Session 2 - Data wrangling

R training

María Reyes Retana
The World Bank | December 2024

THE WORLD BANK
IBRD • IDA | WORLD BANK GROUP
Development Economics • Impact

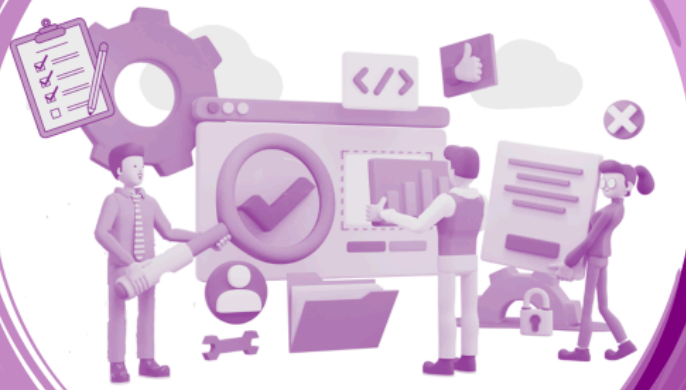
WORLD BANK GROUP
Academy • Development Data

Reproducible Research Repository

Government Analytics and R Training:

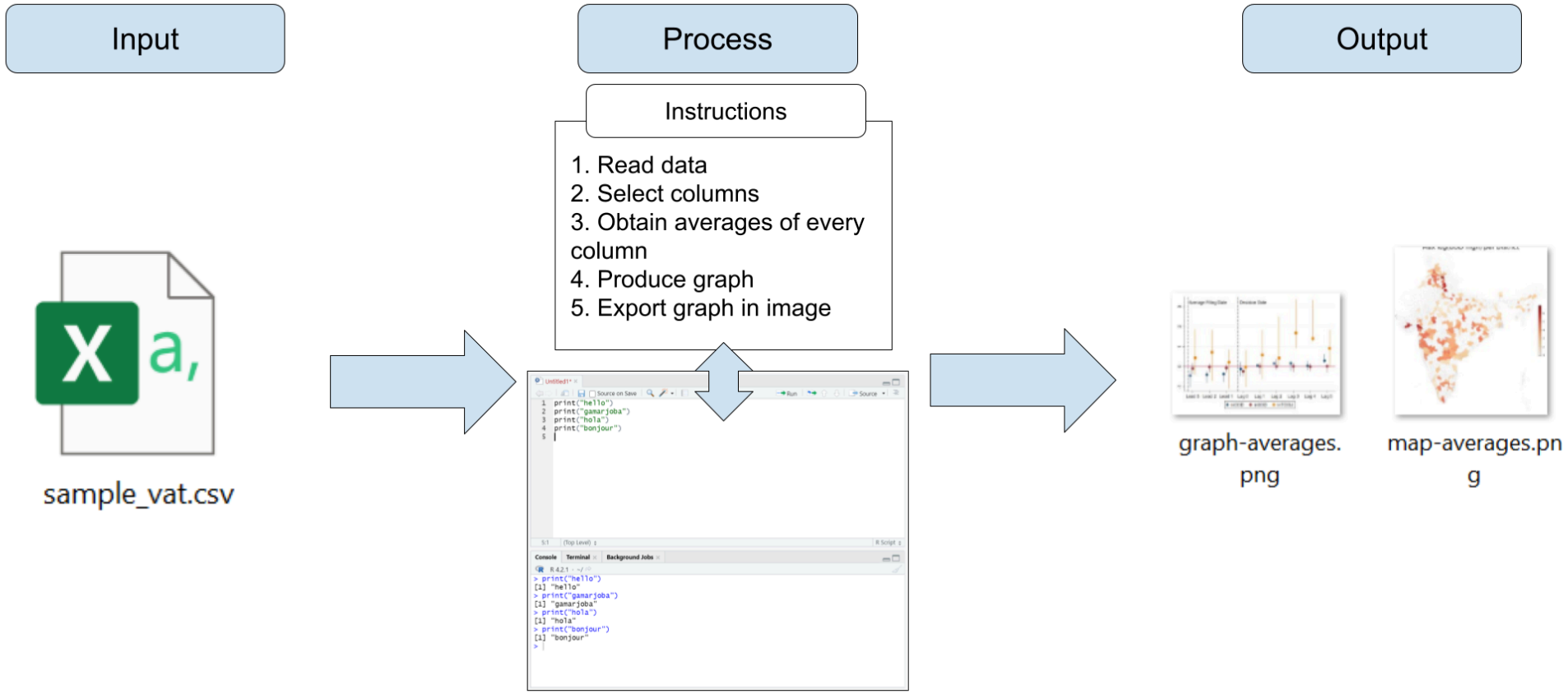
Strengthening Public Sector Reporting and Data Analysis

November 30 - December 3, 2024



About this session

About this session



About this session

Input

Process

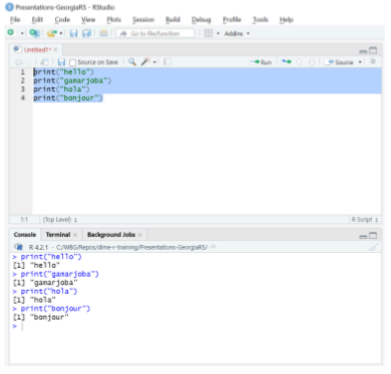
Output

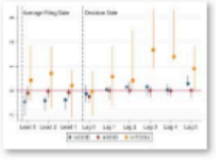
sample_vat.csv

Instructions

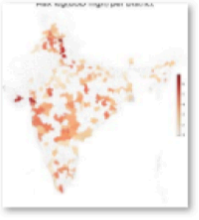
1. Read data
2. Select columns
3. Obtain averages of every column
4. Produce graph
5. Export graph in image

some more of this





graph-averages.png



map-averages.png

and basic implementation of data work pipeline

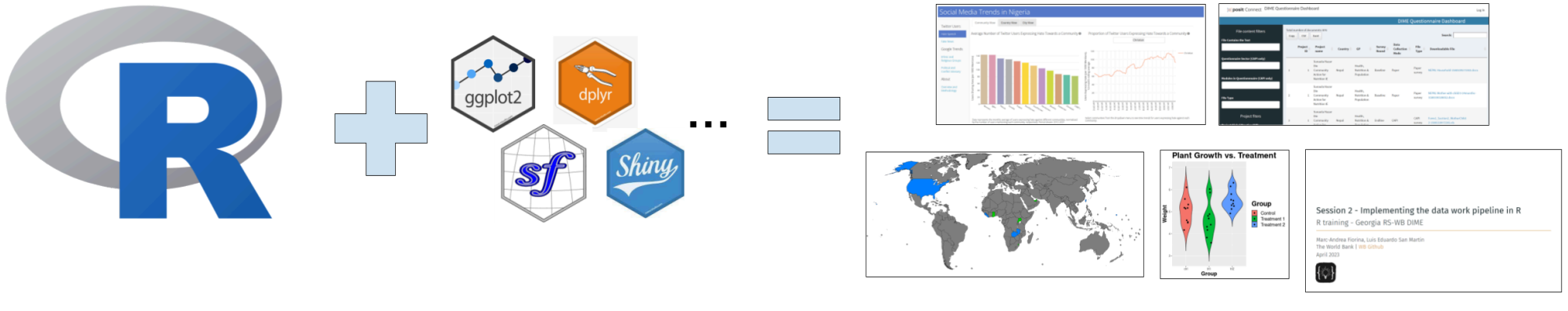
R Packages

R Packages

- Installing R in your computer gives you access to its basic functions
- Additionally, you can also install packages. Packages are a collection of R functions that allow you to do:
    - Operations that basic R functions don't do (example: work with geographic data)
    - Operations that basic R functions do, but easier (example: data wrangling)
- They contain code that other R users have prepared for the community.

R Packages

In a nutshell:



Basic R

Libraries

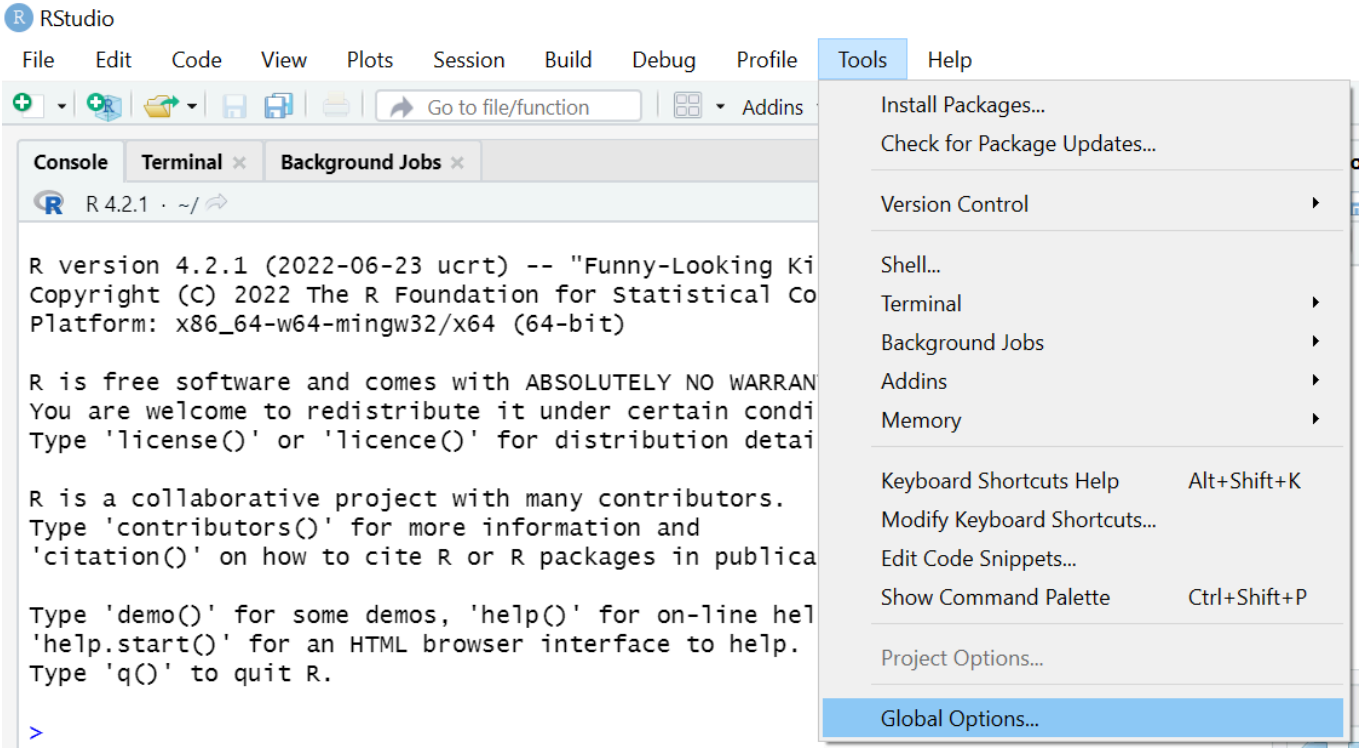
Enhanced R capabilities

Installing R Packages

- Installing packages is usually simple, but it can be challenging in institutional network connections such as the World Bank.
- The next exercise will set up RStudio so that it can install R Packages without problems

Exercise 1: Setting up the installation of packages

1 - In RStudio, go to `Tools` >> `Global Options...`



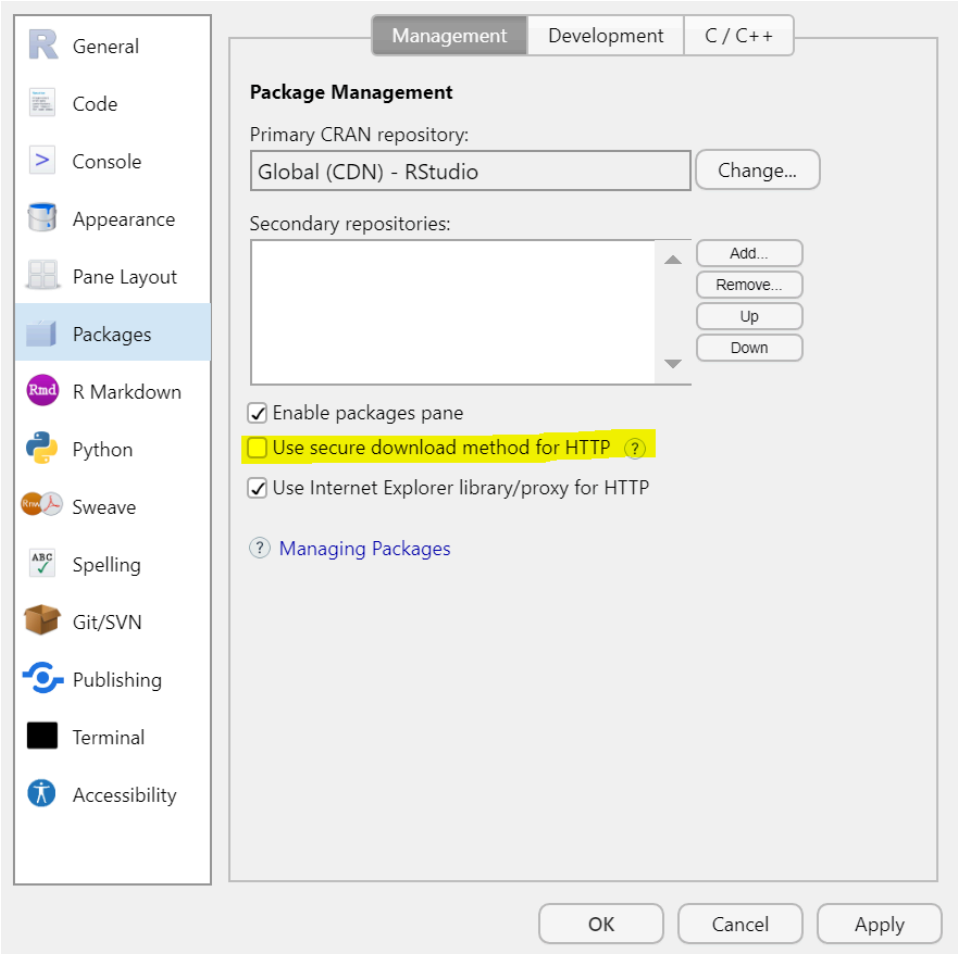
Exercise 1: Setting up the installation of R Packages

2 - Select `Packages` in the left pane

3 - Uncheck `Use secure download method for HTTP`

4 - Click `OK`

You will not see any changes in your RStudio window after this, but now you'll be able to install packages.

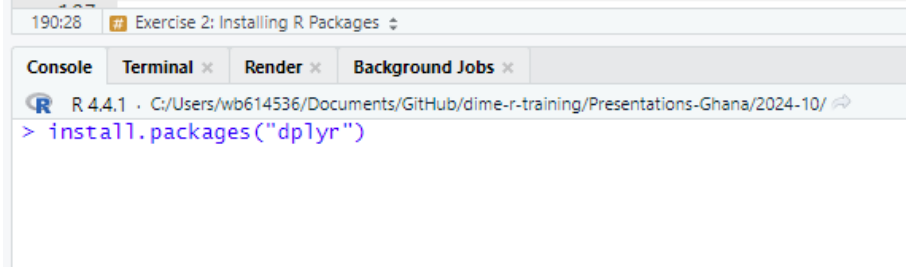


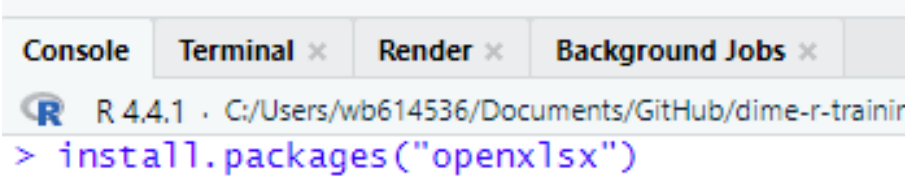
R Packages

We'll use two package in today's session: `dplyr` (really useful library for data wrangling) and `openxlsx` (to work with Excel files)

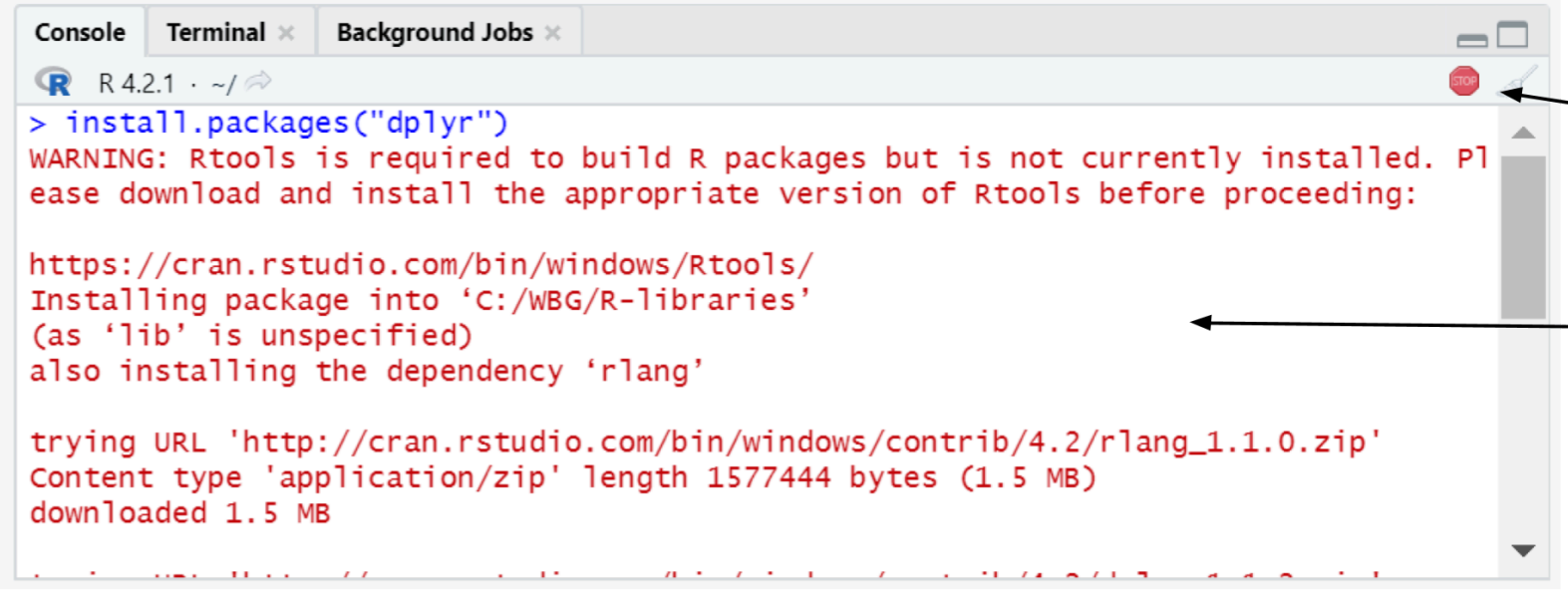
Exercise 2: Installing R Packages

1. Install the R Packages by using `install.packages()`
    - `install.packages("dplyr")`
    - `install.packages("openxlsx")`
    - Note the quotes (`" "`) in the packages names
    - **Introduce this code in the console**, not the script panel





Installing packages



The "STOP" sign means that the code is still running, just wait until it finishes

Note that this message is **not an error**

R packages

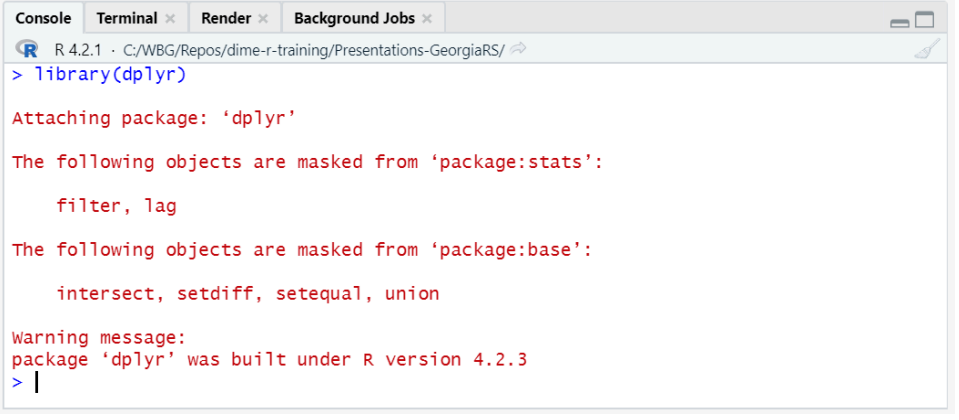
Now that `dplyr` and `openxlsx` are installed, we only need to load them to start using the functions they have.

Exercise 3: Loading packages

1. Open a new script with `File` >> `New File` >> `R Script`
2. Load the packages with:

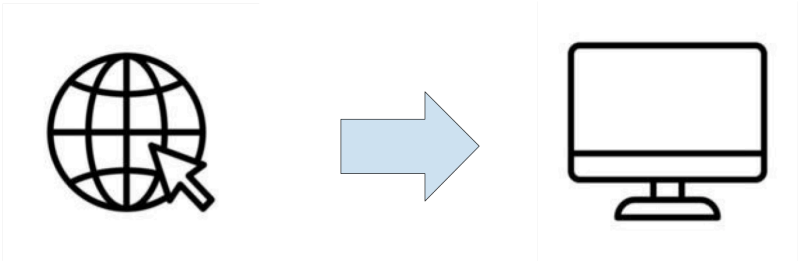
```
library(dplyr)
library(openxlsx)
```

- Run this code from the new script you just opened
- Notice that we don't use quotes in the package names this time

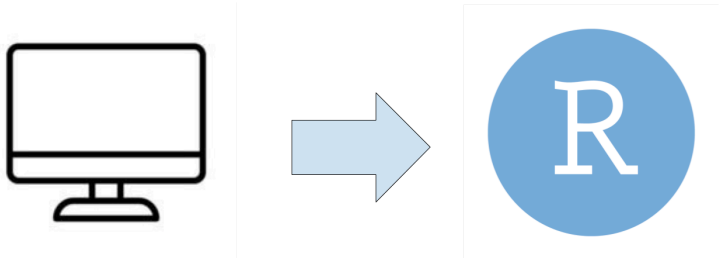


R packages

- Library installation:



- Library loading:



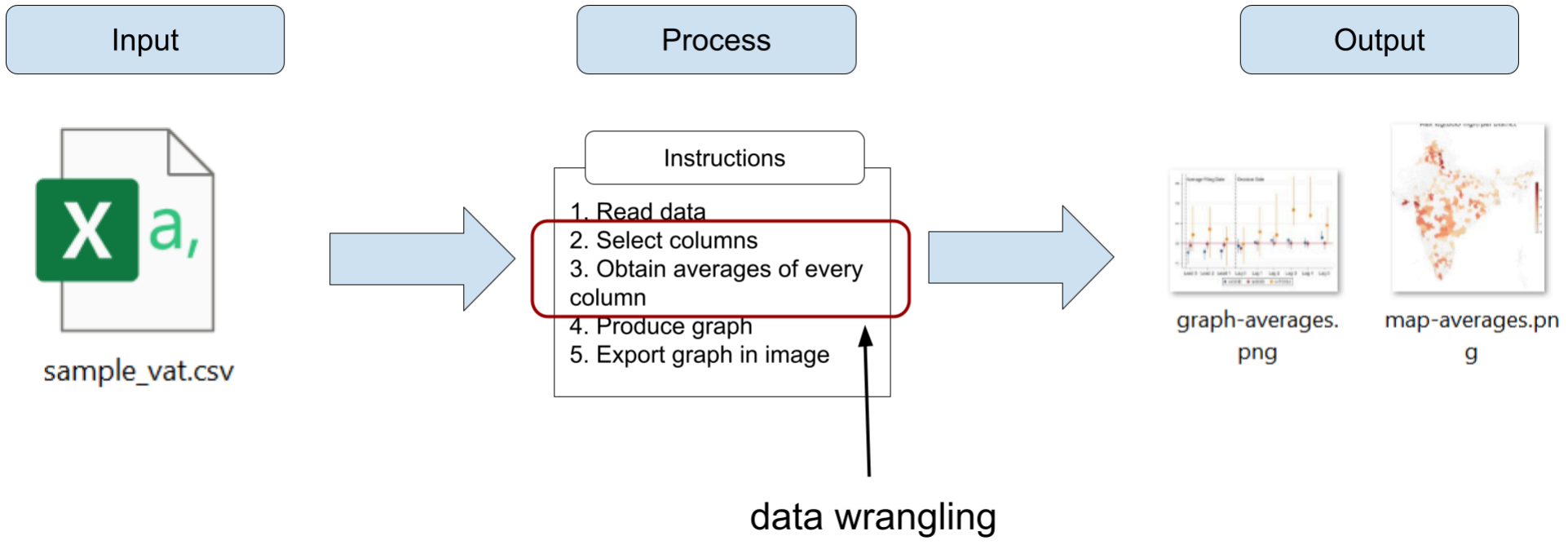
- You install R packages only once in your computer
- You load packages every time you open a new RStudio window (only load the packages you will use)

Data wrangling

Data wrangling

Getting your data ready

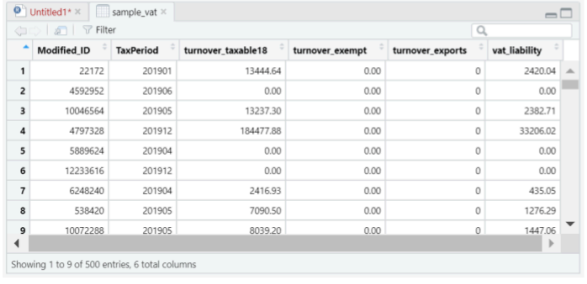
- Data is rarely in a format where it can be converted in an output right away
- In statistical programming, the process of transforming data into a condition where it's ready to be converted into an output is called **data wrangling**



Data wrangling

Getting your data ready

- Data wrangling is one of the most crucial and time-consuming aspects of data work
- It involves not only coding, but also the mental exercise of thinking what is the shape and condition that your dataframe needs to have in order to produce your desired output



	Modified_ID	TaxPeriod	turnover_taxable18	turnover_exempt	turnover_exports	vat_liability
1	22172	201901	13444.64	0.00	0	2420.04
2	4592952	201906	0.00	0.00	0	0.00
3	10046564	201905	13237.30	0.00	0	2382.71
4	4797328	201912	184477.88	0.00	0	33206.02
5	5889624	201904	0.00	0.00	0	0.00
6	12233616	201912	0.00	0.00	0	0.00
7	6248240	201904	2416.93	0.00	0	435.05
8	538420	201905	7090.50	0.00	0	1276.29
9	10072288	201905	8039.20	0.00	0	1447.06

initial dataframe

dataframe needed for outputs

outputs

Data wrangling

Getting your data ready

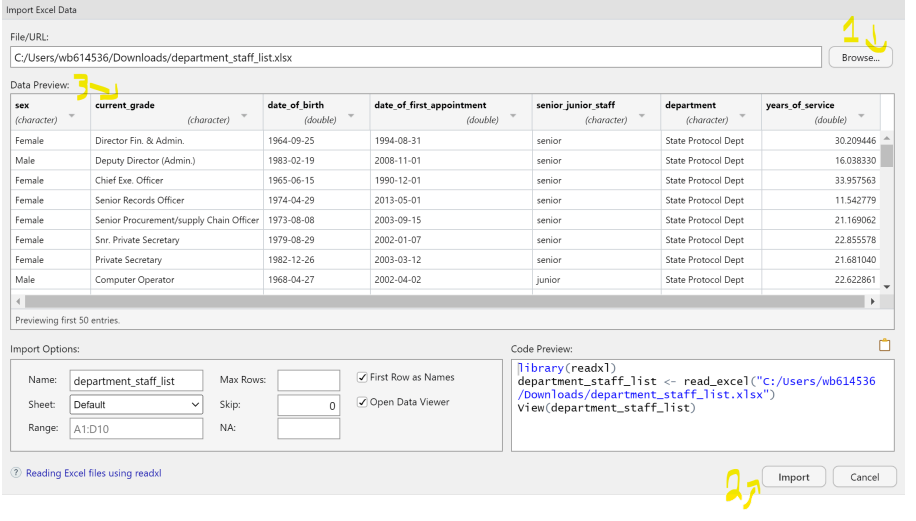
- As we said before we'll use `dplyr` and `tidyr` for data wrangling in this training
- You can also use basic R, but we recommend these packages because its functions are easier to use

dplyr

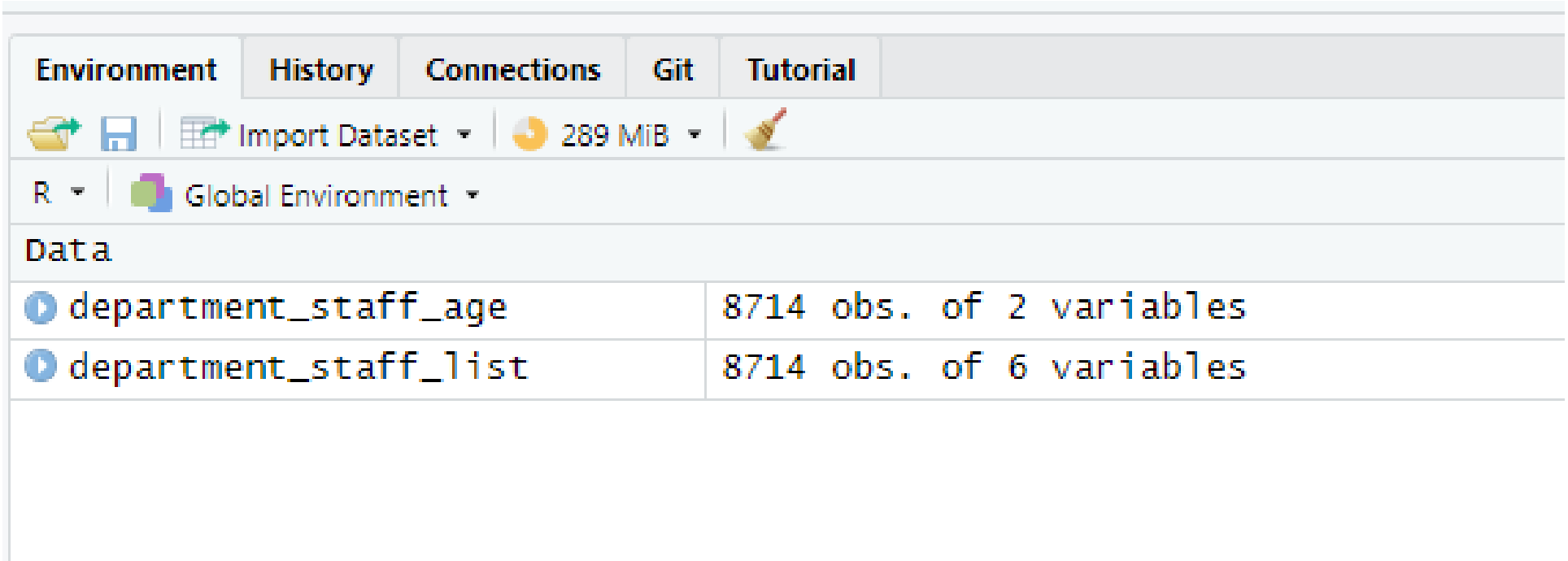
Exercise 4: Loading data

Note that this part of this is the same exercise we did in session 1, but it's okay to repeat it in order to start using a new RStudio session. **If you have RStudio open, start by closing the window and opening RStudio again**.

1. Inyou new RStudio window, go to `File` > `Import Dataset` > `From Excel` and select again the file `department_staff_list.xslx`
    - if you don't know where the file is, check in the `Downloads` folder
    - if you need to download it again, it's here: https://osf.io/2apht
2. Make sure to select `Heading` > `Yes` in the next window
3. Select `Import`
4. Download this new file: https://osf.io/v6psa and repeat steps 1-3 with it



Data wrangling



Note: loading data with a function

- You can also load Excel data with the function `read.xlsx()` instead of using this point-and-click approach
- The **argument** of `read.xlsx()` is the path in your computer where your data is. For example

```
department_staff_list <- read.xlsx("C:/Users/wb614536/Downloads/department_staff_list.xlsx")
```

- As usual, you need to save the result of `read.xlsx()` into a dataframe object with the arrow operator (`<-`) for it to be stored in the environment

Recap: knowing your data

- Dataframe `department_staff_list` is the same dataframe we used last session that contains the data from your department.

```
glimpse(department_staff_list)
```

```
## Rows: 8,754
## Columns: 6
## $ ID                 <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,…
## $ sex                <chr> "Female", "Male", "Female", "Female", "Female", "F…
## $ current_grade      <chr> "Director Fin. & Admin.", "Deputy Director (Admin.…
## $ senior_junior_staff <chr> "senior", "senior", "senior", "senior", "senior", …
## $ department         <chr> "State Protocol Dept", "State Protocol Dept", "Sta…
## $ years_of_service   <dbl> 30.220397, 16.049281, 33.968515, 11.553730, 21.180…
```

Recap: knowing your data

- Now we are introducing a second dataset that only has the ID number (one that I invented), and the age of that person.

```
glimpse(department_staff_age)
```

```
## Rows: 8,754
## Columns: 2
## $ ID  <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,…
## $ age <dbl> 60.15058, 41.74949, 59.43053, 50.55989, 51.28268, 45.22656, 41.900…
```

Data wrangling

- We will only use this second dataframe in one of the next exercises, but we load it now because it's in general a good practice to have data loaded into the memory so it's ready to be used.
- For the next exercises, we will face mention scenarios that could show up in doing your annual reports on in day-to-day operations.

The pipe (%>%) operator

- Before diving into data wrangling, we will to introduce a **super useful tool**:
  the pipe (`%>%`).
- The pipe is part of the `dplyr` package. It helps to write code in a way that is easier to read and understand.
- Reading and understanding multiple operations can be difficult.
- The pipe operator (**%>%**) can help with this.

The pipe operator

- With the pipe, code reads from left to right, top to bottom, which is more intuitive.
- %>% can be read as "then" and simplifies code structure.

For example let's see at this mock get to work sequence example:

Without pipe (%>%) (hard to read and understand the sequence)

```
go_to_work(make_breakfast(work_out(brush_teeth(wake_up(Mer)))))
```

With pipe (%>%) (the order is clear)

```
Mer %>%
  wake_up() %>%
  brush_teeth() %>%
  work_out() %>%
  make_breakfast() %>%
  go_to_work()
```

The pipe

Normally the functions that we will use are organized around a set of verbs, or actions to be taken.

The pipe

Normally the functions that we will use are organized around a set of verbs, or actions to be taken.

- Most *verbs* work as follows:

$$\text{verb}(\underbrace{\text{data.frame}}_{\text{1st argument}}, \underbrace{\text{what to do}}_{\text{2nd argument}})$$

The pipe

Normally the functions that we will use are organized around a set of verbs, or actions to be taken.

- Most *verbs* work as follows:

$$\text{verb}(\underbrace{\text{data.frame}}_{\text{1st argument}}, \underbrace{\text{what to do}}_{\text{2nd argument}})$$

- Alternatively you can (**should**) use the `pipe` operator `%>%`:

$$\underbrace{\text{data.frame}}_{\text{1st argument}} \ \underbrace{\%>\%}_{\text{"pipe" operator}} \ \text{verb}(\underbrace{\text{what to do}}_{\text{2nd argument}})$$

We will start using the pipe from this point. Please ask if something is not clear.

Tip 💡 : Use Shift + Ctrl/Cmd + M as a shortcut for the pipe operator.

Questions?

Filtering and sorting

Filtering and sorting

Filtering and sorting

Data work request

We will use the data from our dataframe department_staff_list

exercises-session2.R* × department_staff_list ×

Filter

	ID	sex	current_grade
1	1	Female	Director Fin. & Admin.
2	2	Male	Deputy Director (Admin.)
3	3	Female	Chief Exe. Officer
4	4	Female	Senior Records Officer
5	5	Female	Senior Procurement/supply
6	6	Female	Snr. Private Secretary
7	7	Female	Private Secretary
8	8	Male	Computer Operator
9	9	Male	Asst. Director IIA
10	10	Female	Private Secretary
11	11	Male	IT/IM Officer

For this we can filter the Female rows from the sex category

Filter

	ID	sex	current_grade
1	1	Female	Director Fin. & Admin.
2	3	Female	Chief Exe. Officer
3	4	Female	Senior Records Officer
4	5	Female	Senior Procurement/supply Chain Officer
5	6	Female	Snr. Private Secretary
6	7	Female	Private Secretary
7	10	Female	Private Secretary
8	12	Female	Stenographer Gd I
9	14	Female	Records Officer
10	15	Female	Principal Exe. Officer
11	16	Female	Steno Secretary
12	17	Female	senior Technical Asst.

Data work request

Now let's say that we are also interested at a first glance of the females that recently joined the department. We would have to do the following

1. Keeping only the female employees
2. Sorting by years of service

Filtering and sorting

1. Keeping only the female employees

Use `filter()` for this:

```
department_female <- department_staff_list %>% # pipe takes data as first argument "and then"
                          filter(sex == "Female") # we filter by Female
```

Remember that we said we were starting to use the pipe (%>%). The pipe takes our data as the first argument, and then we need to pass only the instruction (filter)

exercises-session2.R* × department_female ×

Filter

	ID	sex	current_grade	senior_junior_staff	department	years_of_service
1	1	Female	Director Fin. & Admin.	senior	State Protocol Dept	30.220397
2	3	Female	Chief Exe. Officer	senior	State Protocol Dept	33.968515
3	4	Female	Senior Records Officer	senior	State Protocol Dept	11.553730
4	5	Female	Senior Procurement/supply Chain Officer	senior	State Protocol Dept	21.180014
5	6	Female	Snr. Private Secretary	senior	State Protocol Dept	22.866530
6	7	Female	Private Secretary	senior	State Protocol Dept	21.691992
7	10	Female	Private Secretary	senior	State Protocol Dept	5.056810
8	12	Female	Stenographer Gd I	junior	State Protocol Dept	16.249144
9	14	Female	Records Officer	senior	State Protocol Dept	6.428474
10	15	Female	Principal Exe. Officer	senior	State Protocol Dept	4.908966
11	16	Female	Steno Secretary	senior	State Protocol Dept	15.134839
12	17	Female	senior Technical Asst.	junior	State Protocol Dept	32.134155
13	22	Female	Principal Protocol Officer	senior	State Protocol Dept	22.962355
14	30	Female	State Protocol Officer	senior	State Protocol Dept	5.464750
15	31	Female	State Protocol Officer	senior	State Protocol Dept	5.464750
16	32	Female	State Protocol Officer	senior	State Protocol Dept	5.464750
17	33	Female	Assistant Protocol Officer	senior	State Protocol Dept	5.286790

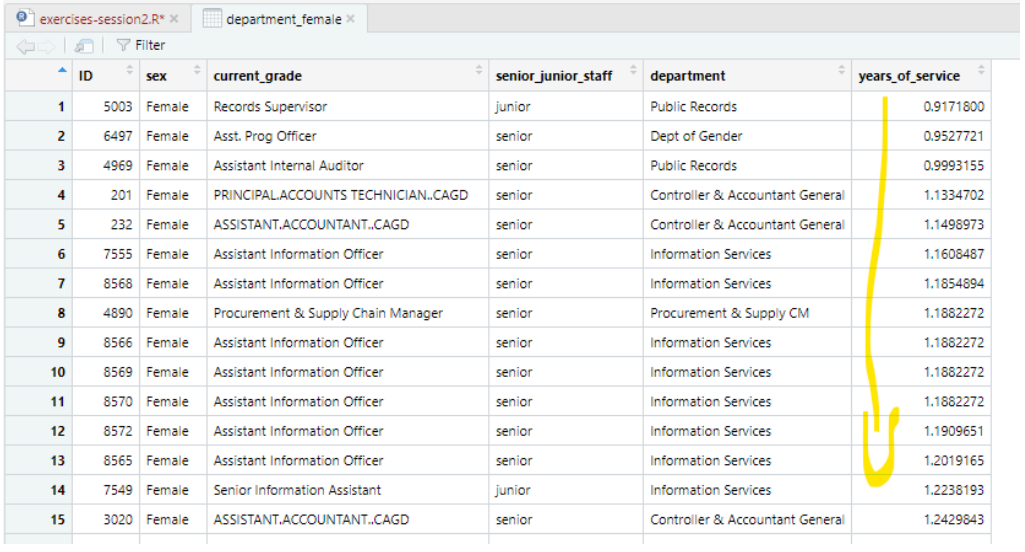
Showing 1 to 18 of 3,559 entries, 6 total columns

Filtering and sorting

2. Sorting by years of service

Use the function `arrange()` to sort. Sortings are ascending by default in R (this will get us from less years of service to more years of service).

```
department_female <- department_staff_list %>%
                          filter(sex == "Female") %>% # pipe takes previous instruction "and then"
                          arrange(years_of_service)  # arrange by years of service
```



exercises-session2.R* × department_female ×

Filter

	ID	sex	current_grade	senior_junior_staff	department	years_of_service
1	5003	Female	Records Supervisor	junior	Public Records	0.9171800
2	6497	Female	Asst. Prog Officer	senior	Dept of Gender	0.9527721
3	4969	Female	Assistant Internal Auditor	senior	Public Records	0.9993155
4	201	Female	PRINCIPAL.ACCOUNTS TECHNICIAN..CAGD	senior	Controller & Accountant General	1.1334702
5	232	Female	ASSISTANT.ACCOUNTANT..CAGD	senior	Controller & Accountant General	1.1498973
6	7555	Female	Assistant Information Officer	senior	Information Services	1.1608487
7	8568	Female	Assistant Information Officer	senior	Information Services	1.1854894
8	4890	Female	Procurement & Supply Chain Manager	senior	Procurement & Supply CM	1.1882272
9	8566	Female	Assistant Information Officer	senior	Information Services	1.1882272
10	8569	Female	Assistant Information Officer	senior	Information Services	1.1882272
11	8570	Female	Assistant Information Officer	senior	Information Services	1.1882272
12	8572	Female	Assistant Information Officer	senior	Information Services	1.1909651
13	8565	Female	Assistant Information Officer	senior	Information Services	1.2019165
14	7549	Female	Senior Information Assistant	junior	Information Services	1.2238193
15	3020	Female	ASSISTANT.ACCOUNTANT..CAGD	senior	Controller & Accountant General	1.2429843

Exercise 5: Now let's do it.. filter and sort your data

We can write the whole code for this in our exercise script. (You can copy and paste the code below)

1.- Filter by `Female`:

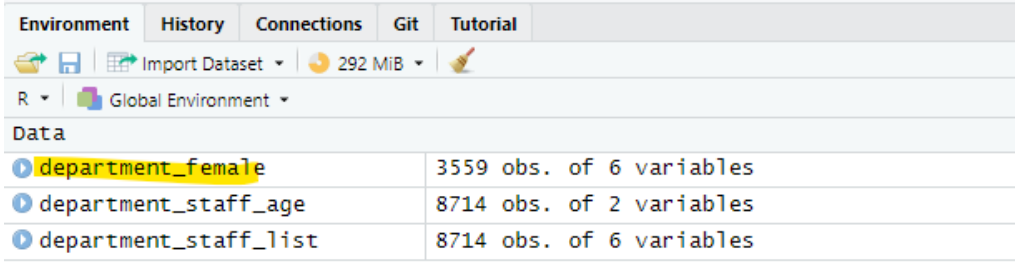
2.- Sort by `years_of_service`:

```
department_female <- department_staff_list %>%
                     filter( sex == "Female") %>%
                     arrange(years_of_service)
```

Filtering and sorting

Some notes:

- `filter()` and `arrange()` are all functions from `dplyr`. Remember you have to always load `dplyr` first with `library(dplyr)` to be able to use them
- The resulting dataframe is `department_female`



Filtering and sorting

- Filtering and sorting are two very common data wrangling operations in statistical programming
- Now we'll review a new data wrangling operation that is also quite common and useful: **mutate**, which basically means creating new variables

Mutate

Mutate

- `mutate` will take a statement like this:

```
mutate(variable_name = some_calculation)
```

- And attach variable_name at the end of the dataset.

Mutate

Example

Using our data frame let's say that we want to include a variable that for any reason instead of years of service we want days of service. We would do this by doing the following:

```
example_mutate <- department_staff_list %>%
  mutate(days_of_service = years_of_service*365)
```

We will not use that variable for our datawork examples, but this is a really useful data wrangling function.

Merging dataframes

Merging dataframes

Merging data is a common task in data analysis, especially when working with large datasets.

Let's see how it would apply to our dataframes.

Data work request

Scenario 2:

Let's imagine that for our annual report we are also interested in the age distribution from the employees

Merging dataframes

Data work request

Use the data `department_staff_list` *that you already know with the department_staff_age*

We want to include the `age` *variable to our dataframe*

Filter

ID	sex	current_grade	senior_junior_staff	department	years_of_service
1	Female	Director Fin. & Admin.	senior	State Protocol Dept	30.220397
2	Male	Deputy Director (Admin.)	senior	State Protocol Dept	16.049281
3	Female	Chief Exe. Officer	senior	State Protocol Dept	33.968515
4	Female	Senior Records Officer	senior	State Protocol Dept	11.553730
5	Female	Senior Procurement/supply Chain Officer	senior	State Protocol Dept	21.180014
6	Female	Snr. Private Secretary	senior	State Protocol Dept	22.866530
7	Female	Private Secretary	senior	State Protocol Dept	21.691992
8	Male	Computer Operator	junior	State Protocol Dept	22.633812
9	Male	Asst. Director IIA	senior	State Protocol Dept	5.379877
10	Female	Private Secretary	senior	State Protocol Dept	5.056810
11	Male	IT/IM Officer	senior	State Protocol Dept	5.043121
12	Female	Stenographer Gd I	junior	State Protocol Dept	16.249144
13	Male	Records Supervisor	junior	State Protocol Dept	12.919918
14	Female	Records Officer	senior	State Protocol Dept	6.428474
15	Female	Principal Exe. Officer	senior	State Protocol Dept	4.908966
16	Female	Steno Secretary	senior	State Protocol Dept	15.134839

Filter

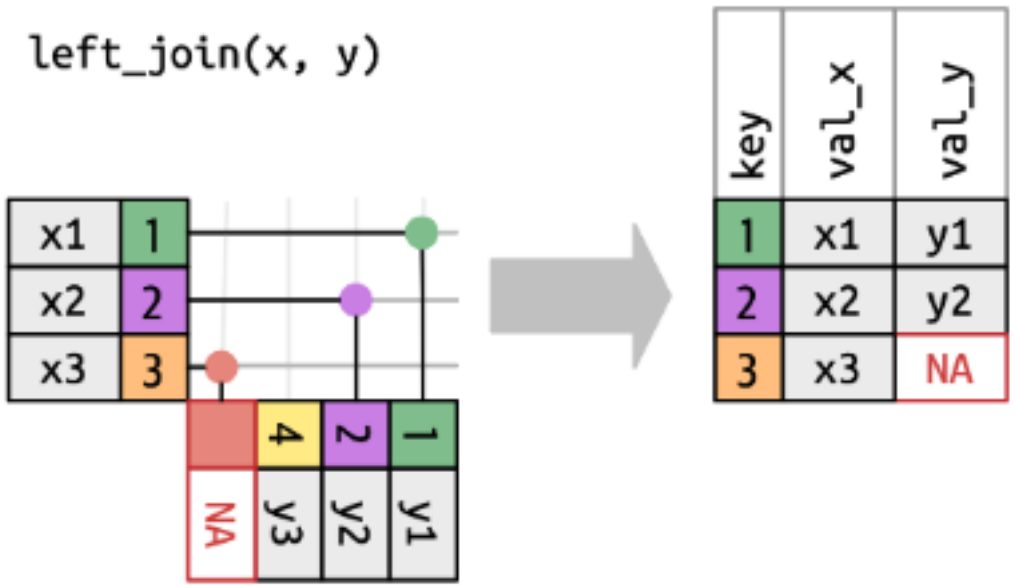
ID	age
1	60.15058
2	41.74949
3	59.43053
4	50.55989
5	51.28268
6	45.22656
7	41.90007
8	56.56400
9	29.09788
10	38.76523
11	32.14784
12	38.01506
13	34.16564
14	43.42505
15	44.99932
16	43.50171

Merging dataframes

To do this we will use `left_join()` to merge the dataframes:

- The arguments of the function are 1. our previous selected data (passed by the pipe) and 2. the dataframe we want to merge to the first one

```
deparment_staff_list_age <- deparment_staff_list %>% # Our original data frame " %>% and then"
                            left_join(department_staff_age) # our second data frame
```

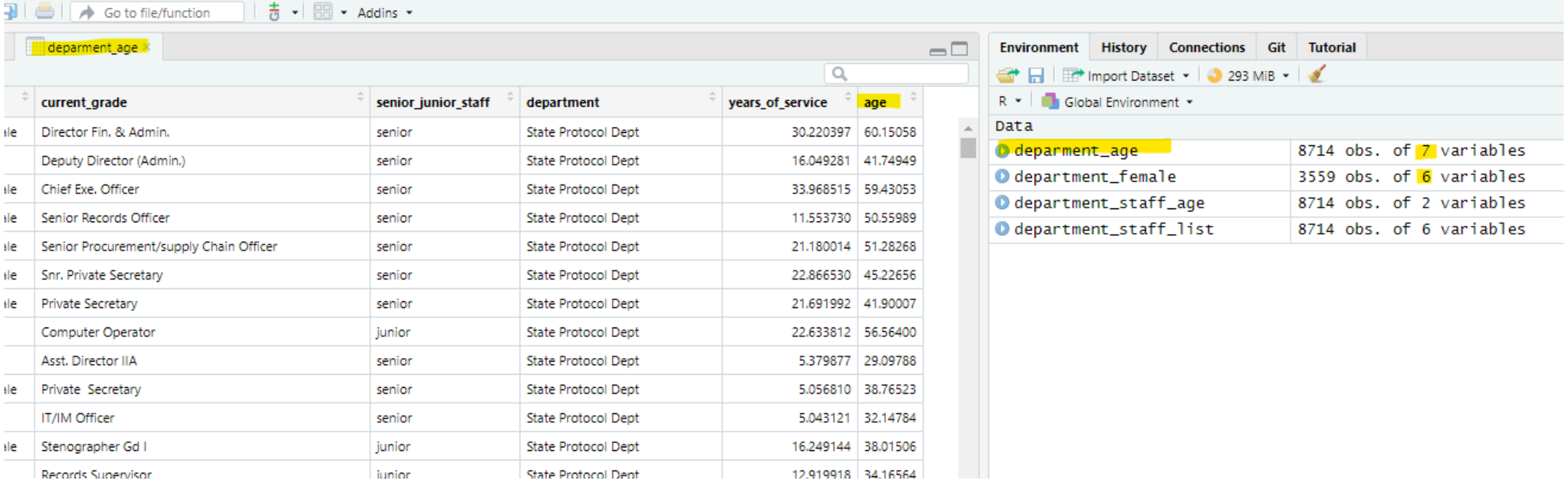


Merging dataframes

Exercise 6: Now let's do it.

- You can copy and paste the code to your exercise script.

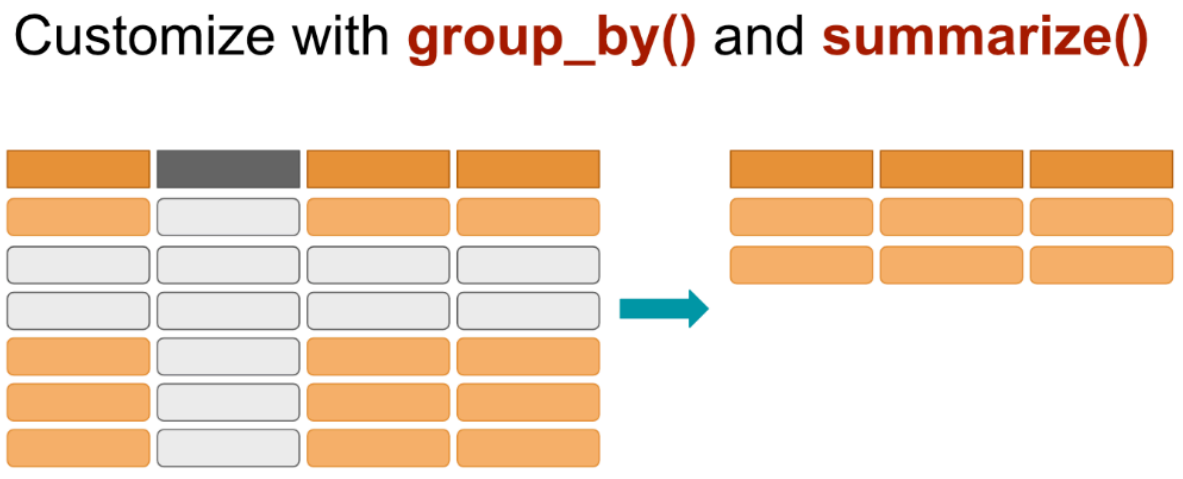
```
deparment_age <- department_staff_list %>% # Our original data frame "and then"
                 left_join(department_staff_age) # our second data frame
```



group_by() and summarize()

We use this when we want to aggregate your data (by groups).

This is one of the most commons operations. Sometimes we want to calculate statistics by groups



Grouping and summarizing

Data work request

Let's say that we are interested again in the number of employees by department

We will use our department_staff_list dataset to do this.

1. The first step is to define the group, in this case `department`

```
employees_by_deparment <- department_staff_list %>%
  group_by(department)
```

If we **only** do this, this won't do anything, to complete the function we need to use this with `summarise()`

Grouping and summarizing

Data work request

summarize works in a similar way to mutate:

```
variable_name = some_calculation
```

In this case the `some_calculation` will be to count the number of employees

```
employees_by_deparment <- department_staff_list %>%
  group_by(department) %>%
  summarise(number = n())
```

Data work request

This will create the following dataframe/table:

```
## # A tibble: 23 × 2
##    department                      number
##    <chr>                            <int>
##  1 Births & Deaths                    179
##  2 Bureau of Languages                 59
##  3 Co-operatives                      293
##  4 Controller & Accountant General   3776
##  5 Department of Comm Devt            109
##  6 Dept of Children                   101
##  7 Dept of Gender                     101
##  8 Dept of Labour                     293
##  9 Factories Inspectorate             122
## 10 Feeder Roads                       103
## # ℹ 13 more rows
```

More wrangling operations

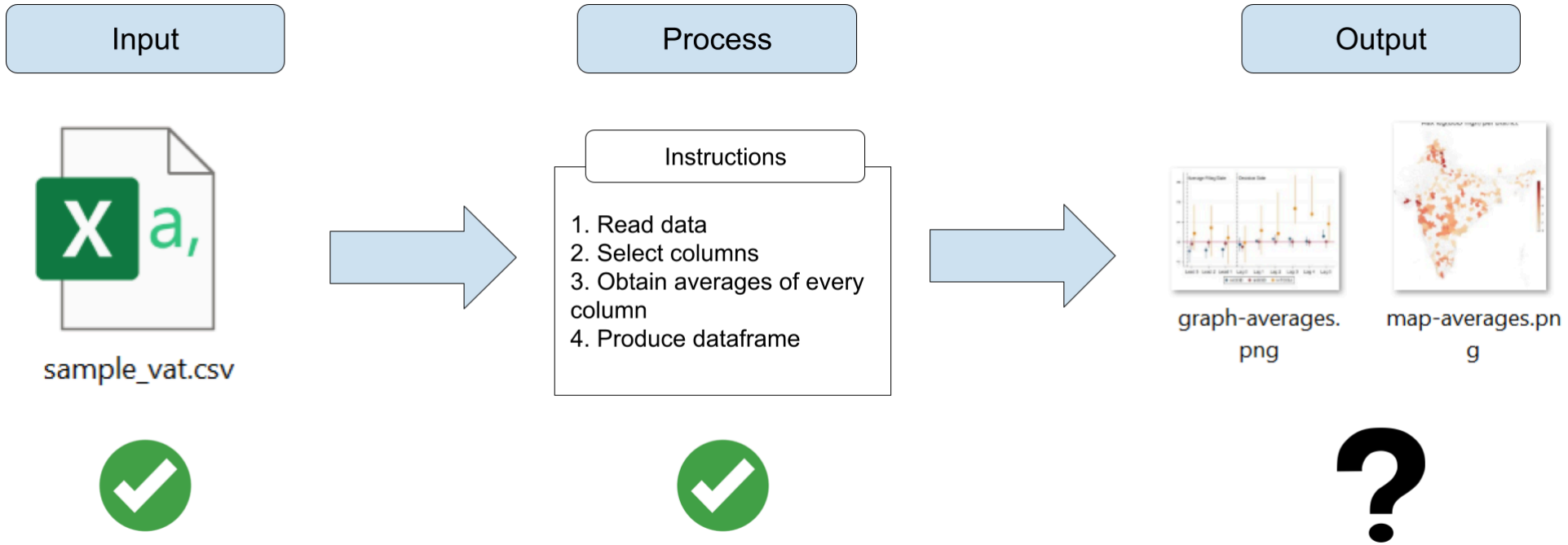
These were two examples we chose to show different possible data wrangling operations. A summary of these and other common operations are:

Operation	Function in `dplyr`
Subset columns	`select()`
Subset rows (based on condition)	`filter()`
Create new columns	`mutate()`
Create new columns based on condition	`mutate()` and `case_when()`
Create new rows	`add_row()`
Merge dataframes	`inner_join()`, `left_join()`, `right_join()`, `full_join()`
Append dataframes	`bind_rows()`
Deduplicate	`distinct()`
Collapse and create summary indicators	`group_by()`, `summarize()`
Pass a result as the first argument for the next function	`%>%` (operator, not function)

Exporting outputs

Exporting outputs

- Until now, we've seen full examples of part 1 and 2 of the data work pipeline
- What about exporting outputs?



- We'll see this in the next exercise

Exporting outputs

Exporting dataframes

- We can export it as an Excel file with the function `write.xlsx()`
- `write.xlsx()` creates a Excel file with the dataframe
- It takes two basic arguments:

    1. The name of the object you want to export
    2. A file path to export the object to

Exporting outputs

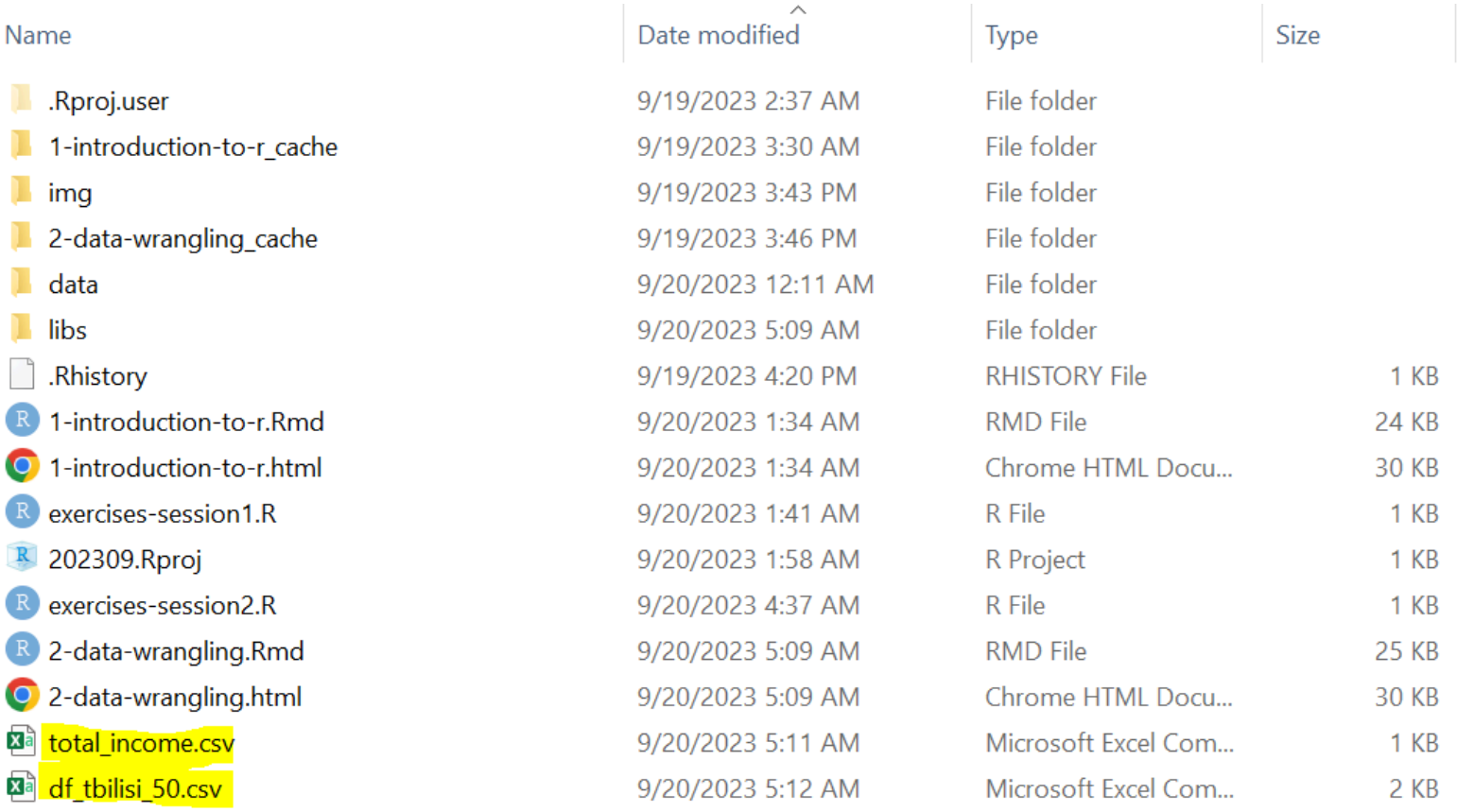
Exercise 7: Export `employees_by_deparment`

1. Use this code to export the results of the last two exercises:

```
write.xslx(employees_by_deparment,
           "employees_by_deparment",
           row.names = FALSE)
```

Exporting outputs

Now `employees_by_deparment.xlsx` (probably in your `Documents` folder).



Name	Date modified	Type	Size
.Rproj.user	9/19/2023 2:37 AM	File folder	
1-introduction-to-r_cache	9/19/2023 3:30 AM	File folder	
img	9/19/2023 3:43 PM	File folder	
2-data-wrangling_cache	9/19/2023 3:46 PM	File folder	
data	9/20/2023 12:11 AM	File folder	
libs	9/20/2023 5:09 AM	File folder	
.Rhistory	9/19/2023 4:20 PM	RHISTORY File	1 KB
1-introduction-to-r.Rmd	9/20/2023 1:34 AM	RMD File	24 KB
1-introduction-to-r.html	9/20/2023 1:34 AM	Chrome HTML Docu...	30 KB
exercises-session1.R	9/20/2023 1:41 AM	R File	1 KB
202309.Rproj	9/20/2023 1:58 AM	R Project	1 KB
exercises-session2.R	9/20/2023 4:37 AM	R File	1 KB
2-data-wrangling.Rmd	9/20/2023 5:09 AM	RMD File	25 KB
2-data-wrangling.html	9/20/2023 5:09 AM	Chrome HTML Docu...	30 KB
total_income.csv	9/20/2023 5:11 AM	Microsoft Excel Com...	1 KB
df_tbilisi_50.csv	9/20/2023 5:12 AM	Microsoft Excel Com...	2 KB

Exporting outputs

Some notes on file paths

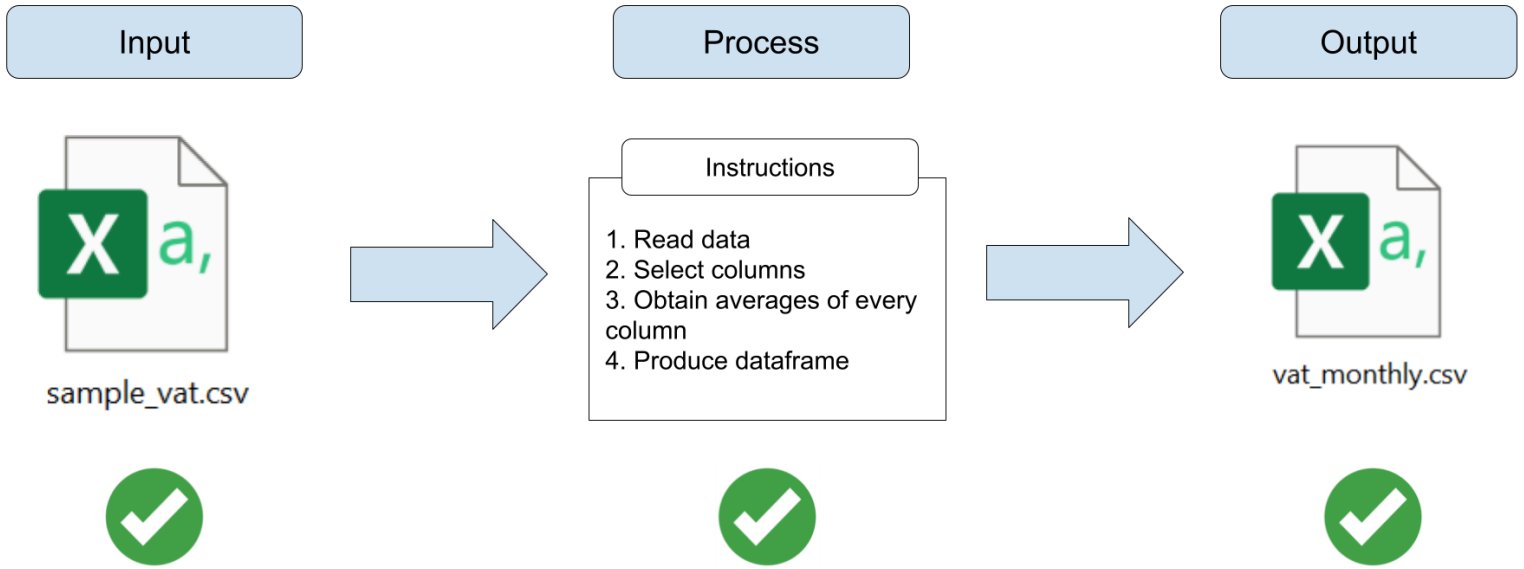
- The second argument of `write.xlsx()` specifies the file path we export the dataframe to

```
write.xslx(employees_by_deparment,
           "employees_by_deparment",
           row.names = FALSE)
```

- You can include any path in your computer and R will write the file in that location
    - For example: `"C:/Users/wb614536/OneDrive - WBG/Desktop"` exports the file to the desktop of my computer (this will not work in other computers)
    - Note that file paths in R use forward slashes (`/`). Back slashes (`\`) **do not work in R**

Exporting outputs

Our data pipeline has been fully implemented at this point. Great!

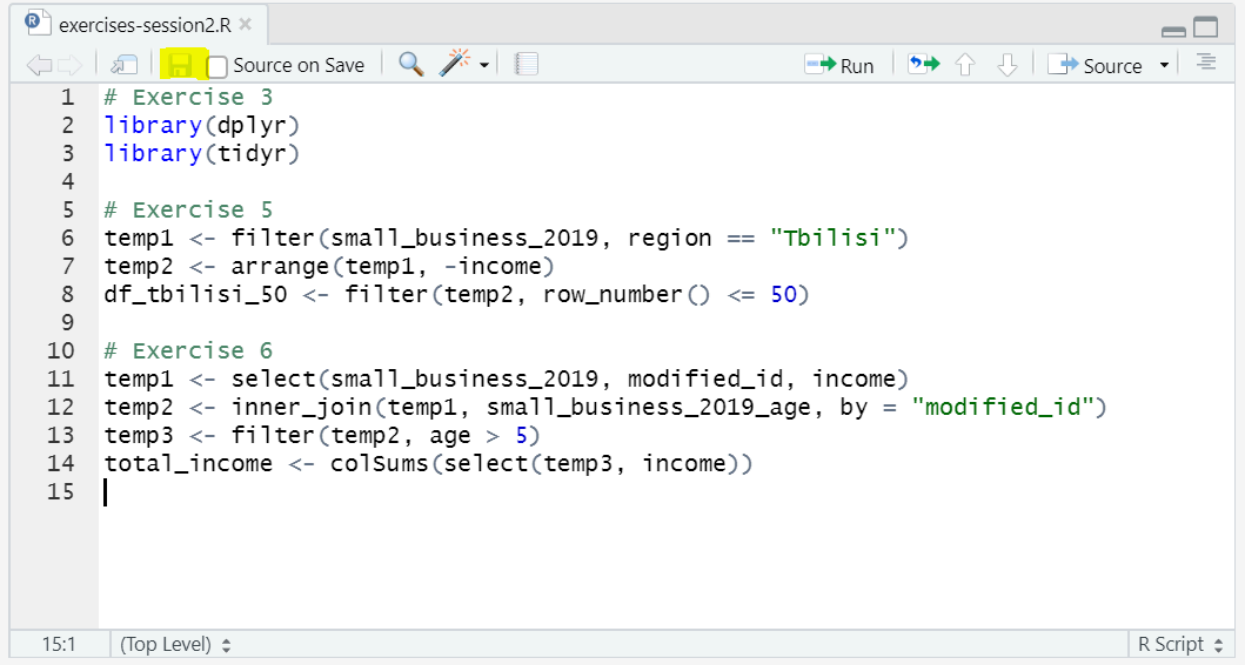


Wrapping up

Wrapping up

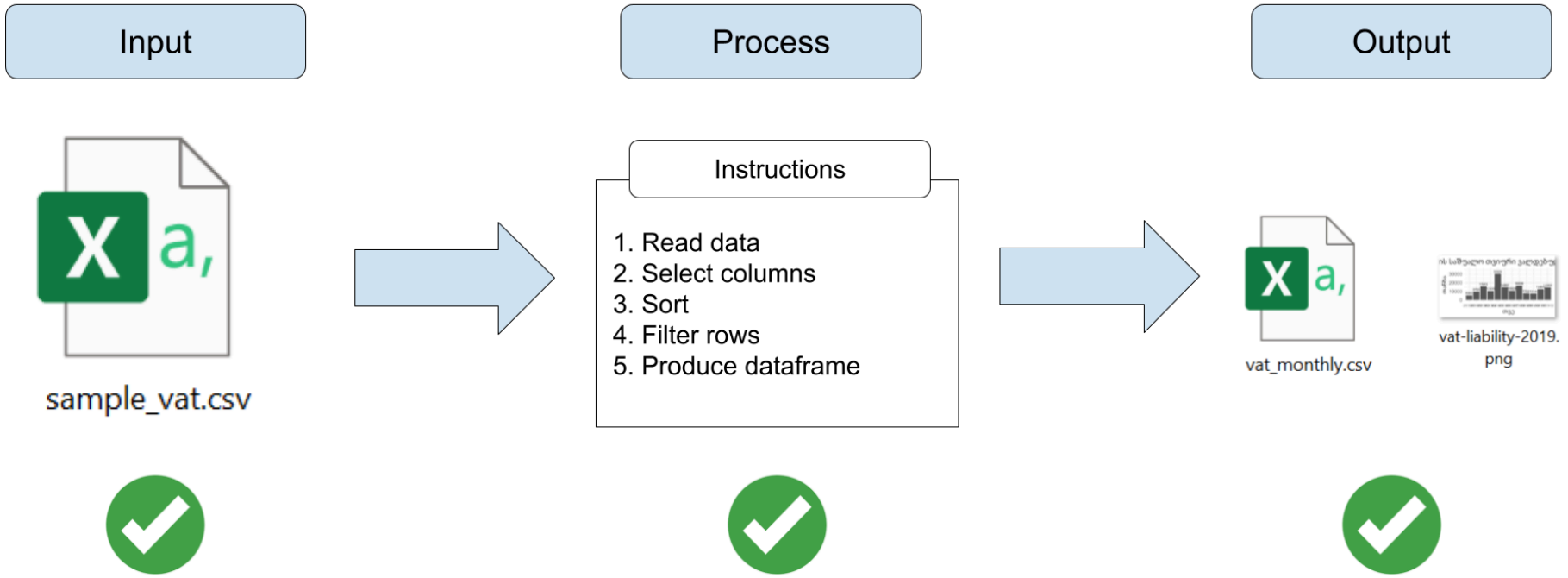
Don't forget to save your work!

- If you haven't, add code comments with `#` to differentiate your solutions for each exercise
- Click the floppy disk to save your work
- Make sure to remember where you're saving your file



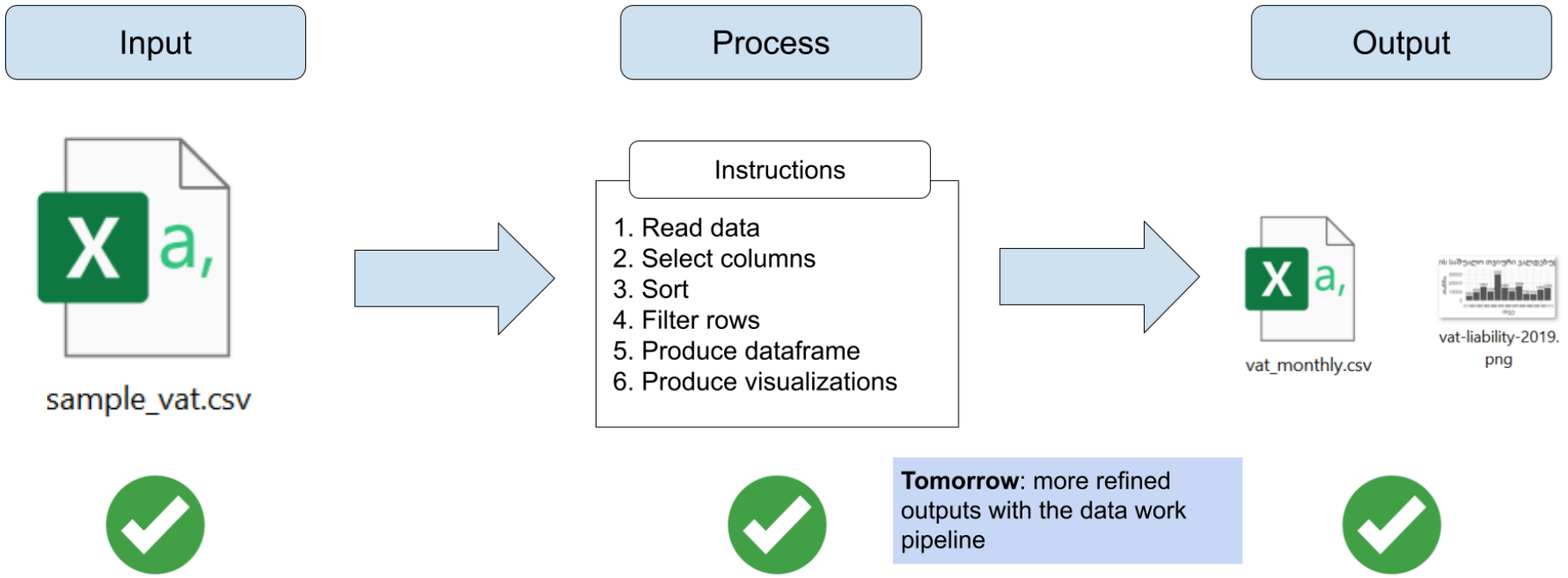
Wrapping up

Data work pipeline



Wrapping up

Data work pipeline



Thanks! // ¡Gracias! // Obrigado!

Appendix

Keeping only relevant columns

Imaging I only want a list with ID and department.

Use `select()` for this:

```
temp1 <- department_staff_list %>%
          select(ID, department)
```

Calculating aggregated columns: total and average income

Use `summarize()` combined with `sum()` and `mean()`:

For our current data this does not apply, but this will be really useful for budget dataframes.

This would look something like this

```
budget_2024 <- budget_data %>%
                group_by(year, product) %>%
                summarise(total = sum(income),
                average = mean(income))
```

Useful links

- **R for Data Science** by Hadley Wickham and Garrett Grolemund
  - Comprehensive introduction to data wrangling and visualization.
  - Read it online for free.
- **Tidyverse Cookbook**
  - Practical solutions for common data wrangling tasks.
  - Tidyverse Cookbook GitHub.
- **Tidyverse Cheat Sheets**
  - Official cheat sheets for `dplyr`, `tidyr`, and other Tidyverse packages.
  - Tidyverse Resources.