Leibniz Institute for Financial Research SAFE
Theodor-W.-Adorno-Platz 3 I 60629 Frankfurt am Main I Germany

LEIBNIZ INSTITUTE
SAFE
Sustainable Architecture for Finance in Europe

An introduction to the tidyverse in R

Ina Krapp
SAFE Research Datacenter[*]

Last Update: April 16, 2024

R is a programming language often used for Data Analysis. In this tutorial, we will walk through the process of preparing a dataset for analysis and running a regression. We will use the tidyverse, a collection of packages that make working with data in R easier.
The dataset used in this tutorial is the Statistical Review of World Energy data 2023. Download it in xlsx format from here.
In this tutorial, we will also use RStudio. RStudio is a user interface which makes working with R much more convenient. This tutorial is intended to be run as a RMarkdown file in RStudio, which allows you to execute the code yourself. To run it, download the RMarkdown version here and open it in a recent version of RStudio.

Contents

[*]krapp@safe-frankfurt.de

1 Contact

If you encounter any difficulties or just want general information, do not hesitate to contact us.

SAFE Research Datacenter: datacenter@safe-frankfurt.de

More information about the SAFE Research Datacenter and further guides can be found here.

2 Prerequisite

R can be downloaded here.
We'll also use RStudio. You can get it here.

3 Starting R

This workshop aims to give an introduction into the tidyverse, following the typical steps of a research project. Assume I have a research question I want to answer. I also have data. What next?
Once you can open RStudio, you can start right away by writing code on your console (on the left side of the monitor). Or you open a document like this one and run your code in code blocks. Even if you never programmed before, some parts will look familiar to you. Run the code below, or enter it in the console.

```
    1 + 1
```

```
> 1 + 1
[1] 2
```

R can be used as a calculator, but it offers many more functions. It offers more functions than an ordinary statistic program. R is used across disciplines, from astronomy to zoology.
Many of its functions are useful for situations nearly every researcher can find themselves in. The tidyverse is good for working with data. So today, we look at how we can find the answer to a question by working with a dataset.
But first, we have to install the tidyverse. Remove the '#' and run the code below. Put the '#' back again afterwards. You don't want to install the tidyverse more than once, because R would have to be restarted if you install it again.

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

The tidyverse is a package. You can imagine it like an extension or add-on, a small program you can download and use for some things. You don't need to use it every time you use R, but if you use it, you have to activate it first. The package is stored in a local library. To activate it, run this code.

```
library(tidyverse)
```

```
Warning: package 'tidyverse' was built under R version 4.3.
i Use the conflicted package to force all conflicts to become errors
```

You will need to run this code every time you open R again, so you don't want to put a '#' in front of it.
R has many packages, and you need to load different ones depending on what you need to do. For example, we need the readxl package to work with Excel files.

```
library(readxl)
```

4 Getting the data into R

Once R and the tidyverse are set up, it is time to start with the data.
The first step to work with the data in R is to import the data. There are various packages which allow R to work with data of nearly all formats. Be it Excel files, csv files or entire databases, R can handle practically anything.
Let's get data from an Excel file. Excel files can be complex to navigate, so the simplest way is to click on 'files' in the lower right pane and then click on the Excel file you want to import. Then click on 'import dataset'. A Viewer opens. You can now look at the different sheets that the Excel file contains. Click on the sheet 'Oil - Proved reserves history'. You'll see the code to load this sheet will appear in the code preview. Set the number of lines to skip to 2. Then run the code. Your code should now look as follows:

```
    library(readxl)
    Statistical_Review_of_World_Energy_Data <- read_excel("Statistical
   Review of World Energy Data.xlsx",
    sheet = "Oil - Proved reserves history",
    skip = 2)
View(Statistical_Review_of_World_Energy_Data)
```

If you run it, the data gets loaded into R. The 'View' command opens the table directly in R. The tidyverse also allows to export data. csv is a widely used format to store data. To export the data, we can write a new csv with the tidyverse. We have to enter two arguments to 'write_csv', the name of the data we want to export, and the name of the csv file we want to create.

```
    write_csv(Statistical_Review_of_World_Energy_Data, "World_Energy_csv.csv
   ")
```

Likewise, data can be imported into R with 'read_csv':

```
    oil_reserves = read_csv("World_Energy_csv.csv")
```

```
Rows: 99 Columns: 45-- Column specification
Delimiter: ","
chr (45): Thousand million barrels, 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987...
i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Clicking on the csv file and then on 'Import dataset' would have created the same code.

5 Data Cleaning

We have the dataset twice in the environment now. We can delete objects from the environment using the 'remove' command.

```
    remove(Statistical_Review_of_World_Energy_Data)
```

You will want to be careful not to delete data you still need.
Next, let's take a closer look at the data. You probably already noticed that the dataset is not looking very pretty. The last 4 columns look different from the others. we don't need the last 3, so we'll remove them.
We can use the 'select' command for this. 'select' allows to list columns by position or by name. A minus is used to select all columns except the ones listened. The command below selects all variables except the ones at position 43, 44 and 45.

```
    oil_reserves = oil_reserves %>% select(-43, -44, -45)
```

Column 42 contains data from 2020, but is named '2020...42'. We will rename it to '2020':

```
    oil_reserves = oil_reserves %>% rename( `2020`= `2020...42`)
```

These commands follow the typical structure of a tidyverse command: First, you give the name of the object where the result should be stored. Then, you give the name of the dataframe you want to use the command on. Since we want to modify the existing dataframe, we use the same name on both sides of the equality sign. Next follows the pipe, the '% %' sign. It can be read as 'then', so the command above means 'take the dataframe 'oil_reserves', then rename its column '2020...42' to '2020'.
We will also rename the first column. It is currently named 'Thousand million barrels', but it contains the Country names.

```
    oil_reserves = oil_reserves %>% rename(Country= `Thousand million
   barrels`)
```

You will want to pick meaningful names, which don't start with numbers and don't contain empty spaces. R can work with them, but as seen above, you need to use quotation signs.
Another remark: If you run the line of code above again, you will get an error message. That is because the column was already renamed, so if the program searches a column with the old name, it won't find any. To rerun it, you'll have to rerun all of the code above.
The dataset also contains many grey 'NA' variables. NA means 'not available'. R interprets them as missing data. But in our case, they are not missing values. The Excel file simply contained some empty lines, which R interprets as missing. We can simply drop them:

```
    oil_reserves = oil_reserves %>% drop_na()
```

You will see in the Environment pane that running this code shrinks the dataset: It used to contain 99 obs., now it are only 72. R always removes entire rows if some of their values contain NA's that get removed.

6 Pivot longer and mutate

R always interprets a table as consisting of variables and observations, and it always assumes that each row contains an observation and each column contains a variable. For this reason, our dataset, with 72 rows and 42 variables, is listed as having 72 observations of 42 variables. If a variable is distributed across several rows, like it is the case for the oil reserves in our dataset, the dataset needs to be turned into a longer format. Such a reformatting of the data is called 'pivoting' in R.

```
    oil_reserves = oil_reserves %>% pivot_longer(cols = !Country, names_to =
    "Year", values_to = "Thousand_million_barrels")
```

As you can see, the data looks differently if you open it now. It only has 3 columns, but nearly 3000 observations. Each observation contains the oil reserves that a certain country had at a certain time. You can also use 'pivot_wider' to turn a dataset into the wide format. The code below creates a new dataframe in the wider format, 'oil_reserves_wider', which you can compare to the longer format.

```
    oil_reserves_wider = oil_reserves %>% pivot_wider(names_from = "Year",
   values_from = Thousand_million_barrels)
```

But working with the longer format is standard in R, so we'll continue with the longer format.
The observations in our dataset contain real missing data. To deal with missing data can be difficult because there are many possible reasons why it may be missing.
In our dataset, some missing datapoints occur because not all countries existed in all years. The dataset starts in 1980, and data for the USSR is only available until 1990. The data for

Russia is starting in 1991. Missing data is denoted in the dataset as ”n/a”. This can change from data to data, so we need to define in R that ’n/a’ is missing data in our case.

```
oil_reserves = oil_reserves %>% mutate(Thousand_million_barrels = na_if(
  Thousand_million_barrels, "n/a"))
```

This ’mutate’ command can be used for many different tasks. We use it here to convert all values that are ”n/a” to the NA format that R registers as missing.
We can also use it to change the type of a column. R usually imports data with type ’character’. ’Character’ can be any data, text or a number, but you can’t perform calculations with a column that has this type.
To perform calculations with it, we need to turn the values into numeric values.

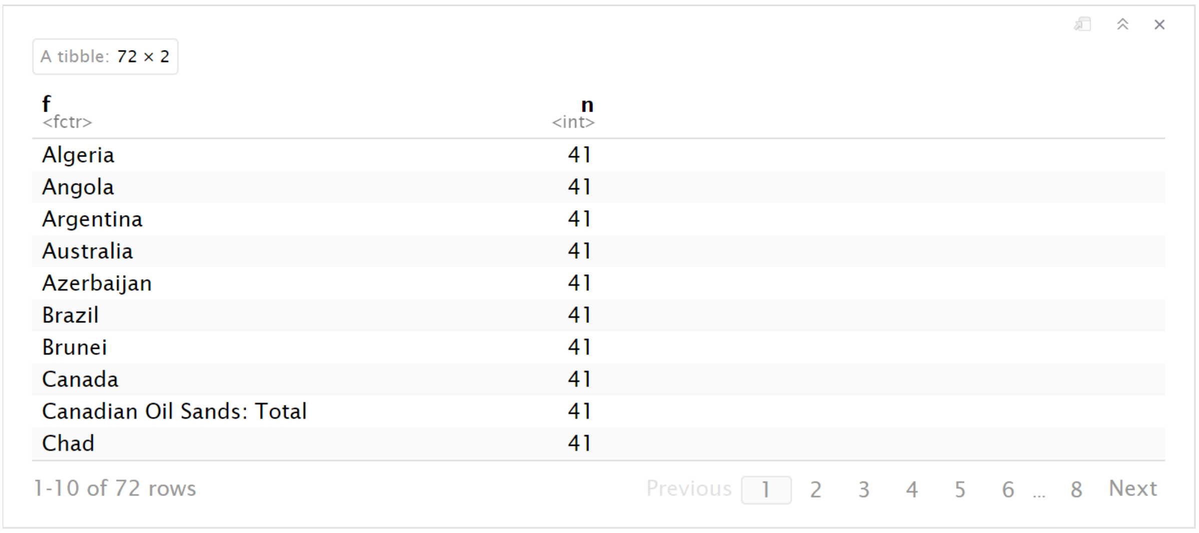
```
oil_reserves = oil_reserves %>% mutate(Thousand_million_barrels = as.
  numeric(Thousand_million_barrels))
```

We can also turn a column into a variable of the type ’factor’. Factors are categories a value may take.

```
oil_reserves = oil_reserves %>% mutate(Country = as.factor(Country))
```

This type is convenient to get an overview over which countries we have in our data. We can look at the factors included in the column ’Country’ with this code:

```
fct_count(oil_reserves$Country)
```


A tibble: 72 × 2

f <fctr>	n <int>
Algeria	41
Angola	41
Argentina	41
Australia	41
Azerbaijan	41
Brazil	41
Brunei	41
Canada	41
Canadian Oil Sands: Total	41
Chad	41

1-10 of 72 rows

We see each country appears 41 times because the data covers 41 years. There are 72 different names in the column ’Country’.

7 Filtering values and working with strings

As you can see above, the data includes some entities which are not nations. That could become a problem because it means some values are implicitly counted twice. For example, the oil reserves of countries like Iran are included in the oil reserves of the Total Middle East.
We can filter such values out using the ’filter’ command. It works much like the ’select’ command we saw previously. But ’select’ selects columns. ’filter’ looks for specific values in rows. We can look at the data in the wide format to see which values we should exclude because they are not Countries.

To remove them, we can use 'stringr' methods. They are used to work with character strings, in our case, country names. The method 'str_detect' detects if a certain combination of letters is present in a certain string. Using the '!' sign, R filters for all rows except those in which the given letters were detected. So the command below keeps all rows whose country does not include the letters 'Total'.

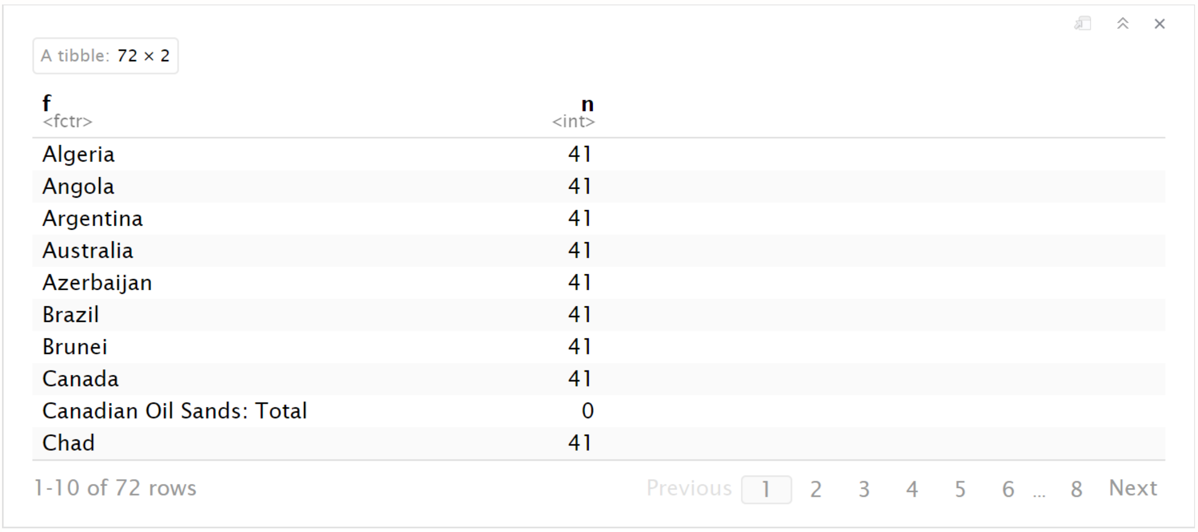
```
    oil_reserves = oil_reserves %>% filter(!str_detect(Country, "Total"))
```

In the same manner, we can exclude international organizations like the OPEC. We can also exclude the Orinoco Belt, whose oil reserves belong to Venezuela, as well as the Canadian oil sands currently under active development, which are a part of the Canadian reserves.

```
    oil_reserves = oil_reserves %>% filter(!str_detect(Country, "OPEC"))
    oil_reserves = oil_reserves %>% filter(!str_detect(Country, "OECD"))
    oil_reserves = oil_reserves %>% filter(!str_detect(Country, "European
  Union"))
    oil_reserves = oil_reserves %>% filter(!str_detect(Country, "Orinoco
  Belt"))
    oil_reserves = oil_reserves %>% filter(!str_detect(Country, "Under
  active development"))
```

stringr commands like 'str_detect' can be used with regular expressions. They allow to, for example, filter by first or last letter. They can become very complex, so I can not cover them here, but if you want to work with text data, they are worth looking into.
If we look at how often a country name appears, we can see there are still 72 different ones, but the ones we excluded appear 0 times now.

```
    fct_count(oil_reserves$Country)
```



A tibble: 72 × 2

f <fctr>	n <int>
Algeria	41
Angola	41
Argentina	41
Australia	41
Azerbaijan	41
Brazil	41
Brunei	41
Canada	41
Canadian Oil Sands: Total	0
Chad	41

1-10 of 72 rows

Now we have a tidy dataset. Cleaning data in this way can take time, but it usually saves a lot of time later when working with the data.

8 Merge data

Assume we want to find out how the production of renewable energies is influenced by a country's oil reserves.
We need another dataset for this, which we will import from the Excel file and clean in a similar way to the previous one. But the process changes slightly with each dataset. For example, the

data on renewable energy does not contain any 'n/a' values, so I don't have to convert them to NA and they are already numeric.

```r
 renewable_energy <- read_excel("Statistical Review of World Energy Data.xlsx",
 sheet = "Renewable power - TWh", skip = 2)
 # Select only columns that are relevant for us.
 renewable_energy = renewable_energy %>% select(-59,-60, -61, -62)
 # Delete empty rows
 renewable_energy = renewable_energy %>% drop_na()
 # Rename the country variable.
 renewable_energy = renewable_energy%>% rename(Country= `Terawatt-hours`)
 # Turn the data into the longer format
 renewable_energy = renewable_energy %>% pivot_longer(cols = !Country, names_to = "Year", values_to = "Terawatt_hours")
```

To merge the data, we use a join. A join combines two tables based on variables they have in common. In our case, every table has the variables 'Country' and 'Year'.
There are different types of joins available: Full join, right join, left join and inner join. The inner join only keeps observations are in both datasets. The full join keeps observations that are in one of the two datasets. That means the full join can create 'NA' values. For example, renewable energy data goes back to 1965, but oil reserves data only starts 1980, so a full join would contain NA values for oil reserves from 1965 to 1980. We can conduct both joins to compare:

```r
 Energy_data_full_join = full_join(oil_reserves, renewable_energy, by = join_by("Country", "Year"))
 Energy_data = inner_join(oil_reserves, renewable_energy, by = join_by("Country", "Year"))
```

You see that the output is a new dataframe with four variables: It contains the two common variables 'Country' and 'Year' as well as the variables which we previously only had in one of the datasets, 'Thousand_million_barrels' and 'Terawatt_hours'.
The full join gives a table that is longer, but it also has more NA values. It also contains variables we don't need and did delete from the oil_reserves table, like oil reserves of OPEC or the OECD.
There are also the options to perform a right join or a left join. A right join keeps all observations from the right table, but only the matching ones from the left table. And a left join keeps all from the left table, but only matching ones from the right table.

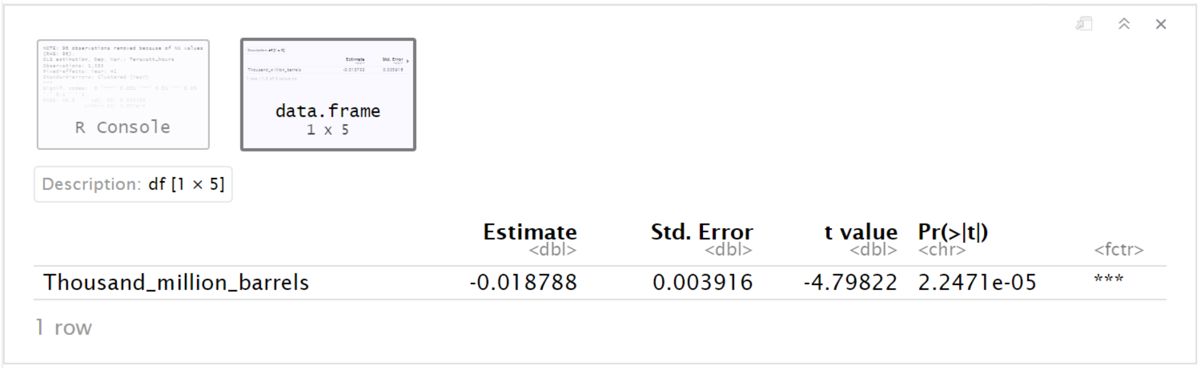
9 A regression

To see if oil reserves have an impact on renewable energy production in a country, let's run a regression.
I like to use the fixest package for regressions.

```r
 #install.packages("fixest")
 library(fixest)
```

A linear regression can be run with the 'feols' command. We will use Terawatt_hours (of renewable energy produced) as yvariable and Thousand_million_barrels (of oil reserves) as x-variable. We will also add Year fixed effects to control for changes over time.

```r
 # Run the regression
 regression_result = feols(Terawatt_hours ~  Thousand_million_barrels | Year, data = Energy_data)
 # Look at the result:
 summary(regression_result, vcov = "cluster")
```



Description: df [1 × 5]

	Estimate <dbl>	Std. Error <dbl>	t value <dbl>	Pr(>|t|) <chr>	<fctr>
Thousand_million_barrels	-0.018788	0.003916	-4.79822	2.2471e-05	***

1 row

As you can see, having large oil reserves has a negative impact on the Terawatt Hours of renewable electricity that are produced.

10 Sources

This workshop is based on the tidyverse, whose main developer Hadley Wickham is one of the authors of "R for Data Science". Their book teaches how to use the tidyverse in more details. It is freely available online: R for Data Science.
The package to run the regression was created by Laurent Berge. His website contains more information on it: Fixest.