R basics for Research

Marcelino Guerra

8/16/2022

Table of contents

Preface

This book is a gentle introduction on how to program in R. Hands-on examples show challenges that researchers frequently face and how to overcome common issues associated to data wrangling and visualization.

1 Setting things up

1.1 Installing R

To install R, go to this link. At the top of the web page, you have three links for downloading R, depending on your operating system. If you are using Windows, follow "Download R for Windows" -> "base." To install R on a Mac, click "Download R for Mac." Then, choose the latest release depending on which Mac you have: Intel or Apple silicon.

1.2 Installing RStudio

RStudio is an integrated development environment for R and is highly recommended - it makes using R much more accessible. Download RStudio for free here. Follow the default instructions.

1.3 Rstudio tour

The standard RStudio set-up consists of four panes. On the top left, you have scripts where you write code and save it. On the bottom left, you have the console. The console waits for you to run coding lines, process them, and show the results of what you did (it might also output an error message). The environment shows stored information on the top right and might also report the session's memory usage. Finally, at the bottom right, you have plots and interactive views.

It is possible to customize your RStudio. For instance, you can change the appearance and pane layout. Go to `Tools -> Global Options -> Appearance` to change font size, and editor theme.

1.4 Creating a Script

To create a **script**, you can either click on the top left icon and then R script or press `Shift + Control/Command + N`. Scripts are handy to save your work and organize tasks. For instance,

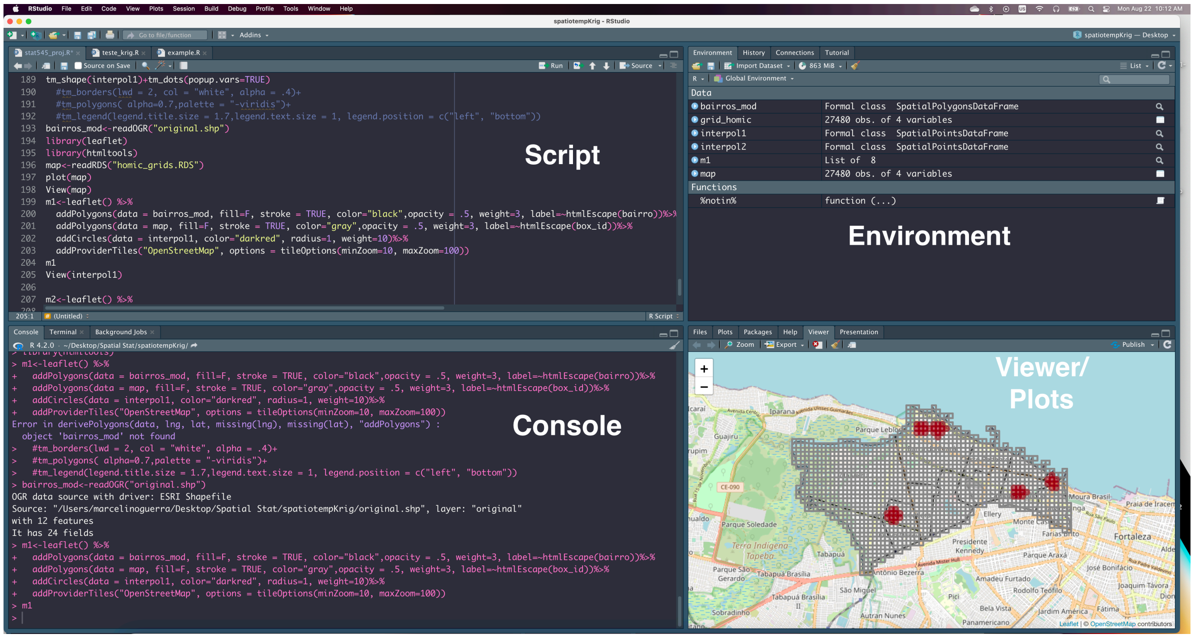


Figure 1.1: RStudio

if you have a project that demands data cleaning, visualization, and model estimation, you might want to create different scripts to deal with various tasks.

1.5 Dealing with Packages

Although the base system contains many built-in tools, you will need to install packages to perform many tasks. For instance, to create beautiful visualizations, you might want to use `ggplot2`, included in the `tidyverse` collection of R packages. To run regressions with many fixed effects, I suggest `fixest`. At this point, you know the drill. In the console, let's try to install `tidyverse` typping `install.packages("tidyverse")` (**yes, between quotation marks**).

Once you have the package installed, there is no need to install it again. However, to use the package, you need to call it first using `library()`:

```
#install.packages("tidyverse")
library(tidyverse)
# tidyverse is a collection of super useful packages like ggplot2, dplyr, readr, etc...
```

Another detail here is the use of `#` in the script. If you use `#` before your coding lines, R won't run them. `#` is also helpful if you want to make comments within your code.

1.6 Installing R Markdown

To install Rmarkdown, **write in the console**:

```
# Install from CRAN
install.packages('rmarkdown')
```

To generate PDF output, you will need to install LaTeX. Your machine might already have MikTeX, but TinyTex is highly recommended. Again in the console:

```
install.packages('tinytex')
tinytex::install_tinytex()
```

1.7 Creating an R Markdown Document

R Markdown documents are fully reproducible and allow the use of multiple languages (R, Python, SQL). If you are teaching a class that demands to code, R Markdown will make grading much easier since the PDF would display the R coding chunk and the output right after.

To create an R Markdown document, click on the top left and `R Markdown… -> Document`. Then, choose the output format. If you want to know more about R Markdown, check Xie, Allaire, and Grolemund (2018).

1.8 Creating a Project

A Project helps you to organize all the files related to a specific task:

- A paper you are writing.
- A replication you are doing.
- Maybe a homework assignment you have.

In that sense, a Project is a folder where you keep all the scripts, data, tables, and results of whatever the task is. Important to mention that keeping everything in one place avoids trouble with the `working directory`.

On the top left (second icon), you have the option to create a new Project. Then, `New Directory -> New Project`. You need to give it a name and locate it (for instance, on the Desktop). Note that you need to open the Project before opening its scripts.

1.8.1 Exercise

Create an R Project and scripts to use during the workshop.

2 Getting Started with R

2.1 Messing around

Math expressions are generally accepted in R. For instance if you type `2+2` the console will output `4`.

```
2+2
```

```
[1] 4
```

Now, try `-`, `*`, `/`, and `^` (for raising to a power). Besides that, there are many built-in math functions - check some of them here.

What you will mostly do is to create objects. For example:

```
odd<-c(1,3,5,7,9,11)
odd
```

```
[1]  1  3  5  7  9 11
```

`odd` is a vector containing some odd numbers. A few details: `c()` concatenates its arguments (odd numbers from 1 to 11) to form a vector named `odd`. Another way to do it is using `seq()` :

```
odd<-seq(from=1, to=11, by=2)
odd
```

```
[1]  1  3  5  7  9 11
```

You can easily apply functions to objects:

```
mean(odd)
```

```
[1] 6
```

```r
sum(odd)
```

```
[1] 36
```

```r
max(odd)
```

```
[1] 11
```

```r
min(odd)
```

```
[1] 1
```

Logical tests are common when dealing with data and now is a good time to get some practice. Test equality with `==` and inequality with `<=`, `<`, `!=`, `>`, or `>=`.

```r
4/2==2 # Is 4 divided by 2 equal to 2?
```

```
[1] TRUE
```

```r
2!=3 # Is 2 different than 3?
```

```
[1] TRUE
```

```r
2>10/5 # Is two greater than 10 divided by 5?
```

```
[1] FALSE
```

It is very common to check whether something belongs to a group, and `%in%` is very helpful in this case:

```r
2 %in% odd # 2 does not belongs to odd
```

```
[1] FALSE
```

Finally, we need to talk about `&` (and) and `|` (or):

```
2 %in% odd | 3 %in% odd # does 2 or 3 belong to odd?
```

```
[1] TRUE
```

```
2 %in% odd & 3 %in% odd # does 2 and 3 belong to odd?
```

```
[1] FALSE
```

Throughout your research, you will constantly work with strings. Any value written within a pair of single or double quotes in R is treated as a string. Below you have stored `Hi` and `Marcelino`.

```
hi<-"Hi"
name<-"Marcelino"
```

The function `paste()` puts things together with any separator:

```
paste(hi, name, sep=" ") # separating strings with space
```

```
[1] "Hi Marcelino"
```

Another useful function is `sample()`. It takes a sample of the specified `size` from the elements of a vector using either `with` or `without replacement`. Before using `sample()`, to make sure we get the same results, lets start the code chunk with `set.seed(123)`.

```
set.seed(123)
numbers<-seq(1:1000)
sample(numbers, size=2, replace = TRUE) # a random sample size 2 of numbers from 1 to 1000
```

```
[1] 415 463
```

```
sample(numbers, size=10, replace=FALSE) # a random sample size 10 of numbers from 1 to 100
```

```
 [1] 179 526 195 938 818 118 299 229 244  14
```

You can also use a sample with strings:

```
fruits<-c("apple", "orange", "lime")
sample(fruits, size=2) ## replace is False by default
```

```
[1] "orange" "apple"
```

2.1.1 Exercises

1. Use `sample()` to simulate a fair coin toss 6 and 1,000 times. Does it look like a fair coin?

Hint: create a vector c("H", "T") and use `sample()` with different sizes. Should you use replace = False or replace = True?

2. Let's play dice! When you roll a fair die, you expect to get 1,2,3,4,5, and 6 with the same probability $\frac{1}{6}$. Hence, the mathematical expectation of that process is:

$$E[X] = \frac{1}{6}[1+2+3+4+5+6] = 3.5$$

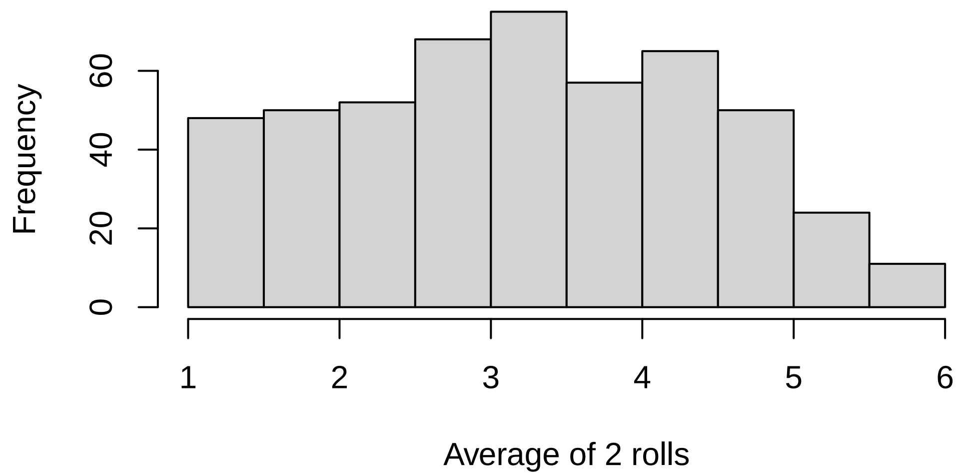
According to the Law of Large Numbers, if you play long enough, the sample average will get close to 3.5. Now, let's look at the histograms for each set of averages according to the sample size (number of rolls). We start rolling a dice two times, and we repeat this process 500 times. The resulting distribution is called "the sampling distribution of the sample mean."

```
roll<-sample(1:6, size=2, replace=TRUE)
mean(roll)
```

```
[1] 2.5
```

Using the function `replicate()` to repeat this process 500 times and plotting the histogram using `hist()`:

```
hist(replicate(500,mean(sample(1:6, size=2, replace=TRUE ))), main=" ", xlab = "Average of
```



Your turn! Roll dice **100 times** and repeat the process again, plotting the histogram. Does this distribution look familiar?

2.2 Data with R

2.2.1 Data Types

The table below summarizes the data types you usually face when working in R:

Table 2.1: Data Types

Type	Definition
Double	A vector containing real values
Integer	A vector containing integer values
Character	A vector containing character values (e.g., "Dog", "1")
Logical	A vector containing logical values (TRUE, FALSE)
Factor	Factors are used to describe items that can have a finite number of values ("male", "female")

Factors look like character vectors, but possess a `levels` attribute that assigns names to each level, or distinct value, in the vector.

Use the `str()` function to identify data types within data structures.

2.2.2 Data Structures

2.2.2.1 Vectors

You can create a vector by combining elements of the same type together using the concatenate function `c()`.

```
vec1<-c(-10:10)
vec1
```

```
 [1] -10  -9  -8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3   4   5   6   7   8
[20]   9  10
```

What happens if you have a string and numbers in the same vector?

```
vec2<-c(-10:10, "cat")
vec2
```

```
 [1] "-10" "-9"  "-8"  "-7"  "-6"  "-5"  "-4"  "-3"  "-2"  "-1"  "0"   "1"  
[13] "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"  "cat"
```

More on data types later! You can use `[ ]` to locate elements within vectors:

```
vec2[22]
```

```
[1] "cat"
```

One additional useful function is `which()` . If you want to find out the position of numbers greater than zero within `vec`:

```
which(vec1>0)
```

```
 [1] 12 13 14 15 16 17 18 19 20 21
```

If you want to find out who are these numbers:

```r
vec1[which(vec1>0)]
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

What happens when you try this using `vec2` instead?

2.2.2.2 Matrices

The matrix is a two-dimensional data structure composed of elements of the *same data type*.

```r
matrix1<-matrix(1:4, ncol=2, nrow=2)
matrix1
```

```
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Let's multiply two matrices:

```r
matrix2<-matrix(6:9, ncol=2, nrow=2)
matrix3<-matrix1 %*% matrix2
```

To find out elements of a matrix you can still use `[ ]` :

```r
matrix3[2,2]
```

```
[1] 52
```

2.2.2.3 Lists

A list is a general form of a vector, where the elements don't need to be of the same type or dimension. You can easily combine arguments:

```r
list1<-list(seq(1:10), c("Cat", "Dog"), matrix(1:6, ncol=3, nrow=2))
list1
```

```
[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10

[[2]]
[1] "Cat" "Dog"

[[3]]
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Given the output you are seeing, how to locate the elements within a list?

2.2.2.4 Dataframes

R usually refers to datasets as dataframes. A data frame is like a list of vectors combined into a matrix-like structure. You can have different columns of different types. Let's create a dataframe:

```r
GDP<-c(10000, 11000, 12431, 500 )
country<-c("Bolivia", "Brazil", "Chile", "Argentina")

df<-data.frame(GDP, country)
df
```

```
    GDP   country
1 10000   Bolivia
2 11000    Brazil
3 12431     Chile
4   500 Argentina
```

2.3 Reading files

In this example, we will use data from the RAND Health Insurance Experiment (HIE), and there are two datasets. Here you have demographic information about the subjects in the study and also health variables (outcomes) both before and after the experiment. The other file (here) has information about health care spending. Finally, here you have a summary of the RAND HIE. To read `.RDS` files, use the `readRDS()` function.

```r
rand_sample<-readRDS("rand_sample.RDS")
rand_spend<-readRDS("rand_spend.RDS")
```

If you want to see the first values on that dataset, you can use the function `head( )` or use `View(rand_sample)` to open the dataframe in a new tab.

```r
#head(rand_spend,5)
#View(rand_spend)
```

Besides the column `plantype`, which identifies the assigned insurance group of each individual, the variables that we are looking for are displayed below:

Table 2.2: Variables Description

Variable	Definition
rand_sample file	
female	Female
blackhisp	Nonwhite
age	Age
educper	Education
income1cpi	Family Income
hosp	Hospitalized last year
ghindx	General Health Index (before)
cholest	Cholesterol (mg/dl) (before)
systol	Systolic blood pressure (mm Hg) (before)
mhi	Mental Health Index (before)
ghindxx	General Health Index (after)
cholestx	Cholesterol (mg/dl) (after)
systolx	Systolic blood pressure (mm Hg) (after)
mhix	Mental Health Index (after)
rand_spend file	
ftf	Face-to-face visits
out_inf	Outpatient expenses
totadm	Hospital admissions
inpdol_inf	Inpatient expenses
tot_inf	Total expenses

In case you have `.csv`, you might want to use `read_csv()`. Also, check the package readxl (here) if you need to load excel files.

3 Data Wrangling with Tidyverse

Tidyverse is a collection of packages that helps you with data management and visualization. Let's keep playing with the Rand HIE data:

```
rand_sample<-readRDS("rand_sample.RDS")
rand_spend<-readRDS("rand_spend.RDS")
```

You can also check the dataset using the function `glimpse()` (tidyverse package). Then you have a good look at all the 319 columns in this dataset:

```
library(tidyverse) ## don't forget to load tidyverse before using its functions
rand_sample%>%glimpse()
```

3.1 select(), arrange(), group_by(), and summarize()

Let's say you want to compare demographic characteristics of the individuals in the RAND HIE across health insurance groups. To do that, you just need the functions `group_by()` and `summarize()` from the `tidyverse` package. Since there are some missing observations (NA), allow the function mean() to ignore those NAs.

Before doing that, let's first select the columns `plantype` (assigned insurance) `female` (1 if female, 0 otherwise) `blackhisp` (1 if black or hispanic, 0 otherwise) `age`, `educper` (education), and `income1cpi` (income).

```
sub_data<-rand_sample%>%select(plantype, female, blackhisp, age, educper, income1cpi)
## check the new dataframe with View(sub_data)
```

Then, using `sub_data`:

```
sub_data%>%
  group_by(plantype)%>%
summarize(
  Female=mean(female, na.rm=T),
  Nonwhite=mean(blackhisp, na.rm=T),              Age=mean(age, na.rm=T),
```

```
    Education=mean(educper, na.rm=T),
    `Family Income`=mean(income1cpi, na.rm=T),
    `Number enrolled`=n())
```

```
# A tibble: 4 x 7
  plantype     Female Nonwhite   Age Education `Family Income` `Number enrolled`
  <fct>         <dbl>    <dbl> <dbl>     <dbl>           <dbl>             <int>
1 Catastrophic  0.560    0.172  32.4      12.1          31603.               759
2 Deductible    0.537    0.153  32.9      11.9          29499.               881
3 Coinsurance   0.535    0.145  33.3      12.0          32573.              1022
4 Free          0.522    0.144  32.8      11.8          30627.              1295
```

The function `arrange()` allows you arrange values within a variable in ascending or descending order. For instance, if you want to arrange the individuals in the `rand_sample` by income:

```
sub_data%>%arrange(income1cpi)
```

```
# A tibble: 3,957 x 6
   plantype     female blackhisp   age educper income1cpi
   <fct>         <dbl>     <dbl> <dbl>   <dbl>      <dbl>
 1 Catastrophic      1         0    30      11          0
 2 Coinsurance       0         0    53       3          0
 3 Coinsurance       1         0    19      12          0
 4 Free              0         0    36      12          0
 5 Catastrophic      1         0    42      16          0
 6 Coinsurance       1         1    25      11          0
 7 Coinsurance       0         1    35      12          0
 8 Coinsurance       1         0    22      12          0
 9 Free              1         0    34      12          0
10 Free              1         0    50      11          0
# ... with 3,947 more rows
```

If you want to arrange by descending order:

```
sub_data%>%arrange(desc(income1cpi))
```

```
# A tibble: 3,957 x 6
   plantype     female blackhisp   age educper income1cpi
   <fct>         <dbl>     <dbl> <dbl>   <dbl>      <dbl>
```

```
 1 Catastrophic      0          NA     21        15      89132.
 2 Catastrophic      1           0     20        13      89132.
 3 Catastrophic      0          NA     19        13      89132.
 4 Catastrophic      0           0     51        16      89132.
 5 Free              1           0     36        17      79757.
 6 Free              0          NA     14        NA      79757.
 7 Free              0           0     42        17      79757.
 8 Catastrophic      1           0     31        17      78775.
 9 Coinsurance       0           0     40        16      77230.
10 Coinsurance       1           0     40        14      77230.
# ... with 3,947 more rows
```

It is also easy to check the top 5 and bottom 10:

```
sub_data%>%select(income1cpi)%>%top_n(2)
```

```
Selecting by income1cpi
```

```
# A tibble: 4 x 1
  income1cpi
       <dbl>
1     89132.
2     89132.
3     89132.
4     89132.
```

```
sub_data%>%select(income1cpi)%>%top_n(-10)
```

```
Selecting by income1cpi
```

```
# A tibble: 33 x 1
   income1cpi
        <dbl>
 1          0
 2          0
 3          0
 4          0
 5          0
 6          0
```

```
 7          0
 8          0
 9          0
10          0
# ... with 23 more rows
```

3.2 filter(), mutate(), and ifelse()

Let's say you only want to work with two types of participants: the ones with either "Free" or "Catastrophic" insurance. To do that, apply the `filter()` function:

```
cat_vs_free<-rand_sample%>%filter(plantype=="Catastrophic"|plantype=="Free")
```

On top of that, you might want to identify the categories with numbers: a dummy variable that takes on one if `plantype=="Free"` and zero otherwise (you can also think about the dummies we already have for female, nonwhite, etc.). Sometimes, it is just easier to work with numbers than strings. For that task we could use `ifelse()` together with `mutate()`.

```
cat_vs_free<-cat_vs_free%>%
  mutate(
    dummy_plan=ifelse(plantype=="Free",1,0)
          )
```

A few details. `ifelse()` is base R, and a tidyverse alternative is `case_when()` - we will talk about it later.

3.3 Exercise I: Racial Discrimination in the Labor Market

We will use a dataset here from a randomized experiment conducted by Marianne Bertrand and Sendhil Mullainathan for this question. The researchers sent 4,870 fictitious resumes out to employers in response to job adverts in Boston and Chicago in 2001. They varied only the names of job applicants while leaving other relevant candidates' attributes unchanged (i.e., candidates had similar qualifications). Some applicants had distinctly white-sounding names such as Greg Baker and Emily Walsh, whereas other resumes contained stereotypically black-sounding names such as Lakisha Washington or Jamal Jones. Hence, any difference in callback rates can solely be attributed to name manipulation.

1. Create a dummy variable named `female` that takes one if `sex=="f"`, and zero otherwise.

2. The dataset contains information about candidates' education (`education`), years of experience (`yearsexp`), military experience (`military`), computer and special skills (`computerskills` and `specialskills`), a dummy for gender (`female`), among others. Summarize that information by getting average values by `race` groups.

3.4 Exercise II: The Tennessee STAR experiment

"Education production" is an area much explored by economists. The terminology reflects that we think of features of the school environment as inputs that cost money, while student learning is the output that schools produce. A major question in the field is which inputs have the highest benefit/cost ratio, and one very costly input is class size. An important experiment conducted in Tennessee was designed to precisely answer the question "Does class size impacts student performance?".

Krueger (1999) analyzed the Project STAR, a longitudinal study that randomly assigned kindergarten students and their teachers to one of three groups beginning in the 1985–1986 school year. The three groups were small classes (13–17 students per teacher), regular-size classes (22–25 students), and regular/aide classes (22–25 students) which also included a full-time teacher's aide. After their initial assignment, the design called for students to remain in the same class type for four years. Some 6000–7000 students were involved in the project each year. You can find part of the sample related to students who entered STAR in kindergarten here to answer the following questions.

1. Create the dummy variables `Free_lunch` (takes 1 if `lunch` is "free"), `White_asian` (equal 1 if ethnicity is either "cauc" or "asian") and `Female` - takes 1 if gender is "female". Also, define the variable `age` as 1986-`birth`, i.e., compute the age of the children in 1986.

2. The first question to ask about a randomized experiment is whether the randomization successfully balanced the subject's characteristics across different groups. Although the STAR data failed to include any pretreatment test scores, we can look at some characteristics of students such as race, gender, age, and free lunch status, which is a good measure of family income since only poor children qualify for free school lunch. Compare the values of `Free_lunch`, `White_asian`, `Female`, and `age` across the three groups `small`, `regular`, `regular+aide`. Do these variables look balanced?

3.5 Merging datasets

Sometimes you get information from different sources but want to analyze all together. When that happens, one can use `merge()` or the many "joins" available on `tidyverse()`. Say you want to put together two datasets: one with socioeconomic information about counties (here) and another that has health indicators also at the county level (here) The first step is to load both of them:

```
library(tidyverse)
HHincome<-read_csv("HHincome18.csv", col_names = TRUE) ## col_names is TRUE because all th
health_data<-readRDS("health_data.RDS")
```

A few details here. Since the `.csv` file has column names, we have `col_names=T`. Also, the function `read_csv()` belongs to `tidyverse` and you need to call the package first.

To put the two datasets (income and health) together, you can use `join` function. **There are some details here**. You have 3,275 counties (rows) in `HHincome`, but only 3,141 counties in `health_data`. Here I will use `left_join` restricting the merge to those 3,141 counties inside `health_data`. Check the other join types here. Using the function `dim()` you will realize that your new data has 3,141 rows and 9 columns.

```
full_data<-left_join(health_data, HHincome,by = "FIPStxt")
dim(full_data)
```

```
[1] 3141    9
```

3.6 drop_na() and replace_na()

You will frequently face datasets that have columns with missing information. Sometimes, you need to replace NAs by zero, sometimes you drop the entire row and ignore that unit of observation. Let's see how that works. First, create a new "full_data" but this time using the `right_join()` . Can you explain the difference between these joins?

```
full_data2<-right_join(health_data, HHincome,by = "FIPStxt")
#View(full_data2)
dim(full_data2)
```

```
[1] 3275    9
```

Now you have 3,275 counties, but some of them do not have health indicators and you might want to drop these places:

```
full_data2<-full_data2%>%drop_na()
dim(full_data2)
```

```
[1] 3134    9
```

To replace `NAs` by zero (or something else) instead, check replace_na().

3.7 Exercise III: `%in%` and `%notin%` more summarize()

1. Instead of using `%in%`, you might want to try `%notin%`. Base R does not provide a `%notin%` function, so we will need to create it:

```
`%notin%`<-Negate(`%in%`)
```

And here is the question: which counties in `full_data` are not in `full_data2`?

2. Using `full_data2`, get the top 5 and bottom 10 **counties** in terms of **% of people eligible for Medicare**. Check also the top 5 and bottom 10 **states** concerning the **total ICU beds**.

Hint: The column `EligibleforMedicare18` refers to the number of people eligible for Medicare per county in 2018. Note that you also have the estimated 2018 population (`PopulationEstimate2018` column).

3.8 group_by() and ungroup()

We already saw how `group_by` works. But what if you want to preserve your dataset but add a new column with some group level information? For instance, what if you want to add a column with state-level information about total ICU beds?

```
full_data2<-full_data2%>%
  group_by(StateName)%>%
  mutate(
    StateICU=sum(ICU_beds)
    )%>%
  ungroup()
```

3.9 Exercise IV: Airbnb and Chicago Communities

The first dataset Airbnb.RDS refers to Airbnb rentals, socioeconomic indicators, and crime by community area in Chicago. The Communities.xls file contains health and socioeconomic indicators for the 77 community areas of Chicago, 2012-2014.

1. Import both datasets and use `View()` to check them.
2. Merge the datasets using the function `full_join()`. What is the dimension of your new dataset? What variables do they have in common (variables with the same column name)?

The `Airbnb` data has the columns `area` and `dist`. They represent the total community area and the distance (in km) from the community to Chicago downtown, respectively.

3. You want to work only with the following columns: `community`, `price_pp`, `num_spots`, `rev_rating`, `PerCInc14` , `num_theft`, `FirearmM` ,`unemployed`, `harship_in`, `Pop2014`, `BirthRate`, `Over65`, `dist`, and `area`. Select only those variables and store them in a new data frame.
4. Create the new variable `theft_rate` dividing the total number of thefts by the **population in 2014**.
5. First, divide the total population in 2014 by the community's area to get values for population density (number of people per square mile). Then, create the new variable `logdens`, taking the natural logarithm of population density.
6. Filter your new dataset to identify **Central Chicago**. In other words, you want to filter communities within 3km from Chicago downtown. What is the average number of Airbnb spots in Central Chicago? What are the average Airbnb prices, per capita income, theft rate, firearm-related deaths, population density, and birth rate in Central Chicago?
7. Finally, compare the values for the same variables in Central Chicago with the average numbers from **Far from downtown** - the communities that have a distance from downtown higher than 19 km.

4 Data Visualization with Tidyverse

4.1 Line Plots

Line plots give you a good understanding about how variables evolve over time. Let's see how simple line plots work using `ggplot2()` and some options available to customize these graphs. In this exercise, we will use with urbanization and per capita income information about a bunch of countries from Our World in Data. The first step is to import this .csv file:

```
library(tidyverse)
urban_gdp<-read_csv("urbanization_gdp.csv", col_names=T)
```

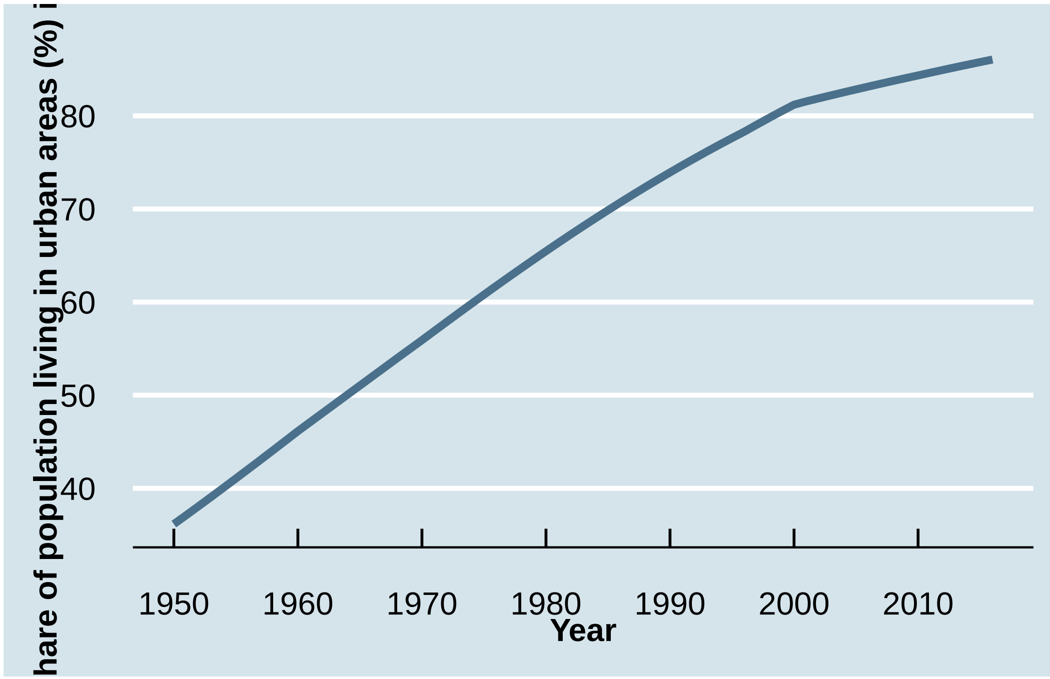
One can check the evolution of the share of urban population (%) in Brazil, from 1945 to 2016 filtering the full dataset

```
brazil<-urban_gdp%>%
  filter(Entity=="Brazil")%>%
  arrange(Year)%>%
  filter(Year>=1950)
```

Working with the filtered data, use `ggplot` package to create line plots. The package `ggthemes` allows you to customize the chart appearance. For this exercise, I will use `theme_economist`. The basic line plot works with geom_line(), and you can easily set the line `type` and its `size`. `theme()` adjust the size of the label in the axis, and `scale_x_continuous()` gives you some freedom to establish the years that appear in the x axis.

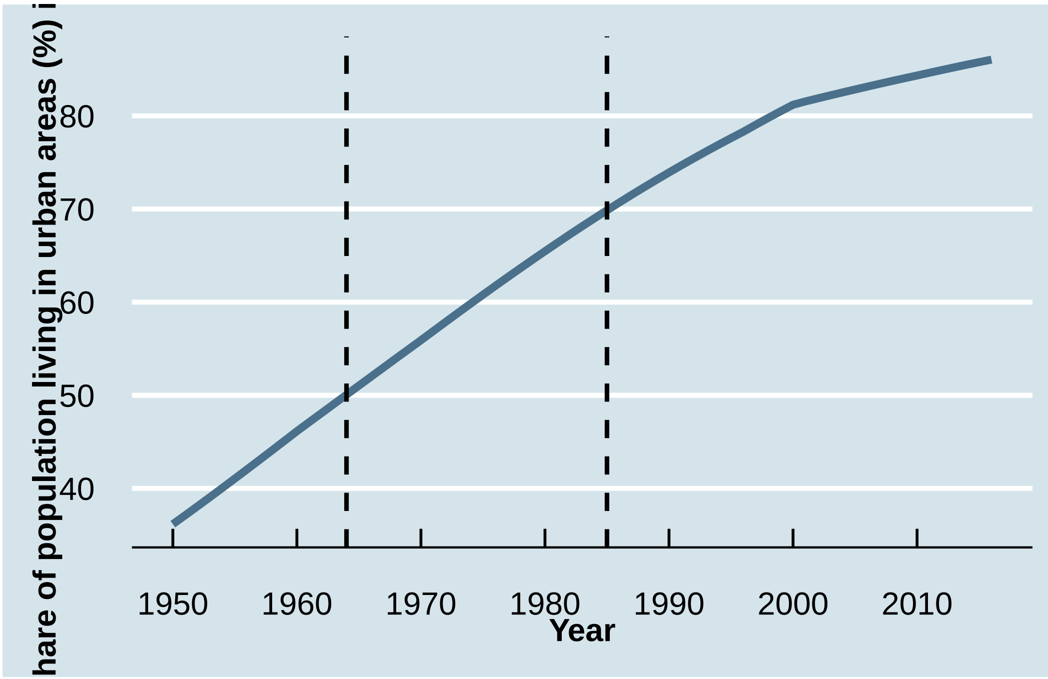
```
library(ggthemes)

ggplot(data=brazil,
       aes(x=Year,y=`Share of population living in urban areas (%)`))+
  geom_line(size=1.4, color="skyblue4")+
  labs(x = "Year", y="Share of population living in urban areas (%) in Brazil")+
  theme_economist(base_size = 14) +
  scale_colour_economist()+
  scale_x_continuous(breaks = seq(from = 1950, to =2016 , by =10))+
  theme(axis.text=element_text(size=12),
```

```
        axis.title=element_text(size=12,face="bold"))
```



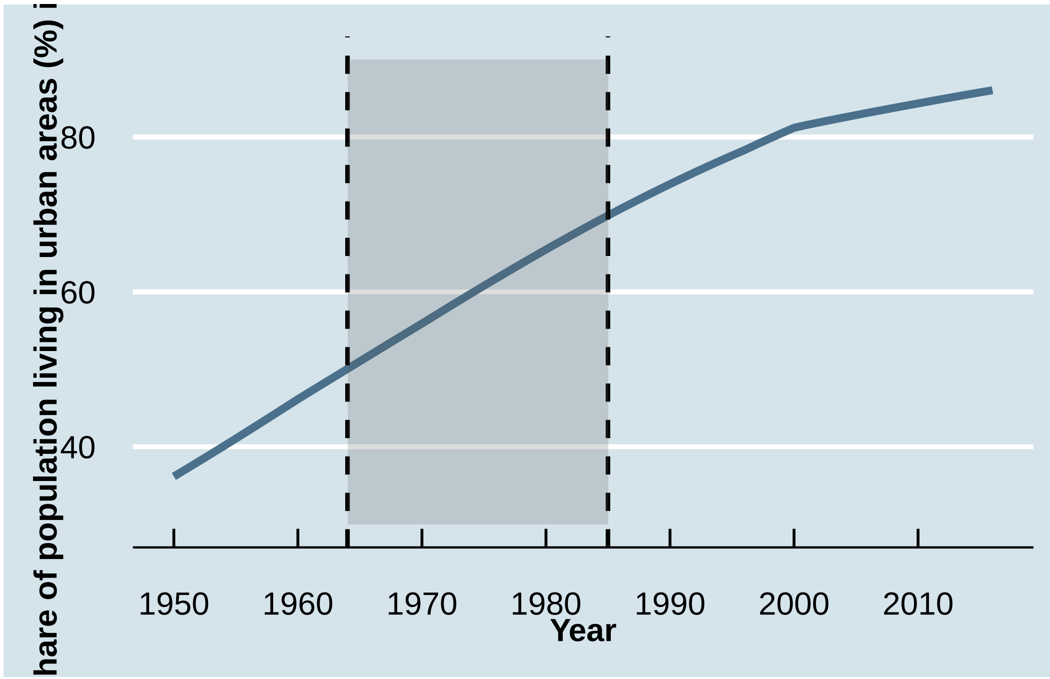
Sometimes you need to highlight a certain date or period, and vertical lines come in handy. For instance, Brazil was under Military dictatorship from 1964 to 1985. Using `geom_vline()`, it is possible to add vertical lines:

```
ggplot(data=brazil,
       aes(x=Year,y=`Share of population living in urban areas (%)`))+
  geom_line(size=1.4, color="skyblue4")+
  labs(x = "Year", y="Share of population living in urban areas (%) in Brazil")+
  theme_economist(base_size = 14) +
  scale_colour_economist()+
  scale_x_continuous(breaks = seq(from = 1950, to =2016 , by =10))+
  theme(axis.text=element_text(size=12),
        axis.title=element_text(size=12,face="bold"))+
  geom_vline(aes(xintercept =1964),
             linetype=2, colour="black", size=.8)+
  geom_vline(aes(xintercept =1985),
             linetype=2, colour="black", size=.8)
```



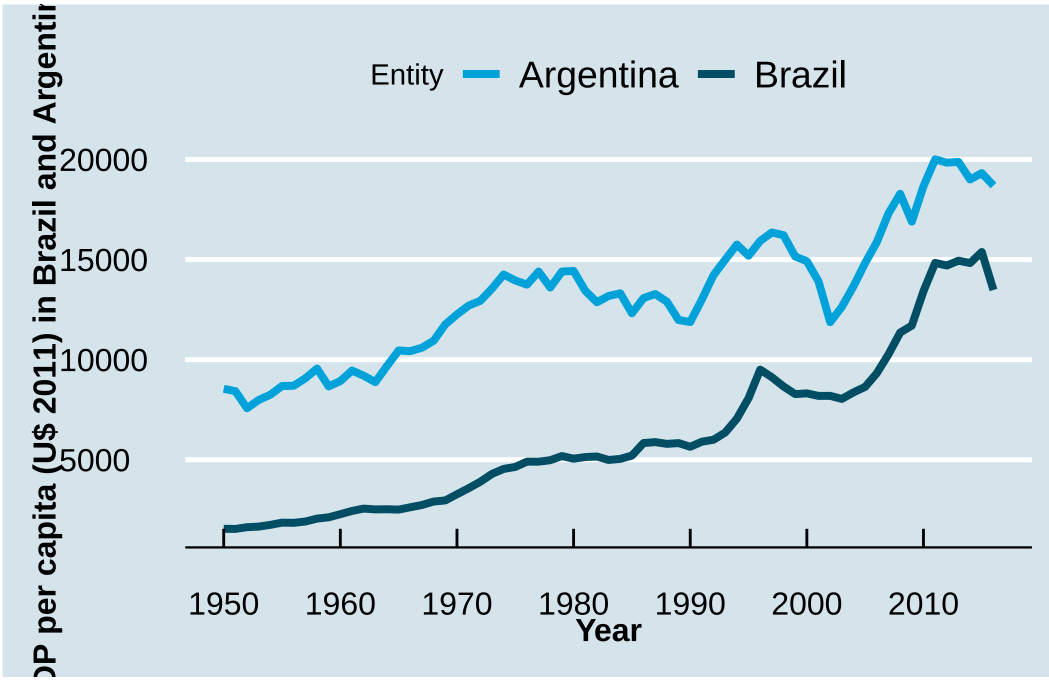
It is also possible to add a colored box between periods with `annotate()`. You will need to define the size of the rectangle with `xmin`, `xmax`, `ymin`, `ymax`, and the transparency choosing `alpha`.

```
ggplot(data=brazil,
       aes(x=Year,y=`Share of population living in urban areas (%)`))+
  geom_line(size=1.4, color="skyblue4")+
  labs(x = "Year", y="Share of population living in urban areas (%) in Brazil")+
  theme_economist(base_size = 14) +
  scale_colour_economist()+
  scale_x_continuous(breaks = seq(from = 1950, to =2016 , by =10))+
  theme(axis.text=element_text(size=12),
        axis.title=element_text(size=12,face="bold"))+
  geom_vline(aes(xintercept =1964),
            linetype=2, colour="black", size=.8)+
  geom_vline(aes(xintercept =1985),
            linetype=2, colour="black", size=.8)+
  annotate("rect", xmin = 1964, xmax = 1985,
           ymin = 30, ymax = 90,
           alpha = .2)
```



There are some options in case you want to display multiple units in the same plot. Here we will leverage `col` inside `aes()` to differentiate Brazil and Argentina. Note that a legend was created on the top.

```
br_ar<-urban_gdp%>%
  filter(Entity=="Brazil"|Entity=="Argentina")%>%
  arrange(Year)%>%
  filter(Year>=1950)

ggplot()+
  geom_line(data=br_ar,
            aes(x=Year, y=`GDP per capita (2011 int-$) ($)`, col=Entity),
            size=1.4)+
  labs(x = "Year", y="GDP per capita (U$ 2011) in Brazil and Argentina")+
  theme_economist(base_size = 14) +
  scale_colour_economist()+
  scale_x_continuous(breaks = seq(from = 1950, to =2016 , by =10))+
  theme(axis.text=element_text(size=12),
        axis.title=element_text(size=12,face="bold"))
```



Depending on how the data is structure, you might need to use multiple `geom_lines()` or reshape the dataset (check these examples).

4.1.1 Exercise I: Germany's GDP per capita evolution

Plot Germany's GDP per capita evolution and add a vertical dashed line at 1989 choosing a different theme - check the options here.

4.1.2 Exercise II: Urbanization in US and China

Plot the urbanization rates from both the US (in darkblue) and China (in red), from 1930 to 2016.

Hint: It is possible to customize the colors adding the line `scale_color_manual("", values=c("red", "darkblue"))`

4.2 Scatterplots

Using scatter plots, you can see the extent to which two variables are associated, and it is a very compelling way to show simple correlations. For example, one can argue that per capita income goes along with urbanization - as countries urbanize, they get richer. To construct a scatter plot between those two variables for 2016, we filter the entire dataset (urban_gdp) for `Year==2016`.There are entities without information about GDP per capita, and **we drop those NA's**. If you look at the data, you will realize that when an Entity has Code equal to NA, the entity is not actually a single country, and we also want to drop those data points. Finally, we dropped the entity World because we only want to work with countries.

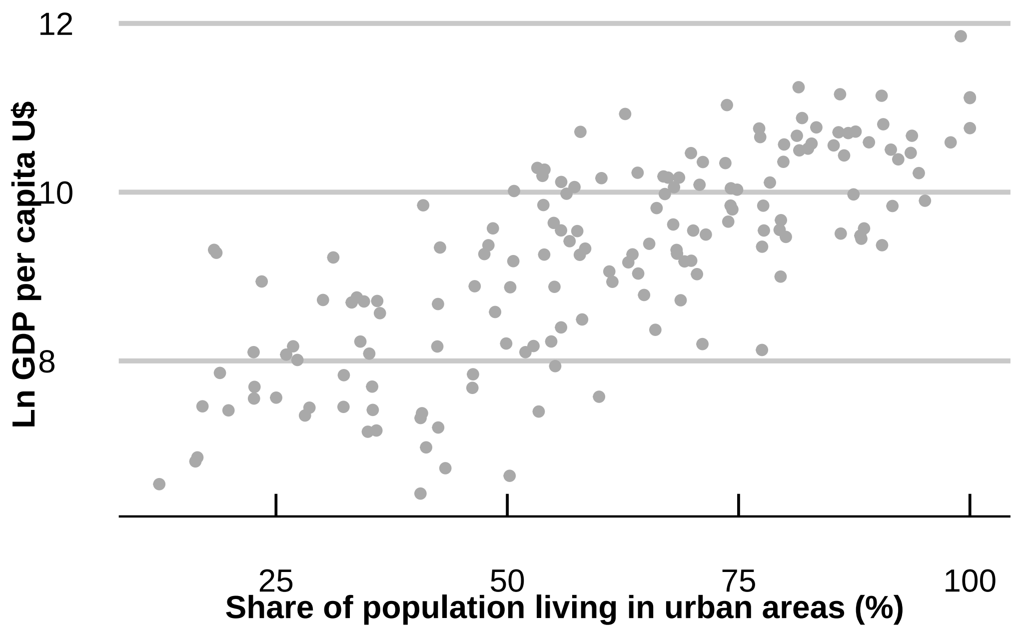
As you can see, cleaning the data is a crucial step before plotting. Finally, I will rename columns 4 and 5 and create a new variable applying the natural logarithm to the GDP per capita.

```
urban_gdp_16<-urban_gdp%>%filter(Year==2016)
urban_gdp_16<-urban_gdp_16%>%drop_na(Code, `GDP per capita (2011 int-$) ($)`)
urban_gdp_16<-urban_gdp_16%>%filter(Entity!="World")
names(urban_gdp_16)[4]<-"urban_pop"
names(urban_gdp_16)[5]<-"gdp"
urban_gdp_16$ln_gdp<-log(urban_gdp_16$gdp)
head(urban_gdp_16,3)
```

```
# A tibble: 3 x 7
  Entity      Code   Year urban_pop   gdp `Total population (Gapminder)` ln_gdp
  <chr>       <chr> <dbl>     <dbl> <dbl>                          <dbl>  <dbl>
1 Afghanistan AFG    2016      25.0  1929                             NA   7.56
2 Albania     ALB    2016      58.4 11285                             NA   9.33
3 Algeria     DZA    2016      71.5 13328                             NA   9.50
```

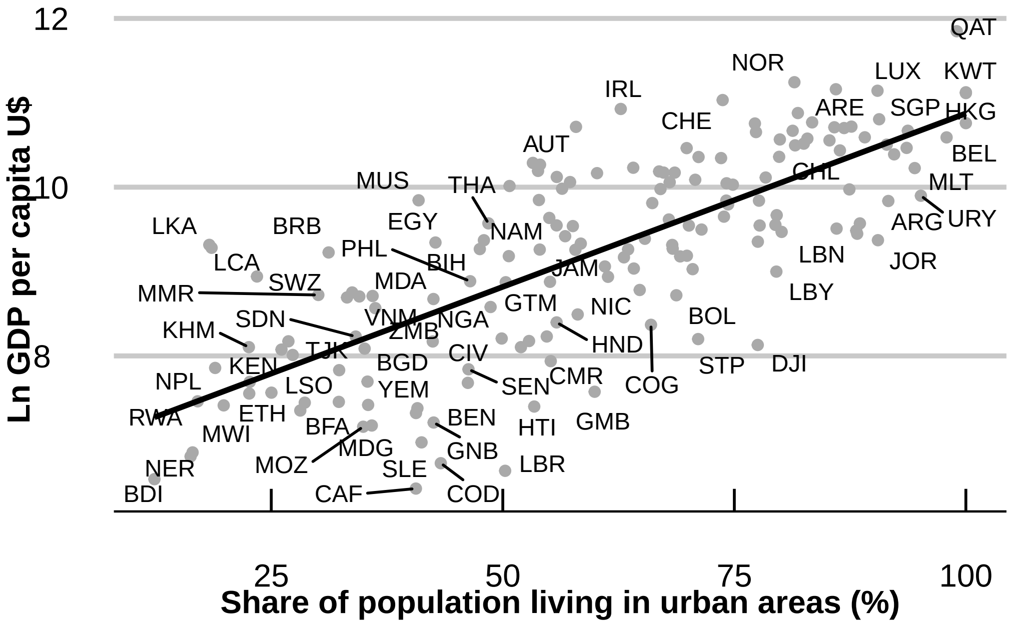
Back to `ggplot()`! Instead of `geom_line()`, we use `geom_point()` to construct scatter plots:

```
ggplot(urban_gdp_16,
       aes(x=urban_pop, y=ln_gdp)) +
  geom_point(color="darkgrey") +
  scale_x_continuous(name = "Share of population living in urban areas (%)") +
  scale_y_continuous(name = "Ln GDP per capita U$") +
  theme_economist_white(base_size = 17, gray_bg=FALSE)+
  theme(axis.text=element_text(size=12),
        axis.title=element_text(size=12,face="bold"))
```



You can do better than that. The package ggrepel allows you to label the dots in your chart (function `geom_text_repel()`). I will label the dots with respective country codes. Finally, `stat_smooth()` adds the regression line to your plot:

```
#install.packages("ggrepel")
library(ggrepel)
ggplot(urban_gdp_16,
       aes(x=urban_pop, y=ln_gdp)) +
  geom_point(color="darkgrey") +
  geom_text_repel(aes(label=Code), size=3)+
  stat_smooth(method = "lm", formula =y~x, se=F, color="black") +
  scale_x_continuous(name = "Share of population living in urban areas (%)") +
  scale_y_continuous(name = "Ln GDP per capita U$") +
  theme_economist_white(base_size = 17, gray_bg=FALSE)+
  theme(axis.text=element_text(size=12),
        axis.title=element_text(size=12,face="bold"))
```



Here we see the strong and positive relationship between the share of the population living in urban areas (%) and the GDP per capita (in U$) for 164 selected countries. For more facts about urbanization, check Our World in Data.

4.2.1 Exercise III:

Here you have ACS 2015-19 data about the percentage of people living below the poverty line in census tracts within the **Chicago metro area**. Let's look at the relationship between poverty levels and distance from the city center, constructing a scatter plot with those variables (y-axis `perc_pov` and x-axis `dist_km`). Do the same but use `income` in the y-axis instead and keep `dist_km` on the x-axis. Store both scatter plots (as plot1 and plot2). Install the package `patchwork` and sum both objects (literally, plot1+plot2) to display them side-by-side.

References

Xie, Yihui, Joseph J Allaire, and Garrett Grolemund. 2018. *R Markdown: The Definitive Guide*. Chapman; Hall/CRC.