],
        swiss$Fertility[swiss$relig == "Catholic"],
                              col = "red", pch = 16)

# Plot the scatterplot for the protestant in blue
 points(swiss$Education[swiss$relig == "Protestant"],
        swiss$Fertility[swiss$relig == "Protestant"],
                              col = "blue", pch = 17)

# Add the legend
    legend("topright", legend = c("Catholic", "Protestant"),
                           col = c("red",       "blue"),
                           pch = c(16, 17))
```



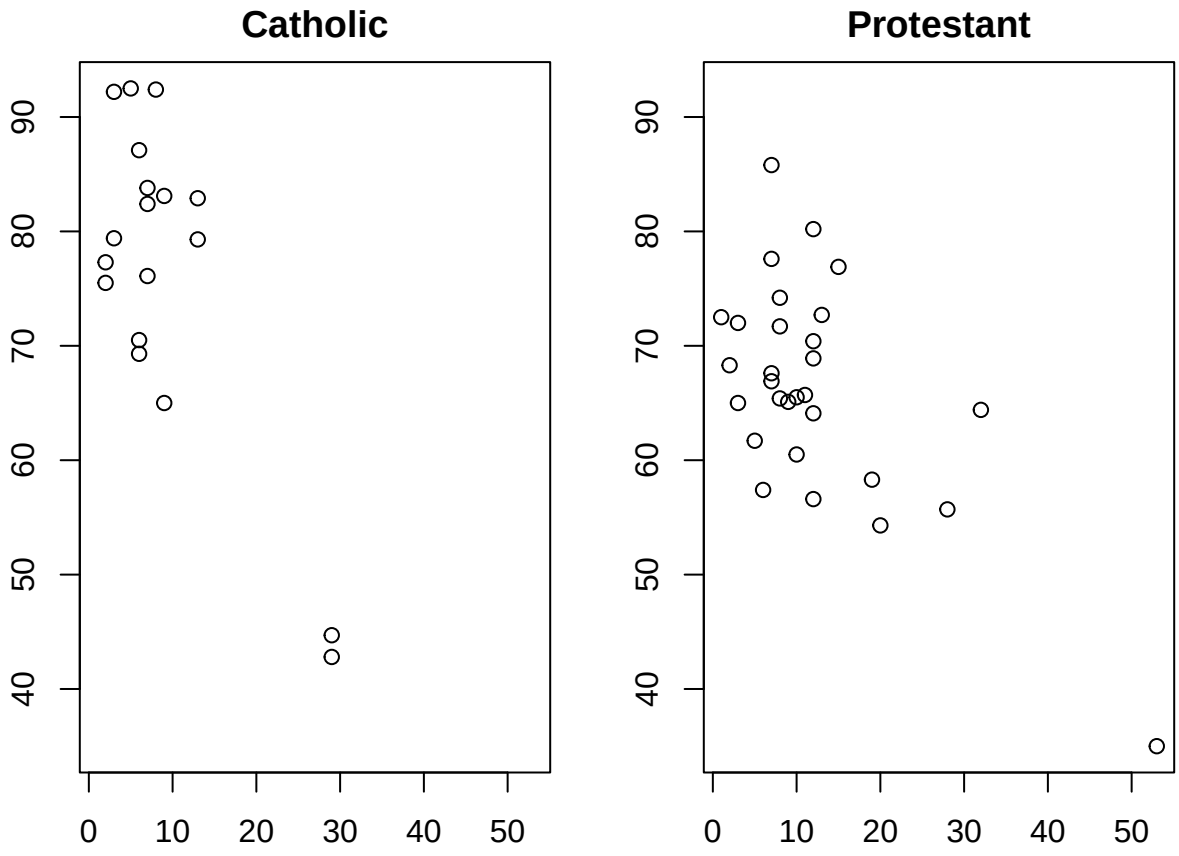
Combining multiple graphs

Sometimes we might want to combine multiple graphs in one single image. We do so by assigning proportions of the plot area with the function par(). par() sets the parameters of the plotting device. Specifically, you can set the mfrow parameter to specify the number of rows and columns in the layout. For example, par(mfrow=c(1,2)) sets the layout to have 1 rows and 2 column, meaning that the figure will have two plots horizontally.

We then produce the two plot and they are placed in order in the final plot device. After creating the graph, we set the layout back to the original format par(mfrow=c(1,1)).

```
par(mfrow=c(1,2),  #Set the row and columns
    mai = c(0.4, 0.4, 0.4, 0.4)) #Set the margins between graphs

# Plot the scatterplot for the catholic in red
 plot(swiss$Education[swiss$relig == "Catholic"],
      swiss$Fertility[swiss$relig == "Catholic"],
           xlim = range(swiss$Education),
           ylim = range(swiss$Fertility),
           xlab = "Education", ylab = "Fertility", main="Catholic")

# Plot the scatterplot for the protestant in blue
  plot(swiss$Education[swiss$relig == "Protestant"],
      swiss$Fertility[swiss$relig == "Protestant"],
           xlim = range(swiss$Education),
```

```
          ylim = range(swiss$Fertility),
          xlab = "Education", ylab = "Fertility", main="Protestant")
```




```
par(mfrow=c(1,1))
```

Useful materials open access

R is an open source software and there is a vibrant community contributing to its development. On line you can find extensive materials and discussions to improve your knowledge and solve problems. In blogs and Q&A websites you can find answers to basically any question.

Here are some example of open access materials and websites:

Materials

Community contributions (2019) *Community contributions for EDAV Fall 2019 (https://jtr13.github.io/cc19/index.html)*

Kabacoff Robb (2020) *Data visualization with R Data* (https://rkabacoff.github.io/datavis/)

French Trevor (2022) *R for data analysis* (https://trevorfrench.github.io/R-for-Data-Analysis/)

Bogart Zach and Robbins Joyce (eds) (2022) *Everything you need for Exploratory Data Analysis & Visualization* (https://jtr13.github.io/EDAV/index.html)

Winston Chang (2023) *R graphics cookbook* (https://r-graphics.org/index.html)

Websites

StackOverflow(https://stackoverflow.com/questions/tagged/r/)

CrossValidated(https://stats.stackexchange.com/)

R-Bloggers (https://www.r-bloggers.com)

RDocumentation(https://www.rdocumentation.org)

Graph Gallery (https://r-graph-gallery.com))

ChatGPT(https://chat.openai.com/)