Session 3 - Descriptive statistics

R training - Georgia RS-WB DIME

Luis Eduardo San Martin
The World Bank | September 2023

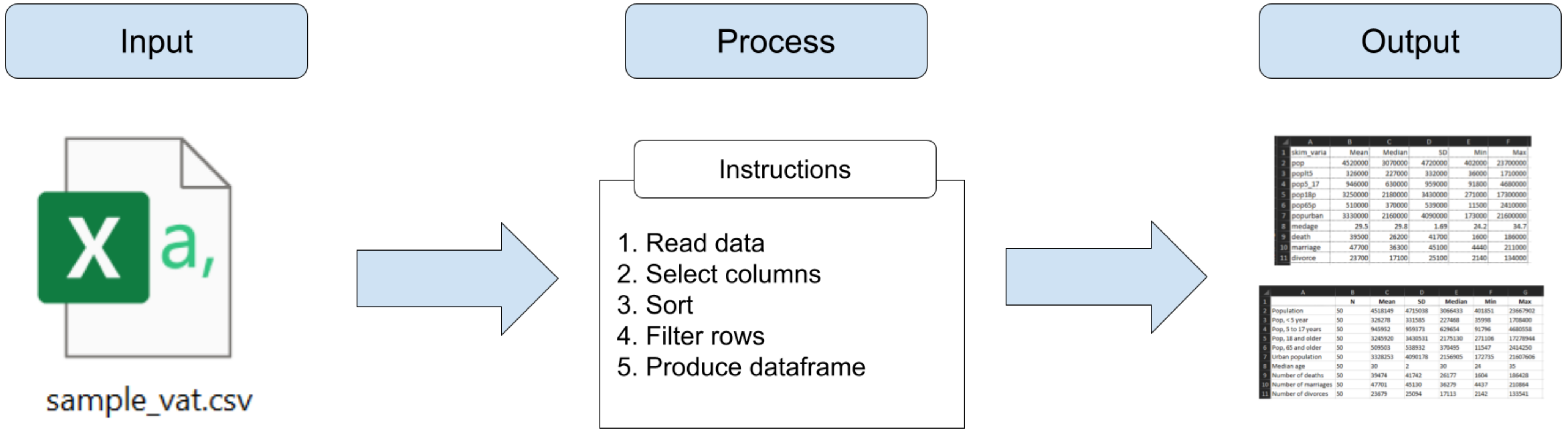
Table of contents // სარჩევი

- Introduction
- Piping
- Quick summary statistics
- Customized summary statistics
- Exporting table
- Customizing table outputs
- Wrapping up

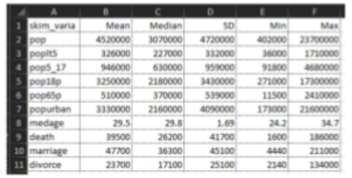
Introduction // გაცნობა

Introduction // გაცნობა

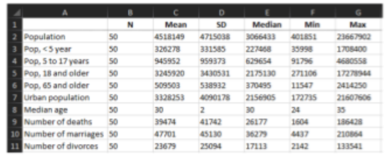
- We learned yesterday how to conduct statistical programming and export the results in `.csv` files
- However, sometime we might need more refined tables than simple (and ugly) CSVs



1. Read data
2. Select columns
3. Sort
4. Filter rows
5. Produce dataframe



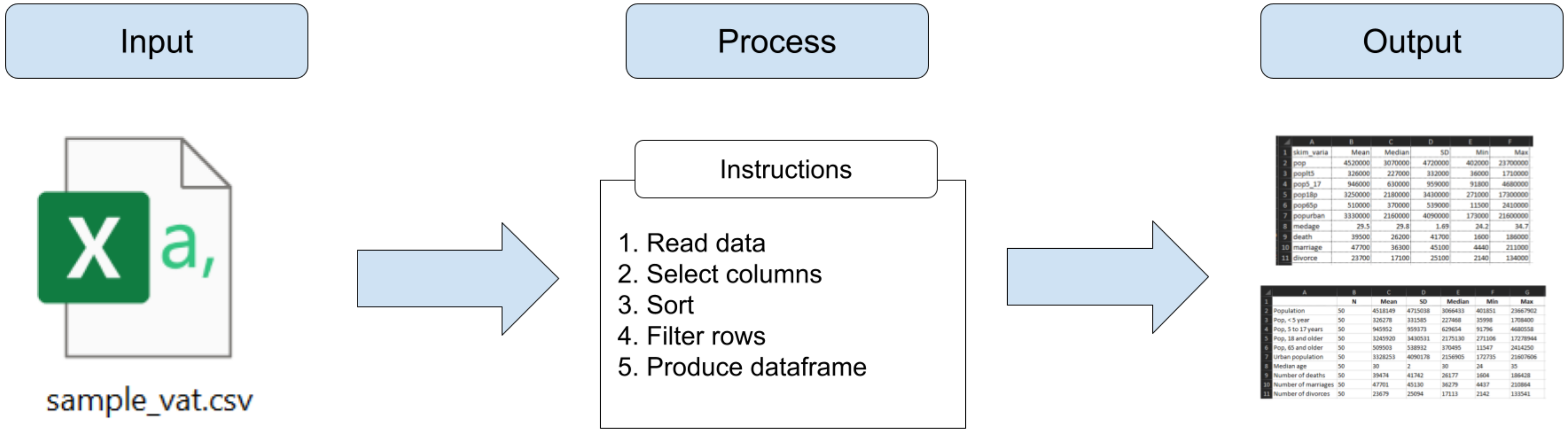
	A	B	C	D	E	F
1	skim_varia	Mean	Median	SD	Min	Max
2	pop	4520000	3070000	4720000	402000	23700000
3	poplt5	326000	227000	332000	36000	1710000
4	pop5_17	946000	630000	959000	91800	4680000
5	pop18p	3250000	2180000	3430000	271000	17300000
6	pop65p	510000	370000	539000	11500	2410000
7	popurban	3330000	2160000	4090000	173000	21600000
8	medage	29.5	29.8	1.69	24.2	34.7
9	death	39500	26200	41700	1600	186000
10	marriage	47700	36300	45100	4440	211000
11	divorce	23700	17100	25100	2140	134000



	A	B	C	D	E	F	G
1		N	Mean	SD	Median	Min	Max
2	Population	50	4518149	4715038	3066433	401851	23667902
3	Pop, < 5 year	50	326278	331585	227468	35998	1708400
4	Pop, 5 to 17 years	50	945952	959373	629654	91796	4680558
5	Pop, 18 and older	50	3245920	3430531	2175130	271106	17278944
6	Pop, 65 and older	50	509503	538932	370495	11547	2414250
7	Urban population	50	3328253	4090178	2156905	172735	21607606
8	Median age	50	30	2	30	24	35
9	Number of deaths	50	39474	41742	26177	1604	186428
10	Number of marriages	50	47701	45130	36279	4437	210864
11	Number of divorces	50	23679	25094	17113	2142	133541

Introduction // გაცნობა

- That's what today's session is about, along with an explanation of the pipes (`%>%`)



1. Read data
2. Select columns
3. Sort
4. Filter rows
5. Produce dataframe

	A	B	C	D	E	F
1	skim_varia	Mean	Median	SD	Min	Max
2	pop	4520000	3070000	4720000	402000	23700000
3	poplt5	326000	227000	332000	36000	1710000
4	pop5_17	946000	630000	959000	91800	4680000
5	pop18p	3250000	2180000	3430000	271000	17300000
6	pop65p	510000	370000	539000	11500	2410000
7	popurban	3330000	2160000	4090000	173000	21600000
8	medage	29.5	29.8	1.69	24.2	34.7
9	death	39500	26200	41700	1600	186000
10	marriage	47700	36300	45100	4440	211000
11	divorce	23700	17100	25100	2140	134000

	A	B	C	D	E	F	G
1		N	Mean	SD	Median	Min	Max
2	Population	50	4518149	4715038	3066433	401851	23667902
3	Pop, <5 year	50	326278	331585	227468	35998	1708400
4	Pop, 5 to 17 years	50	945952	959373	629654	91796	4680558
5	Pop, 18 and older	50	3245920	3430531	2175130	271106	17278944
6	Pop, 65 and older	50	509503	538932	370495	11547	2414250
7	Urban population	50	3328253	4090178	2156905	172735	21607606
8	Median age	50	30	2	30	24	35
9	Number of deaths	50	39474	41742	26177	1604	186428
10	Number of marriages	50	47701	45130	36279	4437	210864
11	Number of divorces	50	23679	25094	17113	2142	133541

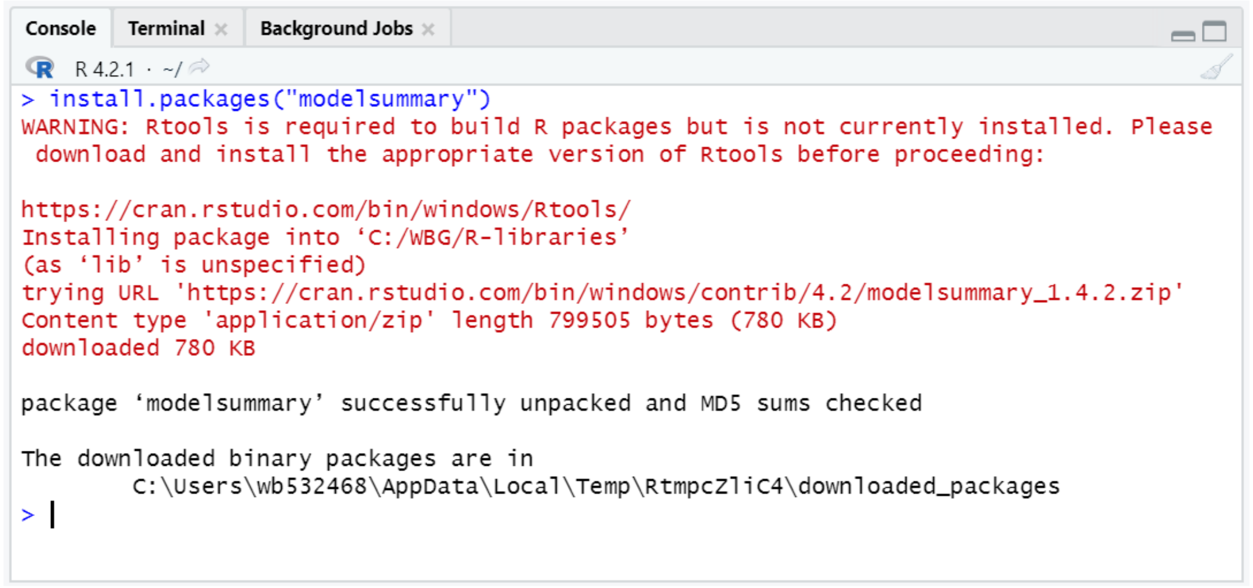
Introduction // გაცნობა

Exercise 1a: Getting the libraries for today's session

We're going to use two R libraries in this session: `modelsummary` and `huxtable`.

1. Install `modelsummary` and `huxtable`:

```
install.packages("modelsummary")
install.packages("huxtable")
```

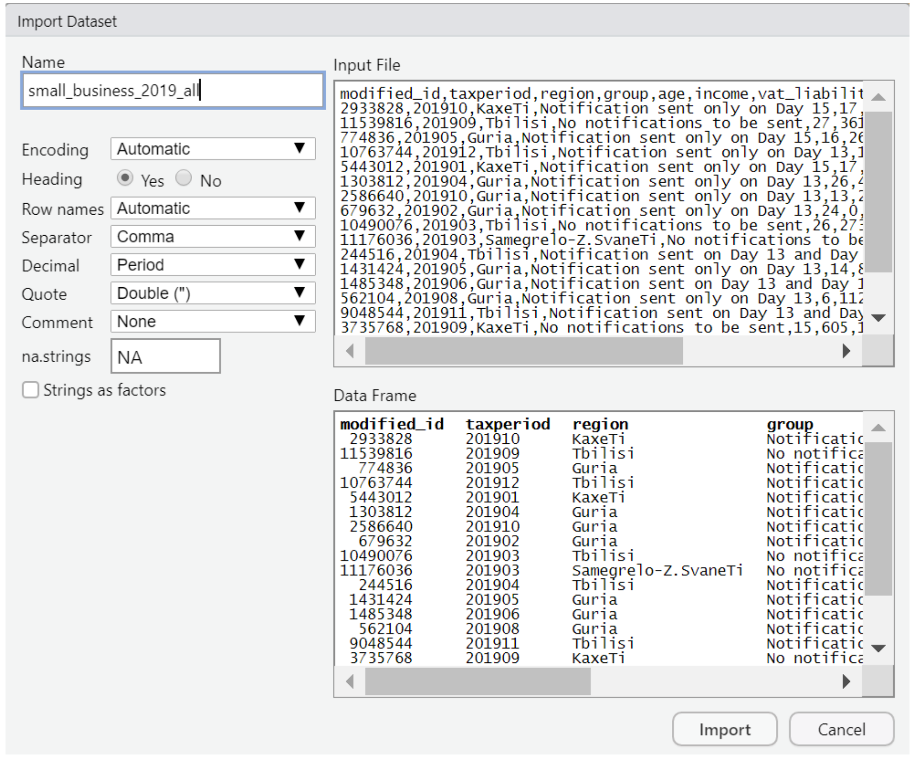


```
> install.packages("modelsummary")
WARNING: Rtools is required to build R packages but is not currently installed. Please
 download and install the appropriate version of Rtools before proceeding:

https://cran.rstudio.com/bin/windows/Rtools/
Installing package into ‘C:/WBG/R-libraries’
(as ‘lib’ is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.2/modelsummary_1.4.2.zip'
Content type 'application/zip' length 799505 bytes (780 KB)
downloaded 780 KB

package ‘modelsummary’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\wb532468\AppData\Local\Temp\RtmpcZ1iC4\downloaded_packages
> 
```

Introduction // გაცნობა

Exercise 1b: Download and load the data we'll use

1. Go to https://osf.io/z8snr and download the file
2. In RStudio, go to `File` > `Import Dataset` > `From Text (base)` and select the file `small_business_2019_all.csv`
   - If you don't know where the file is, remember to check in your `Downloads` folder
3. Select `Import`



Name: small_business_2019_all

Encoding	Automatic
Heading	[✓ Yes] No
Row names	Automatic
Separator	Comma
Decimal	Period
Quote	Double (")
Comment	None
na.strings	NA
Strings as factors	☐

Input File

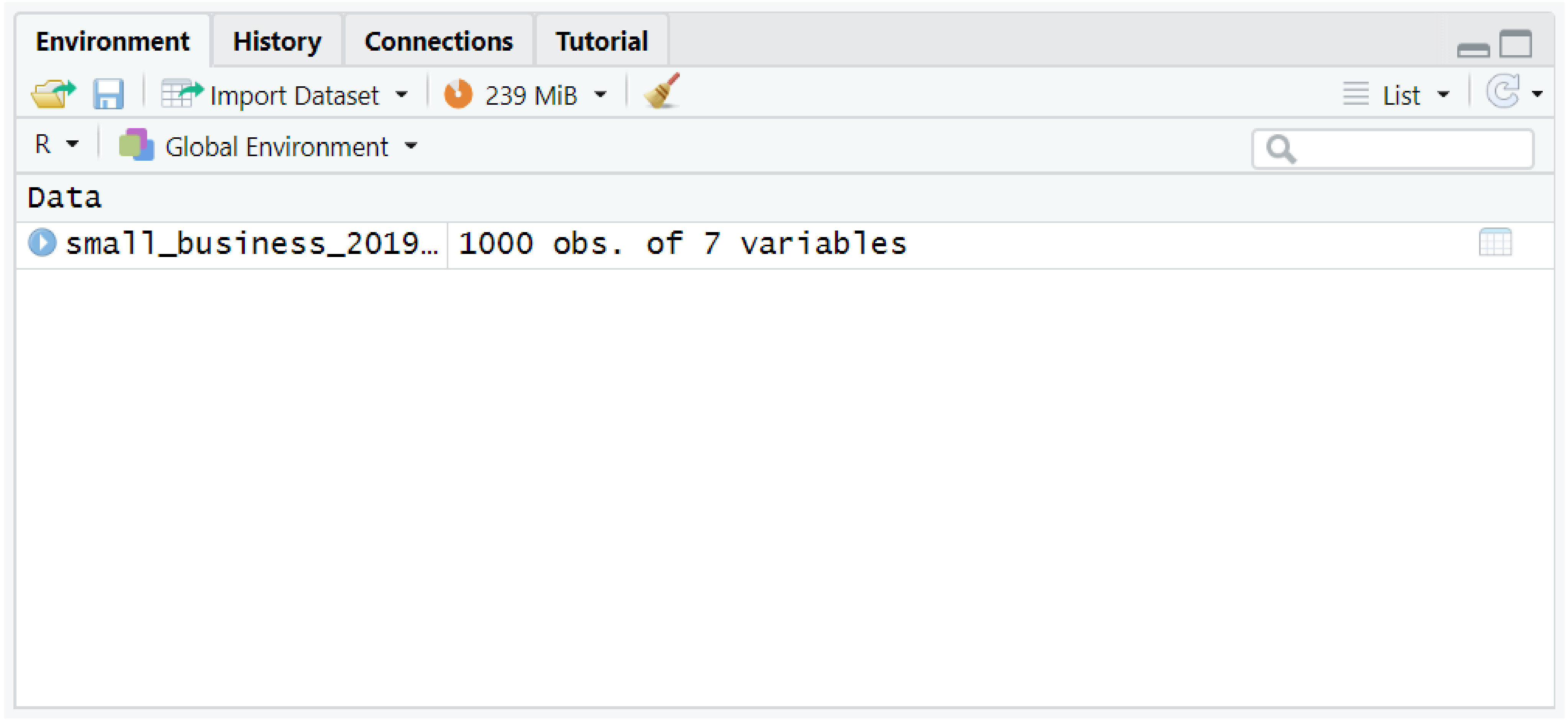
```
modified_id,taxperiod,region,group,age,income,vat_liabilit
2933828,201910,KaxeTi,Notification sent only on Day 15,17,
11539816,201909,Tbilisi,No notifications to be sent,27,361
774836,201905,Guria,Notification sent only on Day 15,16,26
10763744,201912,Tbilisi,Notification sent only on Day 13,1
5443012,201901,KaxeTi,Notification sent only on Day 15,17,
1303812,201904,Guria,Notification sent only on Day 13,26,4
2586640,201910,Guria,Notification sent only on Day 13,13,2
679632,201902,Guria,Notification sent only on Day 13,24,0,
10490076,201903,Tbilisi,No notifications to be sent,26,273
11176036,201903,Samegrelo-Z.SvaneTi,No notifications to be
244516,201904,Tbilisi,Notification sent on Day 13 and Day
1431424,201905,Guria,Notification sent only on Day 13,14,8
1485348,201906,Guria,Notification sent on Day 13 and Day 1
562104,201908,Guria,Notification sent only on Day 13,6,112
9048544,201911,Tbilisi,Notification sent on Day 13 and Day
3735768,201909,KaxeTi,No notifications to be sent,15,605,1
```

Data Frame

```
modified_id  taxperiod  region               group
  2933828    201910     KaxeTi               Notificatic
 11539816    201909     Tbilisi              No notifica
   774836    201905     Guria                Notificatic
 10763744    201912     Tbilisi              Notificatic
  5443012    201901     KaxeTi               Notificatic
  1303812    201904     Guria                Notificatic
  2586640    201910     Guria                Notificatic
   679632    201902     Guria                Notificatic
 10490076    201903     Tbilisi              No notifica
 11176036    201903     Samegrelo-Z.SvaneTi  No notifica
   244516    201904     Tbilisi              Notificatic
  1431424    201905     Guria                Notificatic
  1485348    201906     Guria                Notificatic
   562104    201908     Guria                Notificatic
  9048544    201911     Tbilisi              Notificatic
  3735768    201909     KaxeTi               No notifica
```

Introduction // გაცნობა

You should have one dataframe loaded in the environment after this.

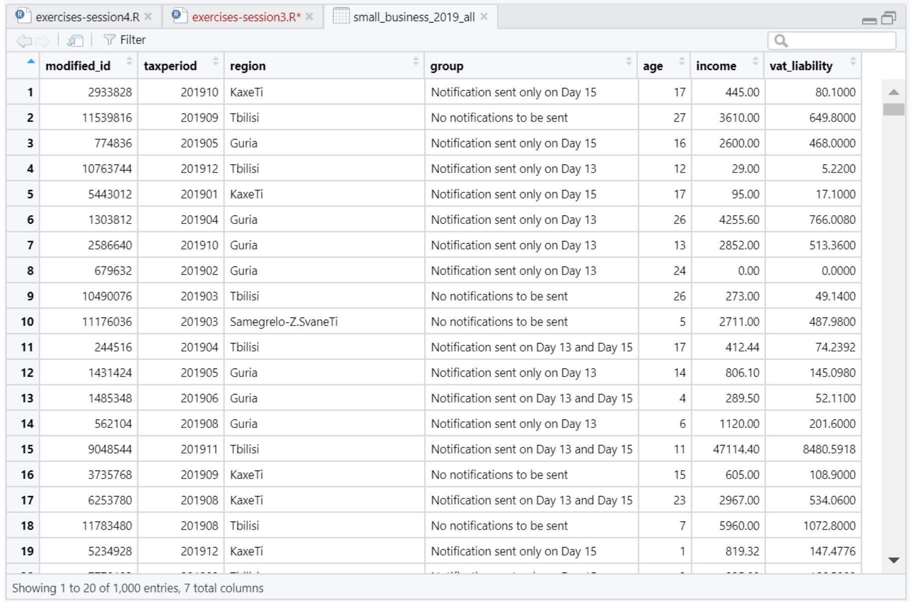


```
Data
small_business_2019…  1000 obs. of 7 variables
```

Introduction // გაცნობა

Recap: always know your data!

- This data is similar to the one we used before
- Every row is one business in one tax period (month)
- `modified_id` is a business identifier
- We also have information about the region, firm age, monthly income, VAT liability
- There is one more variable we didn't see before: `group` contains the group the firm was assigned to in a random experiment



	modified_id	taxperiod	region	group	age	income	vat_liability
1	2933828	201910	KaxeTi	Notification sent only on Day 15	17	445.00	80.1000
2	11539816	201909	Tbilisi	No notifications to be sent	27	3610.00	649.8000
3	774836	201905	Guria	Notification sent only on Day 15	16	2600.00	468.0000
4	10763744	201912	Tbilisi	Notification sent only on Day 13	12	29.00	5.2200
5	5443012	201901	KaxeTi	Notification sent only on Day 15	17	95.00	17.1000
6	1303812	201904	Guria	Notification sent only on Day 13	26	4255.60	766.0080
7	2586640	201910	Guria	Notification sent only on Day 13	13	2852.00	513.3600
8	679632	201902	Guria	Notification sent only on Day 13	24	0.00	0.0000
9	10490076	201903	Tbilisi	No notifications to be sent	26	273.00	49.1400
10	11176036	201903	Samegrelo-Z.SvaneTi	No notifications to be sent	5	2711.00	487.9800
11	244516	201904	Tbilisi	Notification sent on Day 13 and Day 15	17	412.44	74.2392
12	1431424	201905	Guria	Notification sent only on Day 13	14	806.10	145.0980
13	1485348	201906	Guria	Notification sent on Day 13 and Day 15	4	289.50	52.1100
14	562104	201908	Guria	Notification sent only on Day 13	6	1120.00	201.6000
15	9048544	201911	Tbilisi	Notification sent on Day 13 and Day 15	11	47114.40	8480.5918
16	3735768	201909	KaxeTi	No notifications to be sent	15	605.00	108.9000
17	6253780	201908	KaxeTi	Notification sent on Day 13 and Day 15	23	2967.00	534.0600
18	11783480	201908	Tbilisi	No notifications to be sent	7	5960.00	1072.8000
19	5234928	201912	KaxeTi	Notification sent only on Day 15	1	819.32	147.4776

Showing 1 to 20 of 1,000 entries, 7 total columns

Piping

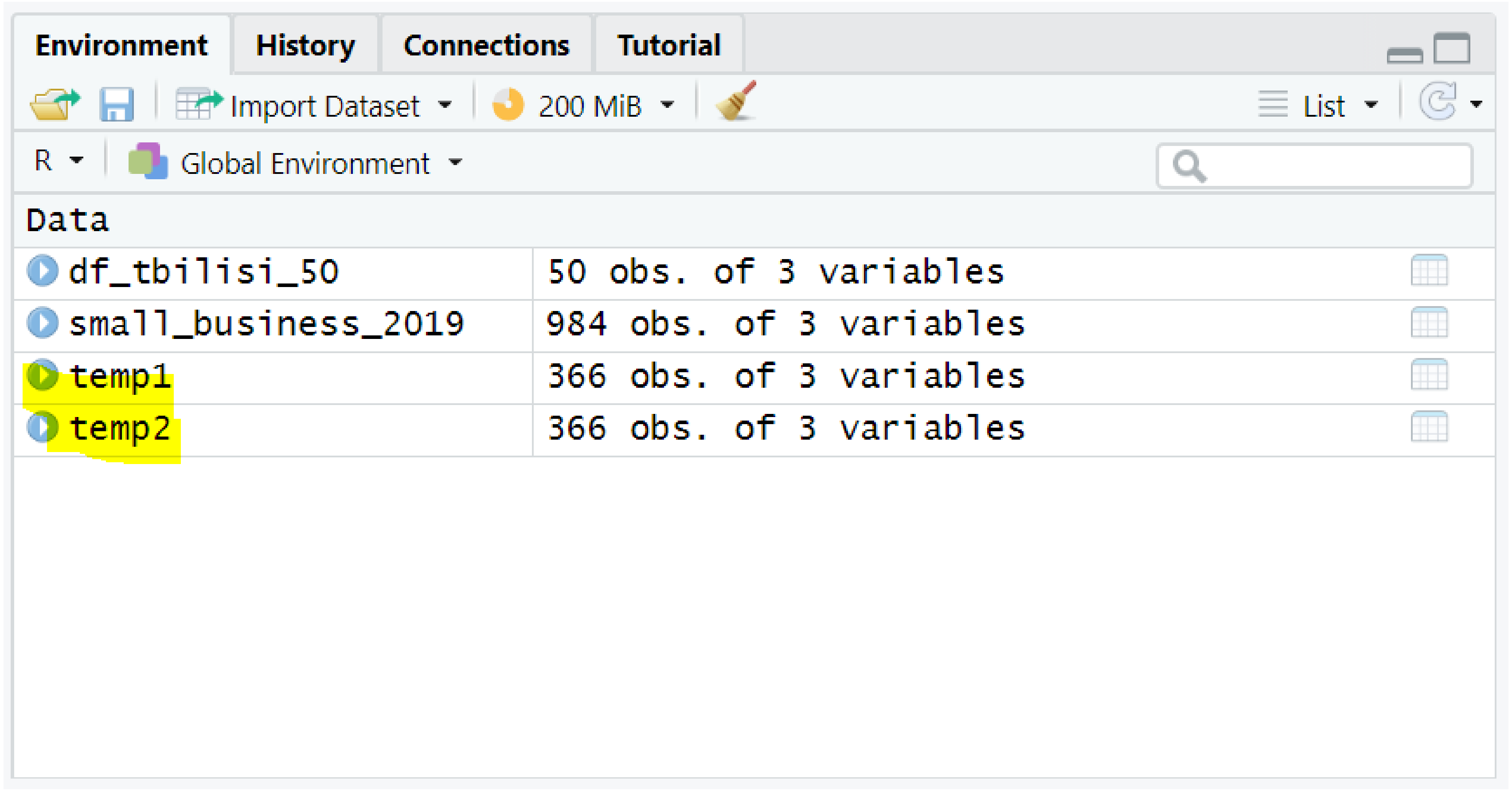
Piping

- Before we start producing more refined outputs, we need to cover piping
- You probably remember this piece of code from one of yesterday's exercise:

```r
# Filter only businesses in Tbilisi:
temp1 <- filter(small_business_2019, region == "Tbilisi")

# Sort previous result by income, descending order:
temp2 <- arrange(temp1, -income)

# Keep only the 50 first businesses after sorting:
df_tbilisi_50 <- filter(temp2, row_number() <= 50)
```

Piping

This code works, but the problem with it is that it makes us generate unnecessary intermediate dataframes (`temp1`, `temp2`) that store results temporarily



```
Data
df_tbilisi_50          50 obs. of 3 variables
small_business_2019    984 obs. of 3 variables
temp1                  366 obs. of 3 variables
temp2                  366 obs. of 3 variables
```

Piping

Instead, we can use pipes to **pass the results of a function and apply a new function on top of it**

```r
# Filter only businesses in Tbilisi:
temp1 <- filter(small_business_2019,
                region == "Tbilisi")

# Sort previous result by income, descending order:
temp2 <- arrange(temp1,
                 -income)

# Keep only the 50 first businesses after sorting:
df_tbilisi_50 <- filter(temp2,
                        row_number() <= 50)
```

```r
# The same but with pipes:
df_tbilisi_50 <- filter(small_business_2019,
                        region == "Tbilisi") %>%
  arrange(-income) %>%
  filter(row_number() <= 50)
```

Piping

```r
# Filter only businesses in Tbilisi:
temp1 <- filter(small_business_2019,
                region == "Tbilisi")

# Sort previous result by income, descending order:
temp2 <- arrange(temp1,
                 -income)

# Keep only the 50 first businesses after sorting:
df_tbilisi_50 <- filter(temp2,
                        row_number() <= 50)
```

```r
# The same but with pipes:
df_tbilisi_50 <- filter(small_business_2019,
                        region == "Tbilisi") %>%
  arrange(-income) %>%
  filter(row_number() <= 50)
```

There are several important details to notice here:

1.- The resulting dataframe `df_tbilisi_50` is **the same in both cases**

Piping

```r
# Filter only businesses in Tbilisi:
temp1 <- filter(small_business_2019,
                region == "Tbilisi")

# Sort previous result by income, descending order:
temp2 <- arrange(temp1,
                 -income)

# Keep only the 50 first businesses after sorting:
df_tbilisi_50 <- filter(temp2,
                        row_number() <= 50)
```

```r
# The same but with pipes:
df_tbilisi_50 <- filter(small_business_2019,
                        region == "Tbilisi") %>%
  arrange(-income) %>%
  filter(row_number() <= 50)
```

2.- The name of the resulting dataframe is now defined in the first line of this data wrangling operation. This is because **R evaluates lines with consecutive pipes as if they were a single line**

Piping

```r
# Filter only businesses in Tbilisi:
temp1 <- filter(small_business_2019,
                region == "Tbilisi")

# Sort previous result by income, descending order:
temp2 <- arrange(temp1,
                 -income)

# Keep only the 50 first businesses after sorting:
df_tbilisi_50 <- filter(temp2,
                        row_number() <= 50)
```

```r
# The same but with pipes:
df_tbilisi_50 <- filter(small_business_2019,
                        region == "Tbilisi") %>%
  arrange(-income) %>%
  filter(row_number() <= 50)
```

3.- Notice that the functions `arrange()` and `filter()` used after the pipes now have only **one argument instead of two**. This is because when using pipes the first argument is implied to be result of the function before the pipes

Piping

Exercise 2: filtering and sorting revisited

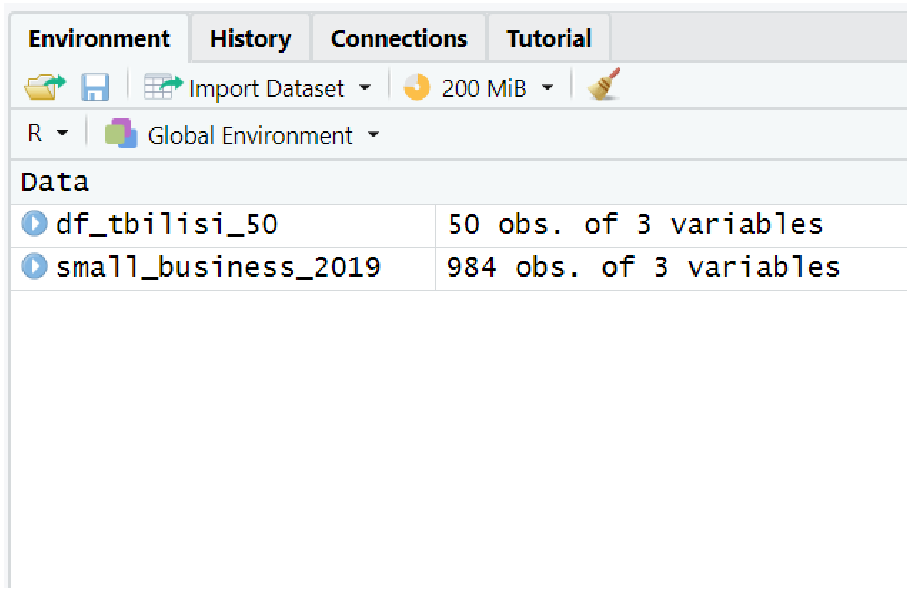
1. Apply the same filtering and sorting now with pipes

```r
df_tbilisi_50 <- filter(small_business_2019,
                        region == "Tbilisi") %>%
  arrange(-income) %>%
  filter(row_number() <= 50)
```

Piping

Now we will not have any annoying intermediate results stored in our environment!

```r
# The same but with pipes:
df_tbilisi_50 <- filter(small_business_2019,
                        region == "Tbilisi") %>%
  arrange(-income) %>%
  filter(row_number() <= 50)
```



```
Data
df_tbilisi_50          50 obs. of 3 variables
small_business_2019    984 obs. of 3 variables
```

Piping

Lastly, we can also add more formatting to this code to improve its clarity even more:

```r
# Previous solution
df_tbilisi_50 <- filter(small_business_2019,
                        region == "Tbilisi") %>%
  arrange(-income) %>%
  filter(row_number() <= 50)
```

```r
# The same with better spacing
df_tbilisi_50 <-
  small_business_2019 %>%
  filter(region == "Tbilisi") %>%
  arrange(-income) %>%
  filter(row_number() <= 50)
```

Piping

```r
# Previous solution
df_tbilisi_50 <- filter(small_business_2019,
                        region == "Tbilisi") %>%
  arrange(-income) %>%
  filter(row_number() <= 50)
```

```r
# The same with better spacing
df_tbilisi_50 <-
  small_business_2019 %>%
  filter(region == "Tbilisi") %>%
  arrange(-income) %>%
  filter(row_number() <= 50)
```

- Good code is code that is both correct (does what it's supposed to) and it's easy to understand
- Piping is **instrumental for writing good code in R**

Piping

Always use pipes!

Now that you now about the power of the pipes, use them wisely!

- Remember that pipes are part of the library `dplyr`, you need to load it before using them
- Pipes also improve code clarity drastically
- Many R coders use pipes and internet examples assume you know them
- **We'll use pipes now in the next examples and exercises of the rest of this training**

Quick summary statistics // სწრაფი შემაჯამებელი სტატისტიკა

Quick summary statistics

We learned yesterday how to produce dataframes with results and export them.

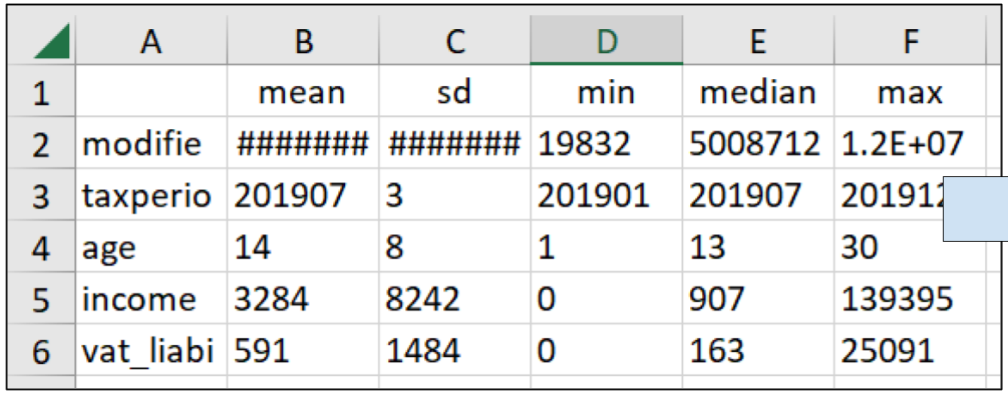
But what if you want to ... ?

- ...export results in a different format (example: Excel)
- ...further customize which rows and columns to display in a result
- ...format the results you export

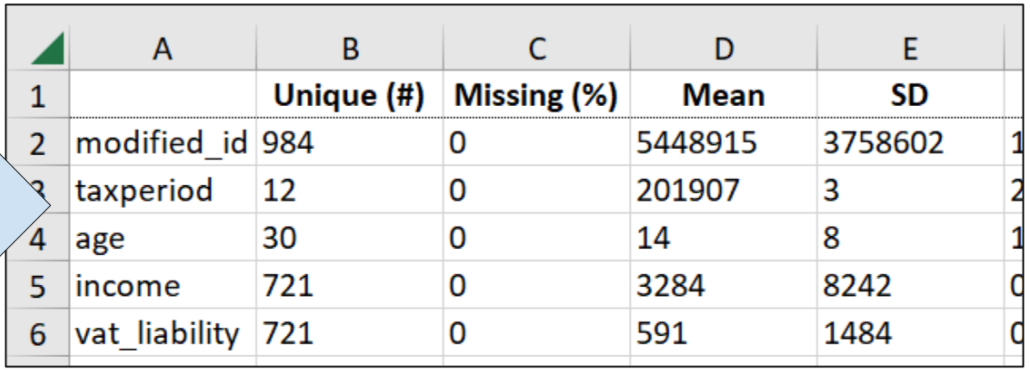
Quick summary statistics

You will need `modelsummary` and `huxtable` for this

- These libraries allow you to export results in a customized way
- We chose a combination of both because together they export a large range of output types and allow fine-grained customization of outputs



	A	B	C	D	E	F
1		mean	sd	min	median	max
2	modifie	#######	#######	19832	5008712	1.2E+07
3	taxperio	201907	3	201901	201907	20191
4	age	14	8	1	13	30
5	income	3284	8242	0	907	139395
6	vat_liabi	591	1484	0	163	25091



	A	B	C	D	E
1		**Unique (#)**	**Missing (%)**	**Mean**	**SD**
2	modified_id	984	0	5448915	3758602
3	taxperiod	12	0	201907	3
4	age	30	0	14	8
5	income	721	0	3284	8242
6	vat_liability	721	0	591	1484

Quick summary statistics

We'll start by introducing the function `datasummary_skim()` from `modelsummary`

`datasummary_skim(data, output, ...)`

- **data:** the data set to be summarized, the only required argument
- **output:** the type of output desired
- **... :** additional options allow for formatting customization, such as including notes and titles

For example:

```
datasummary_skim(
  data,
  output = "default",
  type = "numeric",
  title = NULL,
  notes = NULL,
  ...
)
```

Quick summary statistics

Exercise 3: Calculate quick summary statistics

1. Load `modelsummary` with `library(modelsummary)`
2. Use `datasummary_skim()` to create a descriptive statistics table for `small_business_all`

```
datasummary_skim(small_business_2019_all)
```

Quick summary statistics

You should be seeing this result in the lower right panel of RStudio.

	Unique (#)	Missing (%)	Mean	SD	Min	Median	Max	
modified_id	984	0	5448915.1	3758602.4	19832.0	5008712.0	12296912.0	
taxperiod	12	0	201906.7	3.4	201901.0	201907.0	201912.0	
age	30	0	14.0	8.4	1.0	13.0	30.0	
income	721	0	3283.9	8242.4	0.0	906.8	139394.5	
vat_liability	721	0	591.1	1483.6	0.0	163.2	25091.0	

Quick summary statistics

- Most functions of `modelsummary` summarize only numeric variables by default
- To summarize categorical variables, use the argument `type = "categorical"`

```r
datasummary_skim(small_business_2019_all, type = "categorical")
```

Quick summary statistics

		N	%
region	Guria	259	25.9
	ImereTI-Racha-Lechkhum-kv.SvaneTi	37	3.7
	KaxeTi	270	27.0
	Kvemo KarTli	9	0.9
	Samegrelo-Z.SvaneTi	28	2.8
	Samtskhe-Javakheti	7	0.7
	Shida KarTli	17	1.7
	Tbilisi	373	37.3
group	No notifications to be sent	286	28.6
	Notification sent on Day 13 and Day 15	226	22.6
	Notification sent only on Day 13	247	24.7
	Notification sent only on Day 15	241	24.1

Quick summary statistics

- `datasummary_skim()` is convenient because it's fast, easy, and shows a lot of information

	Unique (#)	Missing (%)	Mean	SD	Min	Median	Max	
modified_id	984	0	5448915.1	3758602.4	19832.0	5008712.0	12296912.0	
taxperiod	12	0	201906.7	3.4	201901.0	201907.0	201912.0	
age	30	0	14.0	8.4	1.0	13.0	30.0	
income	721	0	3283.9	8242.4	0.0	906.8	139394.5	
vat_liability	721	0	591.1	1483.6	0.0	163.2	25091.0	

- But what if we wanted to customize what to show? that's when we use `datasummary()` instead, also from the library `modelsummary`

Customized summary statistics // მორგებული შემაჯამებელი სტატისტიკა

Customized summary statistics

`datasummary()` is very similar to `data_summary_skim()`. The only difference is that it requires a **formula argument**.

`datasummary(formula, data, output)`

- **formula:** a two-sided formula to describe the table as: rows ~ columns
- **data:** the data set to be summarized
- **output:** the type of output desired
- **... :** additional options allow for formatting customization

```
datasummary(
  var1 + var2 + var3 ~ stat1 + stat2 + stat3 + stat4,
  data = data
)
```

Customized summary statistics

Exercise 4:

Create a summary statistics table showing the nuber of observations, mean, standard deviation, minimum, and maximum for variables `age`, `income`, and `vat_liability` of the dataframe `small_business_2019_all`

1. Use `datasummary()` for this:

```r
datasummary(
  age + income + vat_liability ~ N + Mean + SD + Min + Max,
  small_business_2019_all
)
```

Customized summary statistics

	N	Mean	SD	Min	Max
age	1000	14.00	8.37	1.00	30.00
income	1000	3283.87	8242.45	0.00	139394.52
vat_liability	1000	591.10	1483.64	0.00	25091.01

Customized summary statistics

```r
datasummary(
  age + income + vat_liability ~ N + Mean + SD + Min + Max, # this is the formula
  small_business_2019_all                                 # this is the data
)
```

Some notes:

- The arguments **formula** and **data** are mandatory for `datasummary()`
- All other arguments are optional (like `title = *some-title*`, to add a table title)
- The formula should always be defined as: rows ~ columns
- The rows and columns in the formula are separated by a plus (`+`) sign

Customized summary statistics

```r
datasummary(
  age + income + vat_liability ~ N + Mean + SD + Min + Max, # this is the formula
  small_business_2019_all                                   # this is the data
)
```

In this exercise we used the statistics N (number of observations), mean, SD (standard deviation), Min (minimum), and Max (maximum). Other statistics you can include are:

Statistic	Keyword
Median	`Median`
25th percentile	`P25`
75th percentile	`P75`
In general: percentile XX	`PXX`
Small histogram	`Histogram`

Exporting tables // მაგიდების ექსპორტი

Exporting tables // ცხრილების ექსპორტი

Remember that both `datasummary_skim()` and `datasummary()` have an optional argument named *output*? We can use it to specify a file path for an output file.

For example:

```
datasummary_skim(small_business_2019_all,
                 output = "quick_stats.docx")
```

Will export the result to the `Documents` folder (in Windows) in a Word file named `quick_stats.docx`

Exporting tables // მაგიდების ექსპორტი

The file type of the output is dictated by the file extension. For example:

File name	File extension	Output type
"quick_stats.docx"	`.docx`	Word
"quick_stats.pptx"	`.pptx`	Power Point
"quick_stats.html"	`.html`	HTML (to open in a web explorer)
"quick_stats.tex"	`.tex`	Latex
"quick_stats.md"	`.md`	Markdown

Noticed that we're missing Excel?

Exporting tables // მაგიდების ექსპორტი

That's because the functions of `modelsummary` can't export to Excel

- Nonetheless, we can use the library `huxtable` as an intermediary to transform results from `modelsummary` functions to Excel files

- `huxtable` is a package for exporting tables in general that allows you to **customize the output you're exporting**

- We'll know how to use it in the next exercise

Exporting tables // მაგიდების ექსპორტი

Exercise 5: Export a table to Excel

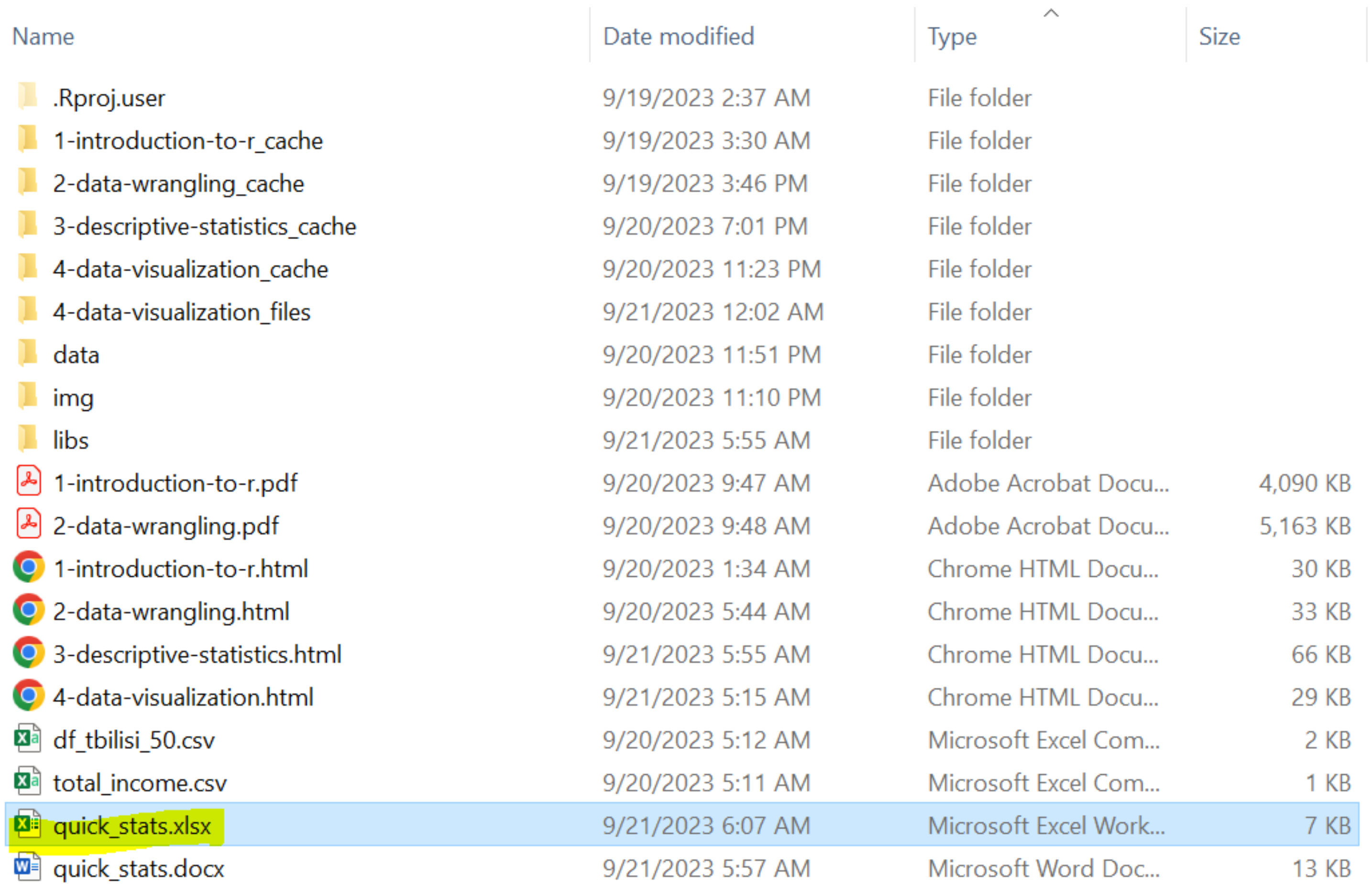
1. Load `huxtable` with `library(huxtable)`

2. Run the following code to export the result of `datasummary_skim()` to Excel:

```r
# Store the table in a new object
stats_table <- datasummary_skim(small_business_2019_all, output = "huxtable")

# Export this new object to Excel with quick_xlsx()
quick_xlsx(stats_table, file = "quick_stats.xlsx")
```

Exporting tables // მაგიდების ექსპორტი

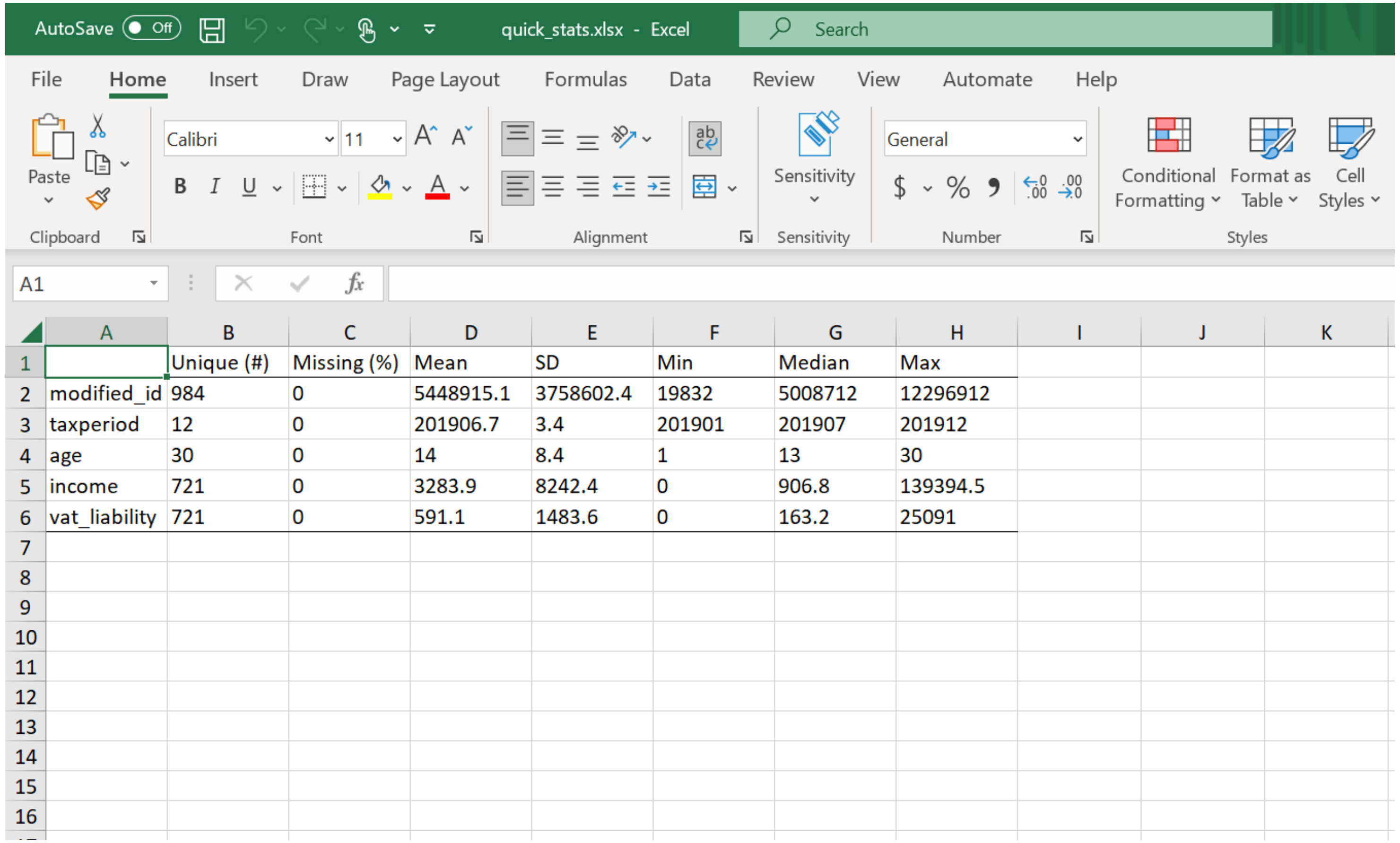
Now the result will show in your `Documents` folder



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	
2-data-wrangling_cache	9/19/2023 3:46 PM	File folder	
3-descriptive-statistics_cache	9/20/2023 7:01 PM	File folder	
4-data-visualization_cache	9/20/2023 11:23 PM	File folder	
4-data-visualization_files	9/21/2023 12:02 AM	File folder	
data	9/20/2023 11:51 PM	File folder	
img	9/20/2023 11:10 PM	File folder	
libs	9/21/2023 5:55 AM	File folder	
1-introduction-to-r.pdf	9/20/2023 9:47 AM	Adobe Acrobat Docu...	4,090 KB
2-data-wrangling.pdf	9/20/2023 9:48 AM	Adobe Acrobat Docu...	5,163 KB
1-introduction-to-r.html	9/20/2023 1:34 AM	Chrome HTML Docu...	30 KB
2-data-wrangling.html	9/20/2023 5:44 AM	Chrome HTML Docu...	33 KB
3-descriptive-statistics.html	9/21/2023 5:55 AM	Chrome HTML Docu...	66 KB
4-data-visualization.html	9/21/2023 5:15 AM	Chrome HTML Docu...	29 KB
df_tbilisi_50.csv	9/20/2023 5:12 AM	Microsoft Excel Com...	2 KB
total_income.csv	9/20/2023 5:11 AM	Microsoft Excel Com...	1 KB
quick_stats.xlsx	9/21/2023 6:07 AM	Microsoft Excel Work...	7 KB
quick_stats.docx	9/21/2023 5:57 AM	Microsoft Word Doc...	13 KB

Exporting tables // მაგიდების ექსპორტი

And you can open it with Excel for further customization if you want



	Unique (#)	Missing (%)	Mean	SD	Min	Median	Max
modified_id	984	0	5448915.1	3758602.4	19832	5008712	12296912
taxperiod	12	0	201906.7	3.4	201901	201907	201912
age	30	0	14	8.4	1	13	30
income	721	0	3283.9	8242.4	0	906.8	139394.5
vat_liability	721	0	591.1	1483.6	0	163.2	25091

Exporting tables // მაგიდების ექსპორტი

```r
# Store the table in a new object
stats_table <- datasummary_skim(small_business_2019_all, output = "huxtable")

# Export this new object to Excel with quick_xlsx()
quick_xlsx(stats_table, file = "quick_stats.xlsx")
```

Some comments about this code:

- `quick_xlsx()` is a function from `huxtable`. The first argument is the object we export and the second is the file name. We could also use a file path here
- Note that we now use the argument `output = "huxtable"` in `datasummary_skim()`. This tells R that the output should be an object type that we can operate later with `huxtable` functions, such as `quick_xlsx()`

Customizing table outputs // ცხრილის შედეგების მორგება

Customizing table outputs

The code below shows how the table `stats_table` can be formatted:

```r
# We start with stats_table:
stats_table %>%
  # Use first row as table header
  set_header_rows(1, TRUE) %>%
  # Use first column as row header
  set_header_cols(1, TRUE)  %>%
  # Don't round large numbers
  set_number_format(everywhere, 2:ncol(.), "%9.0f") %>
  # Center cells in first row
  set_align(1, everywhere, "center") %>%
  # Set a theme for quick formatting
  theme_basic()
```

	Unique (#)	Missing (%)	Mean	SD	Min	Median	Max
modified_id	984	0	5448915	3758602	19832	5008712	12296912
taxperiod	12	0	201907	3	201901	201907	201912
age	30	0	14	8	1	13	30
income	721	0	3284	8242	0	907	139395
vat_liability	721	0	591	1484	0	163	25091

Customizing table outputs

Exercise 6: Export a customized table to Excel

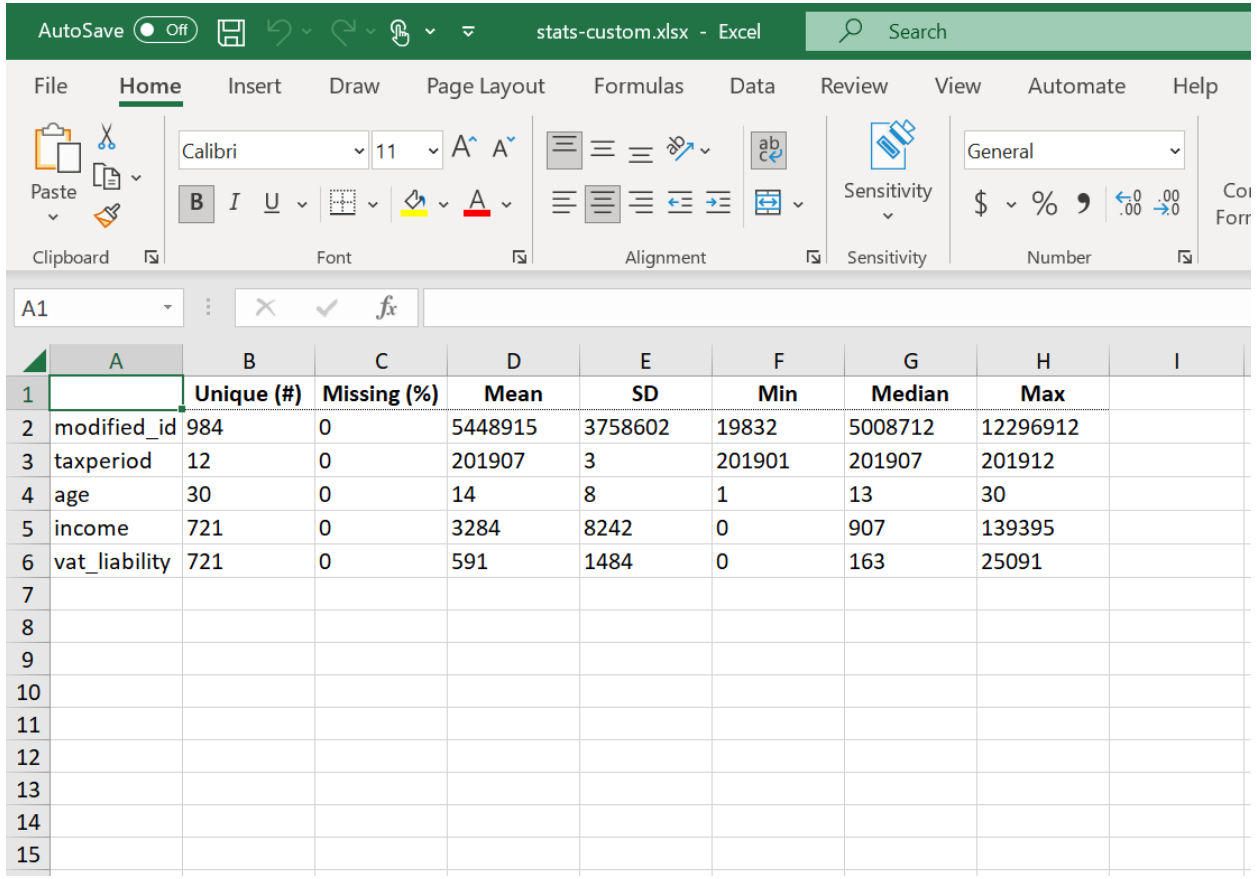
1.- Customize `stats_table` in a new object called `stats_table_custom`

```
stats_table_custom <- stats_table %>%
  # Use first row as table header
  set_header_rows(1, TRUE) %>%
  # Use first column as row header
  set_header_cols(1, TRUE)  %>%
  # Don't round large numbers
  set_number_format(everywhere, 2:ncol(.), "%9.0f") %>
  # Center cells in first row
  set_align(1, everywhere, "center") %>%
  # Set a theme for quick formatting
  theme_basic()
```

2.- Export `stats_table_custom` to a file named `stats-custom.xlsx` with `quick_xlsx()`

```
quick_xlsx(
  stats_table_custom,
  file = "stats-custom.xlsx"
  )
```

Customizing table outputs


AutoSave Off
stats-custom.xlsx - Excel
Search
File Home Insert Draw Page Layout Formulas Data Review View Automate Help
Paste
Calibri 11
General
Sensitivity
Clipboard Font Alignment Sensitivity Number
A1

	Unique (#)	Missing (%)	Mean	SD	Min	Median	Max
modified_id	984	0	5448915	3758602	19832	5008712	12296912
taxperiod	12	0	201907	3	201901	201907	201912
age	30	0	14	8	1	13	30
income	721	0	3284	8242	0	907	139395
vat_liability	721	0	591	1484	0	163	25091

Customizing table outputs

Notice that here in the first part of the exercise we stored the result in a new object

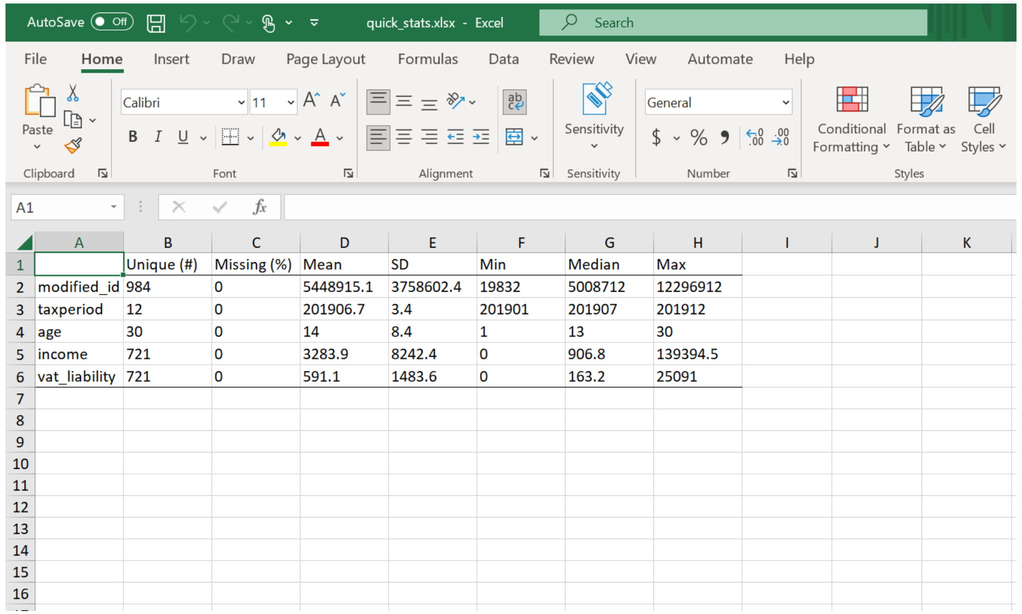
```
stats_table_custom <- stats_table %>%  # <---- here
  set_header_rows(1, TRUE) %>%
  set_header_cols(1, TRUE)  %>%
  set_number_format(everywhere, 2:ncol(.), "%9.0f") %>%
  set_align(1, everywhere, "center") %>%
  theme_basic()
```

This is the object that we export later with `quick_xslx()`

```
quick_xlsx(
  stats_table_custom,
  file = "stats-custom.xlsx"
  )
```

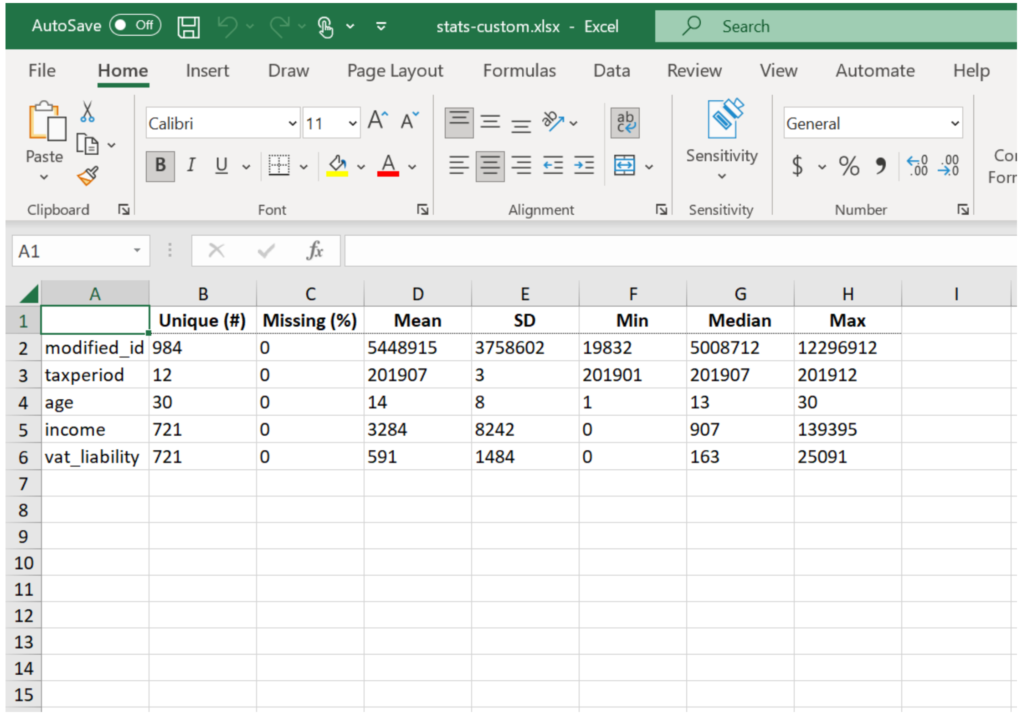
Customizing table outputs

Before:



	Unique (#)	Missing (%)	Mean	SD	Min	Median	Max
modified_id	984	0	5448915.1	3758602.4	19832	5008712	12296912
taxperiod	12	0	201906.7	3.4	201901	201907	201912
age	30	0	14	8.4	1	13	30
income	721	0	3283.9	8242.4	0	906.8	139394.5
vat_liability	721	0	591.1	1483.6	0	163.2	25091

After:

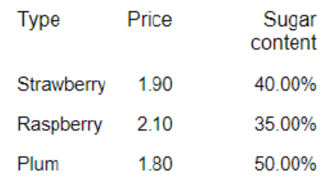


	Unique (#)	**Missing (%)**	**Mean**	**SD**	**Min**	**Median**	**Max**
modified_id	984	0	5448915	3758602	19832	5008712	12296912
taxperiod	12	0	201907	3	201901	201907	201912
age	30	0	14	8	1	13	30
income	721	0	3284	8242	0	907	139395
vat_liability	721	0	591	1484	0	163	25091

Customizing table outputs

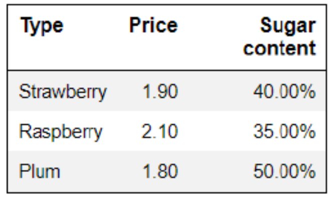
We used `theme_basic()` to give a minimalistic, basic theme to the table. Other available themes are:

jams



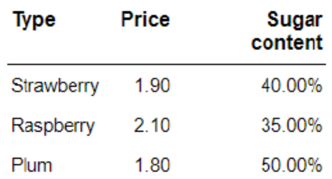
Type	Price	Sugar content
Strawberry	1.90	40.00%
Raspberry	2.10	35.00%
Plum	1.80	50.00%

theme_plain



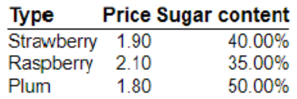
Type	Price	Sugar content
Strawberry	1.90	40.00%
Raspberry	2.10	35.00%
Plum	1.80	50.00%

theme_basic



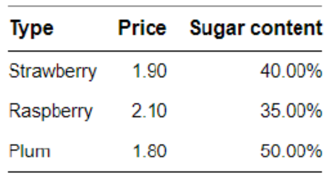
Type	Price	Sugar content
Strawberry	1.90	40.00%
Raspberry	2.10	35.00%
Plum	1.80	50.00%

theme_compact



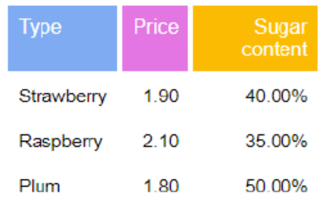
Type	Price	Sugar content
Strawberry	1.90	40.00%
Raspberry	2.10	35.00%
Plum	1.80	50.00%

theme_article



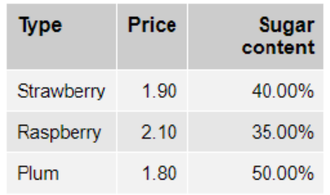
Type	Price	Sugar content
Strawberry	1.90	40.00%
Raspberry	2.10	35.00%
Plum	1.80	50.00%

theme_bright



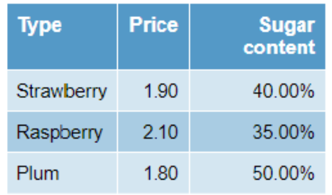
Type	Price	Sugar content
Strawberry	1.90	40.00%
Raspberry	2.10	35.00%
Plum	1.80	50.00%

theme_grey



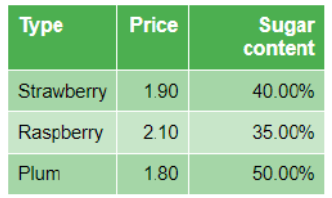
Type	Price	Sugar content
Strawberry	1.90	40.00%
Raspberry	2.10	35.00%
Plum	1.80	50.00%

theme_blue



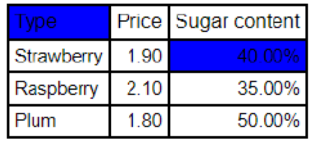
Type	Price	Sugar content
Strawberry	1.90	40.00%
Raspberry	2.10	35.00%
Plum	1.80	50.00%

theme_green



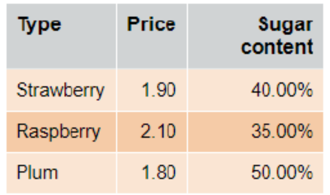
Type	Price	Sugar content
Strawberry	1.90	40.00%
Raspberry	2.10	35.00%
Plum	1.80	50.00%

theme_mondrian



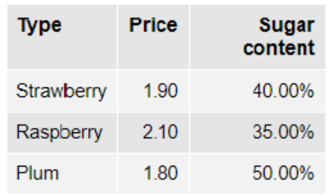
Type	Price	Sugar content
Strawberry	1.90	40.00%
Raspberry	2.10	35.00%
Plum	1.80	50.00%

theme_orange



Type	Price	Sugar content
Strawberry	1.90	40.00%
Raspberry	2.10	35.00%
Plum	1.80	50.00%

theme_striped



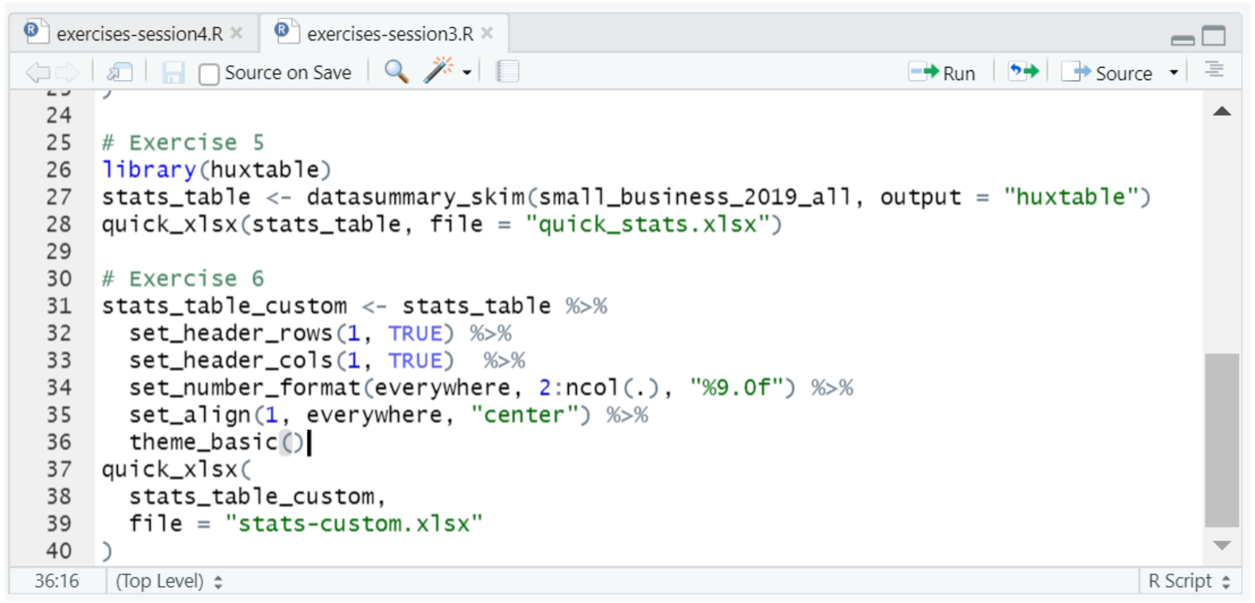
Type	Price	Sugar content
Strawberry	1.90	40.00%
Raspberry	2.10	35.00%
Plum	1.80	50.00%

Wrapping up // შეჯამება

Wrapping up // შეჯამება

Save your work!

Click the floppy disk to save the script you wrote in this session.



```r

# Exercise 5
library(huxtable)
stats_table <- datasummary_skim(small_business_2019_all, output = "huxtable")
quick_xlsx(stats_table, file = "quick_stats.xlsx")

# Exercise 6
stats_table_custom <- stats_table %>%
  set_header_rows(1, TRUE) %>%
  set_header_cols(1, TRUE)  %>%
  set_number_format(everywhere, 2:ncol(.), "%9.0f") %>%
  set_align(1, everywhere, "center") %>%
  theme_basic()
quick_xlsx(
  stats_table_custom,
  file = "stats-custom.xlsx"
)
```

Wrapping up // შეჯამება

What else is available?

- This was a short overview of how `modelsummary` and `huxtable` work together to produce professional-looking table outputs in R
- Other formatting options are: (all from `huxtable`)

Formatting	Command
Export in new Excel tabs instead of new files	`as_Workbook()`
Change row names	`add_rownames()`
Change column names	`add_colnames()`
Cells in bold	`set_bold()`
Cells in italics	`set_italic()`
Cell font size	`font_size()`
Cell color	`background_color()`

Wrapping up // შეჯამება

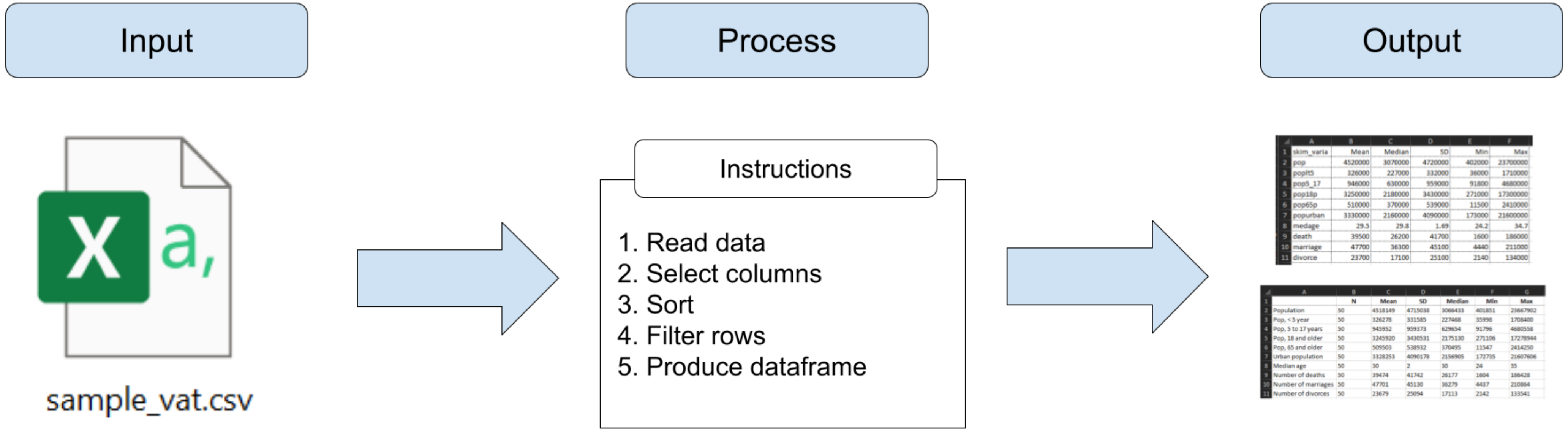
What else is available?

More of this is explained in the libraries documentation:

- `modelsummary` documentation: https://modelsummary.com/index.html
- `huxtable` documentation: https://hughjonesd.github.io/huxtable/

Wrapping up // შეჯამება

This session



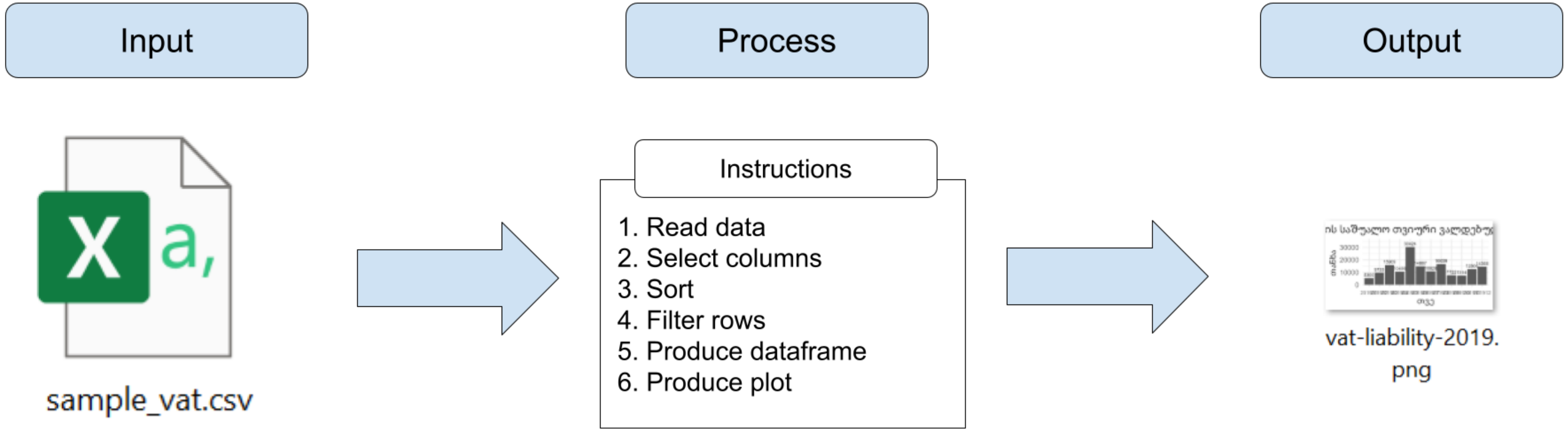
1. Read data
2. Select columns
3. Sort
4. Filter rows
5. Produce dataframe

	A	B	C	D	E	F
1	skim_varia	Mean	Median	SD	Min	Max
2	pop	4520000	3070000	4720000	402000	23700000
3	poplt5	326000	227000	332000	36000	1710000
4	pop5_17	946000	630000	959000	91800	4680000
5	pop18p	3250000	2180000	3430000	271000	17300000
6	pop65p	510000	370000	539000	11500	2410000
7	popurban	3330000	2160000	4090000	173000	21600000
8	medage	29.5	29.8	1.69	24.2	34.7
9	death	39500	26200	41700	1600	186000
10	marriage	47700	36300	45100	4440	211000
11	divorce	23700	17100	25100	2140	134000

	A	B	C	D	E	F	G
1		N	Mean	SD	Median	Min	Max
2	Population	50	4518149	4715038	3066433	401851	23667902
3	Pop, < 5 year	50	326278	331585	227468	35998	1708400
4	Pop, 5 to 17 years	50	945952	959373	629654	91796	4680558
5	Pop, 18 and older	50	3245920	3430531	2175130	271106	17278944
6	Pop, 65 and older	50	509503	538932	370495	11547	2414250
7	Urban population	50	3328253	4090178	2156905	172735	21607606
8	Median age	50	30	2	30	24	35
9	Number of deaths	50	39474	41742	26177	1604	186428
10	Number of marriages	50	47701	45130	36279	4437	210864
11	Number of divorces	50	23679	25094	17113	2142	133541

Wrapping up // შეჯამება

Next session (last one)



1. Read data
2. Select columns
3. Sort
4. Filter rows
5. Produce dataframe
6. Produce plot

Thanks! // მადლობა! // ¡Gracias! // Obrigado!