Session 3 - Descriptive Statistics

R Training

María Reyes Retana
The World Bank | January 2025

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:

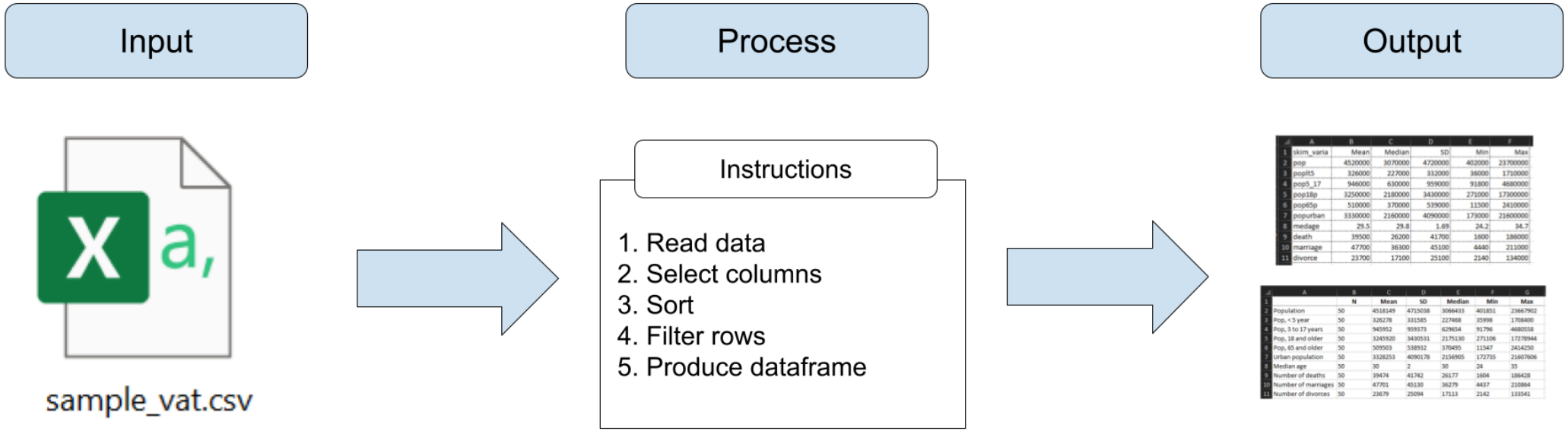
Strengthening Public Sector Reporting and Data Analysis

January 13 –17, 2024

Introduction

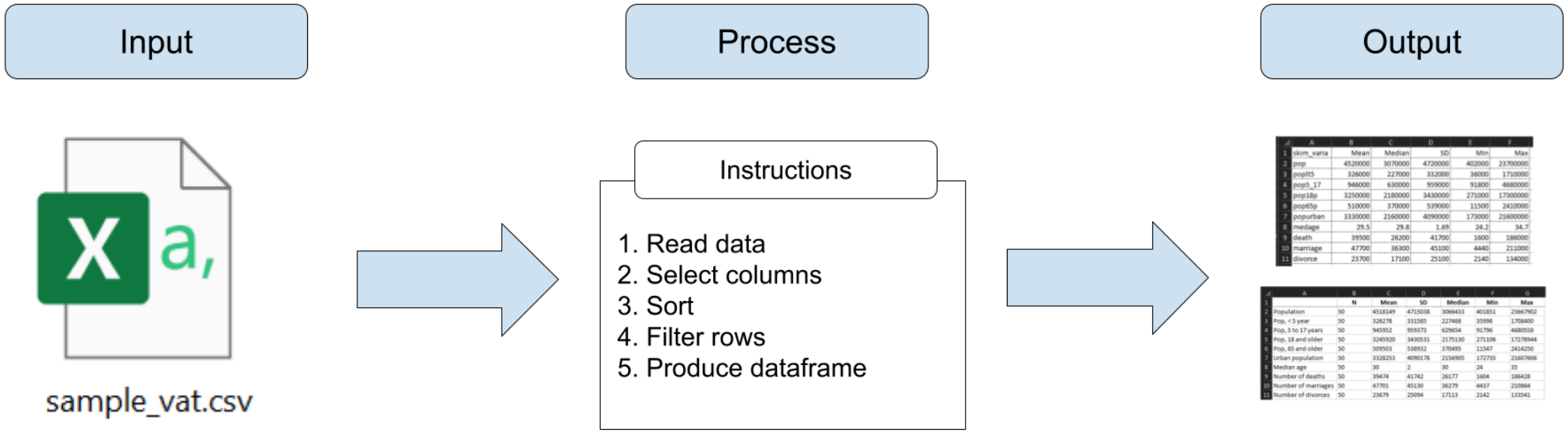
Introduction

- 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



Introduction

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



	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

		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

Relevance to your work

In your **annual reports**, you often include:

- Summary tables like **revenue by year**, **spending across categories**, or **staff counts by department**.
- These tables help summarize key trends and patterns for decision-makers.

Table 8: Programmed PSRL for 2023 (In GH₵)

Month	Diesel	LPG	Petrol	MGO-F	Gas Oil	Gas Oil	Unified/ Napththa	Total
Jan-23	23,872,267.72	3,682,560.43	31,661,429.70	55,898.08	5,237,078.13	753,524.51	18,461.87	65,281,220.45
Feb-23	24,309,842.16	3,182,459.63	28,976,008.73	50,014.07	4,427,220.69	674,206.14	24,212.29	61,643,963.71
Mar-23	24,096,033.18	3,818,951.55	28,386,586.25	48,248.87	4,211,258.70	650,410.63	24,212.29	61,235,701.47
Apr-23	26,358,863.90	3,409,778.17	31,439,960.64	46,483.67	4,049,287.21	626,615.12	30,265.37	65,961,254.09
May-23	24,599,710.89	3,500,705.59	31,281,630.87	44,130.07	4,103,277.71	594,887.77	24,212.29	64,148,555.19
Jun-23	27,312,455.37	3,909,878.97	31,923,406.89	43,541.66	3,995,296.72	586,955.93	24,212.29	67,795,747.84
Jul-23	27,123,060.55	4,091,733.81	32,816,321.77	42,364.86	3,887,315.73	571,092.26	24,212.29	68,556,101.27
Aug-23	25,959,652.42	4,000,806.39	31,640,924.49	41,188.06	4,265,249.20	555,228.58	24,212.29	66,487,261.44
Sep-23	21,590,679.66	4,000,806.39	33,402,163.23	52,367.68	4,589,192.18	705,933.48	27,238.83	64,368,381.44
Oct-23	27,235,725.40	3,909,878.97	27,364,475.58	52,956.08	4,697,173.17	713,865.32	27,238.83	64,001,313.35
Nov-23	28,242,906.95	3,909,878.97	36,501,774.26	54,132.88	5,129,097.14	729,728.99	26,936.18	74,594,455.37
Dec-23	27,965,119.88	4,046,270.10	39,826,295.00	57,074.88	5,399,049.62	769,388.18	27,238.83	78,090,436.49
Total	308,666,318.08	45,463,708.96	385,220,977.42	588,400.87	53,990,496.19	7,931,836.90	302,653.68	802,164,392.11

Source: NPA

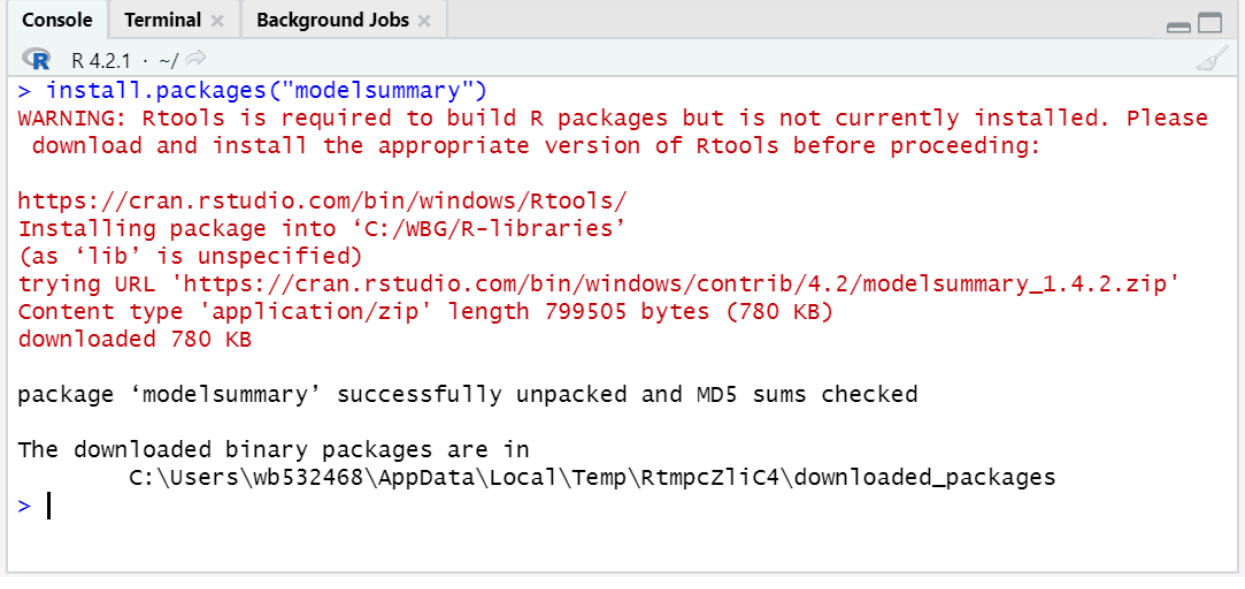
- Today, we will practice **creating similar summary tables** using mock data.
- While we're using a simple dataset today, the same steps can be applied to your own data for reports.

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

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

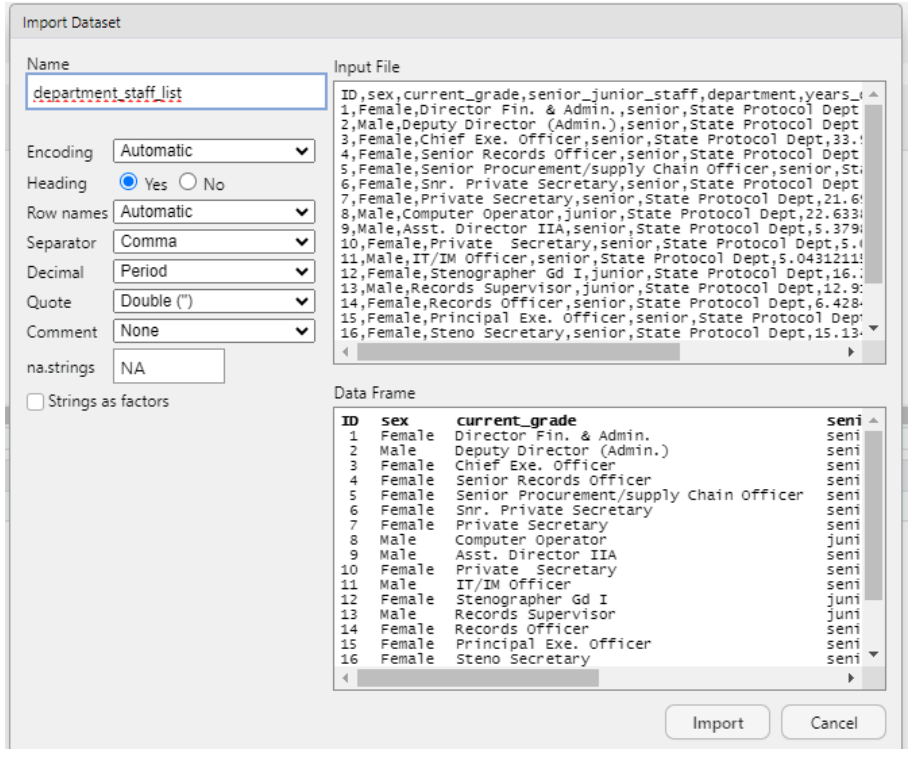
1. Install `modelsummary` and `huxtable`:

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



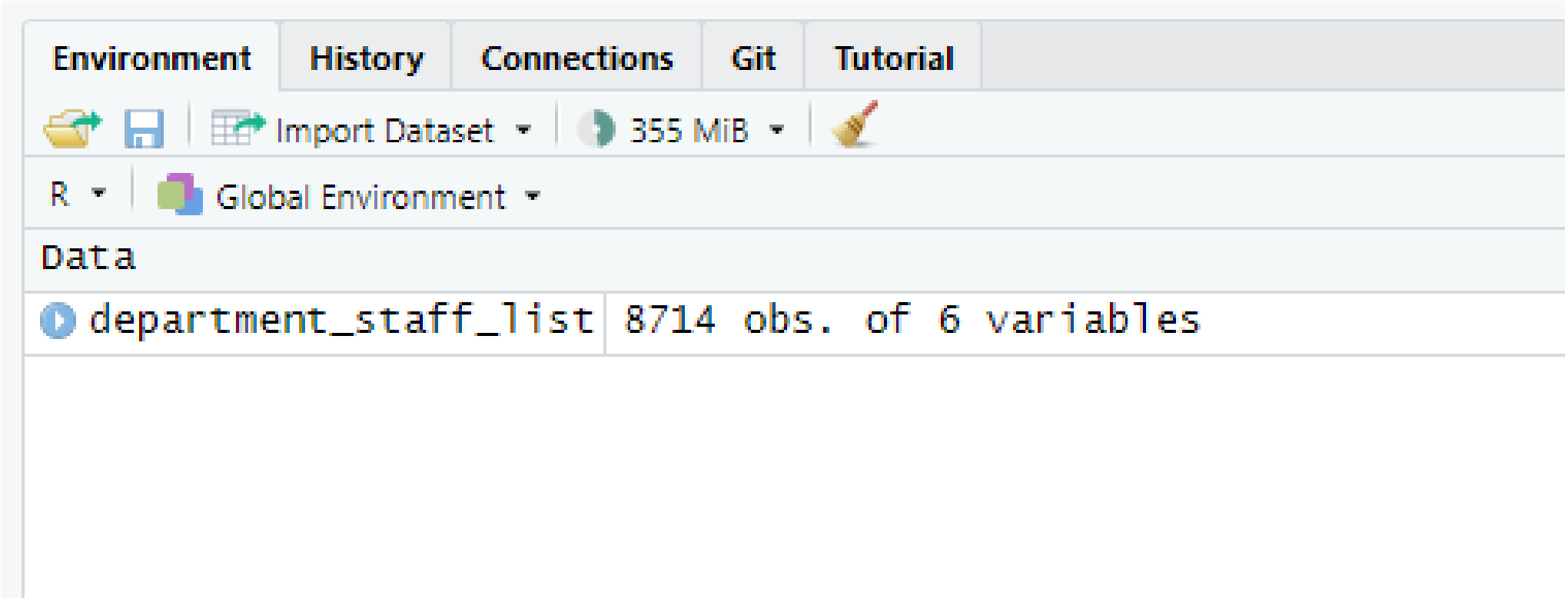
Exercise 1b: Download and load the data we'll use

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 `department_staff_final.csv`
   - If you don't know where the file is, remember to check in your `Downloads` folder
3. Select `Import`



Introduction

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



Recap: always know your data!

- This is the data that we used yesterday!

```
glimpse(department_staff)
```

```
## Rows: 8,645
## Columns: 6
## $ 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…
## $ age                <dbl> 60.15058, 41.74949, 59.43053, 50.55989, 51.28268, …
```

Piping

The pipe (%>%) operator

- Before diving into the contents of today's session, 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

As we saw yesterday the functions are normally organized around a set of verbs, or actions to be taken.

The pipe

As we saw yesterday the functions are normally 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

As we saw yesterday the functions are normally 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.

The pipe

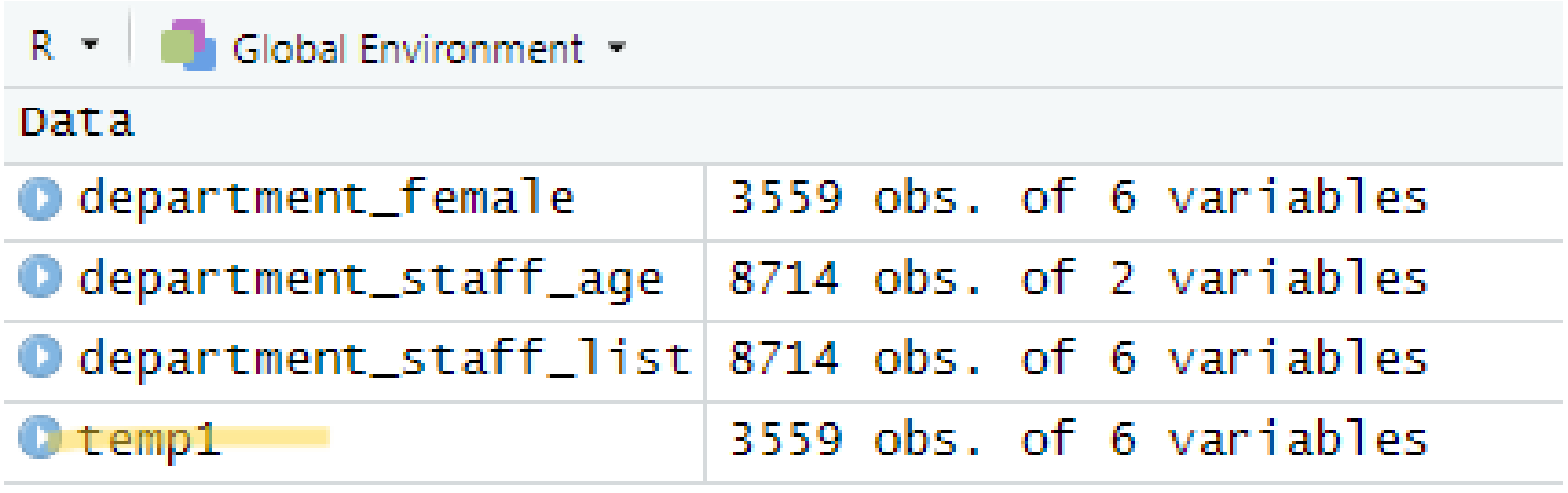
- You probably remember this piece of code from one of yesterday's exercise:

```
# Filter only female employees:
temp1 <-  filter(department_staff_list, sex == "Female") # filter by female

# Sort previous result by years of service
department_female <- arrange(temp1, years_of_service) # order by years of service
```

The pipe

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



The pipe

Instead, we can use pipes to **pass the results of a function and apply a new function on top of it** (just like Mer's waking up sequence)

```
# Filter only female employees:
temp1 <-  filter(department_staff_list, sex == "Female

# Sort previous result by years of service
department_female <- arrange(temp1, years_of_service)
```

```
# The same but with pipes:
department_female <- filter(department_staff_list, sex
                            arrange(years_of_service)
```

- The usefulness of the pipe (%>%) becomes more evident when the code starts to get more complicated.

Piping

```
# Filter only female employees:
temp1 <-  filter(department_staff_list, sex == "Female

# Sort previous result by years of service
department_female <- arrange(temp1, years_of_service)
```

```
# The same but with pipes:
department_female <- filter(department_staff_list, sex
                     arrange(years_of_service)
```

There are several important details to notice here:

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

Piping

```
# Filter only female employees:
temp1 <-  filter(department_staff_list, sex == "Female

# Sort previous result by years of service
department_female <- arrange(temp1, years_of_service)
```

```
# The same but with pipes:
department_female <- filter(department_staff_list, sex
                     arrange(years_of_service)
```

2.- 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

Exercise 2: filtering and sorting revisited

1. Apply the same filtering and sorting now with pipes

Solution

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

Piping

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

- 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**

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

Quick Summary Statistics

Refresher: Grouping and Summarizing Data

Yesterday, we learned how to:

1. **Group data** using `group_by()`
2. **Summarize** results with `summarise()`

This is a powerful tool to create **summary tables**—such as totals, averages, or counts—that are essential for your annual reports.

Example

```
# Summarize total revenue by month
summary_table <- psrl_data %>%
  group_by(Month) %>%
  summarise(Total_Revenue = sum(Revenue, na.rm = TRUE))

print(summary_table)
```

Quick summary statistics

Applying This to Your Work

Annual Report Table

Table 8: Programmed PSRL for 2023 (In GH¢)

Month	Diesel	LPG	Petrol	MGO-F	Gas Oil	Gas Oil	Unified/ Napththa	Total
Jan-23	23,872,267.72	3,682,560.43	31,661,429.70	55,898.08	5,237,078.13	753,524.51	18,461.87	65,281,220.45
Feb-23	24,309,842.16	3,182,459.63	28,976,008.73	50,014.07	4,427,220.69	674,206.14	24,212.29	61,643,963.71
Mar-23	24,096,033.18	3,818,951.55	28,386,586.25	48,248.87	4,211,258.70	650,410.63	24,212.29	61,235,701.47
Apr-23	26,358,863.90	3,409,778.17	31,439,960.64	46,483.67	4,049,287.21	626,615.12	30,265.37	65,961,254.09
May-23	24,599,710.89	3,500,705.59	31,281,630.87	44,130.07	4,103,277.71	594,887.77	24,212.29	64,148,555.19
Jun-23	27,312,455.37	3,909,878.97	31,923,406.89	43,541.66	3,995,296.72	586,955.93	24,212.29	67,795,747.84
Jul-23	27,123,060.55	4,091,733.81	32,816,321.77	42,364.86	3,887,315.73	571,092.26	24,212.29	68,556,101.27
Aug-23	25,959,652.42	4,000,806.39	31,640,924.49	41,188.06	4,265,249.20	555,228.58	24,212.29	66,487,261.44
Sep-23	21,590,679.66	4,000,806.39	33,402,163.23	52,367.68	4,589,192.18	705,933.48	27,238.83	64,368,381.44
Oct-23	27,235,725.40	3,909,878.97	27,364,475.58	52,956.08	4,697,173.17	713,865.32	27,238.83	64,001,313.35
Nov-23	28,242,906.95	3,909,878.97	36,501,774.26	54,132.88	5,129,097.14	729,728.99	26,936.18	74,594,455.37
Dec-23	27,965,119.88	4,046,270.10	39,826,295.00	57,074.88	5,399,049.62	769,388.18	27,238.83	78,090,436.49
Total	**308,666,318.08**	**45,463,708.96**	**385,220,977.42**	**588,400.87**	**53,990,496.19**	**7,931,836.90**	**302,653.68**	**802,164,392.11**

Source: NPA

Monthly totals for different products.

Code to Recreate the Table

```
# Summarize all numeric columns by Month
summary_table <- psrl_data %>%
  group_by(Month) %>%
  summarise(across(where(is.numeric), \(x) sum(x, na.r
```

Quick summary statistics

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

```
## # A tibble: 12 × 8
##    Month  Diesel   LPG Petrol MGO_F Gas_Oil Unified_Naphtha Total
##    <chr>   <dbl> <dbl>  <dbl> <dbl>   <dbl>            <dbl> <dbl>
##  1 Apr-23    140    70    115    40      35               22   422
##  2 Aug-23    170    85    135    48      45               30   513
##  3 Dec-23    200   100    155    60      55               40   610
##  4 Feb-23    130    65    105    35      30               18   383
##  5 Jan-23    120    60    100    30      25               15   350
##  6 Jul-23    160    80    130    50      42               28   490
##  7 Jun-23    155    78    125    45      40               26   469
##  8 Mar-23    125    63    110    33      28               20   379
##  9 May-23    150    75    120    42      38               24   449
## 10 Nov-23    190    95    150    58      52               38   583
## 11 Oct-23    180    90    145    55      50               35   555
## 12 Sep-23    165    83    140    47      48               32   515
```

Quick summary statistics

Beyond Basic Summaries: Customizing Your Results

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

But what if you want to ... ?

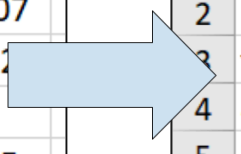
- ...export results in a different format (example: Excel)
- ...further customize which rows and columns to display in a result
- ...format the results you export

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	201911
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 `department_staff`

```
datasummary_skim(department_staff)
```

Quick summary statistics

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

	Unique	Missing Pct.	Mean	SD	Min	Median	Max	Histogram
years_of_service	2916	0	14.7	9.6	0.2	13.7	50.7	
age	5998	0	44.2	8.8	17.5	43.5	66.7	
		N	%					
sex	Female	3529	40.8					
	Male	5116	59.2					
senior_junior_staff	junior	2408	27.9					
	senior	6237	72.1					
department	Births & Deaths	165	1.9					
	Bureau of Languages	58	0.7					
	Co-operatives	268	3.1					
	Controller & Accountant General	3776	43.7					
	Department of Comm Devt	109	1.3					
	Dept of Children	99	1.1					
	Dept of Gender	100	1.2					
	Dept of Labour	290	3.4					
	Factories Inspectorate	121	1.4					

Quick summary statistics

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

```
datasummary_skim(department_staff, type = "categorical")
```

Quick summary statistics

		N	%
sex	Female	3529	40.8
	Male	5116	59.2
senior_junior_staff	junior	2408	27.9
	senior	6237	72.1
department	Births & Deaths	165	1.9
	Bureau of Languages	58	0.7
	Co-operatives	268	3.1
	Controller & Accountant General	3776	43.7
	Department of Comm Devt	109	1.3
	Dept of Children	99	1.1
	Dept of Gender	100	1.2
	Dept of Labour	290	3.4
	Factories Inspectorate	121	1.4

Quick summary statistics

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

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 number of observations, mean, standard deviation, minimum, and maximum for variables `years_of_service` of the dataframe `department_staff`

1. Use `datasummary()` for this:

```
datasummary(
  years_of_service  ~ N + Mean + SD + Min + Max,
  department_staff
)
```

	N	Mean	SD	Min	Max
years_of_service	8645	14.73	9.59	0.16	50.66

Customized summary statistics

```
datasummary(
   years_of_service  ~ N + Mean + SD + Min + Max, # this is the formula
   department_staff                              # 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

```
datasummary(
  years_of_service  ~ N + Mean + SD + Min + Max, # this is the formula
  department_staff                               # 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

Exporting tables

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(department_staff,
                 output = "quick_stats.docx")
```

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

Note for this code to work we would need to install an extra package `pandoc`

Exporting tables

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

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

Exercise 5: Export a table to Excel

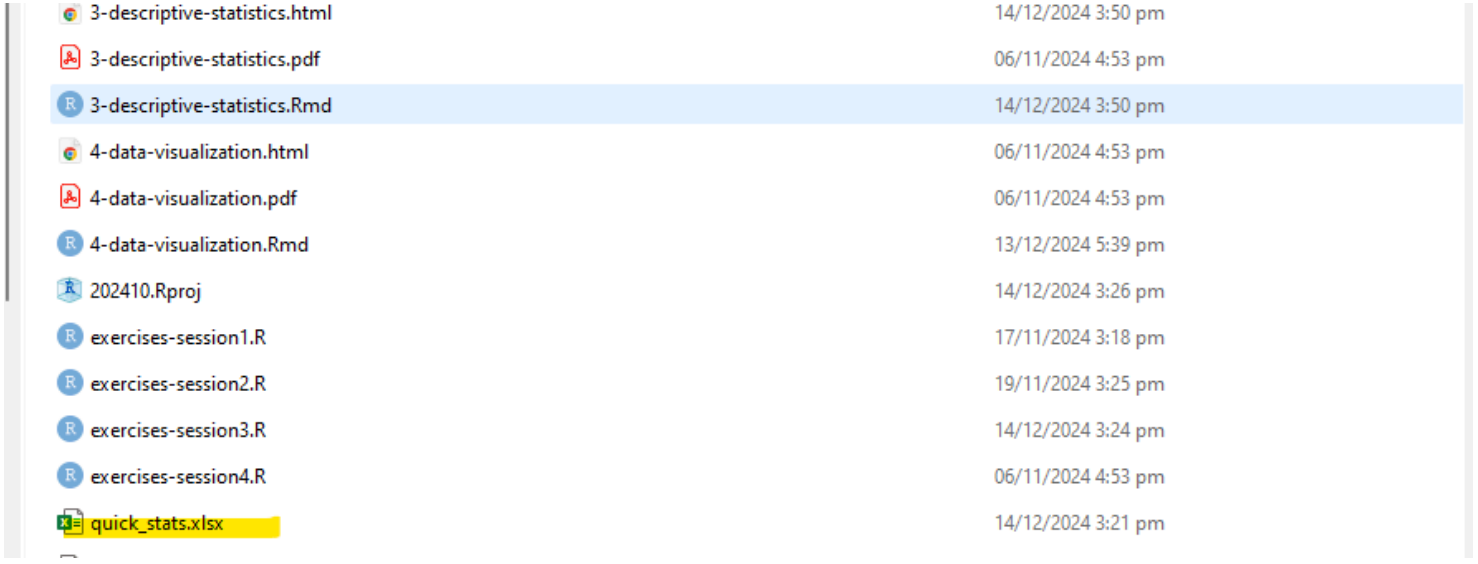
1. Load `huxtable` with `library(huxtable)` (we already did this at the beginning of the session)
2. Run the following code to export the result of `datasummary_skim()` to Excel:

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

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

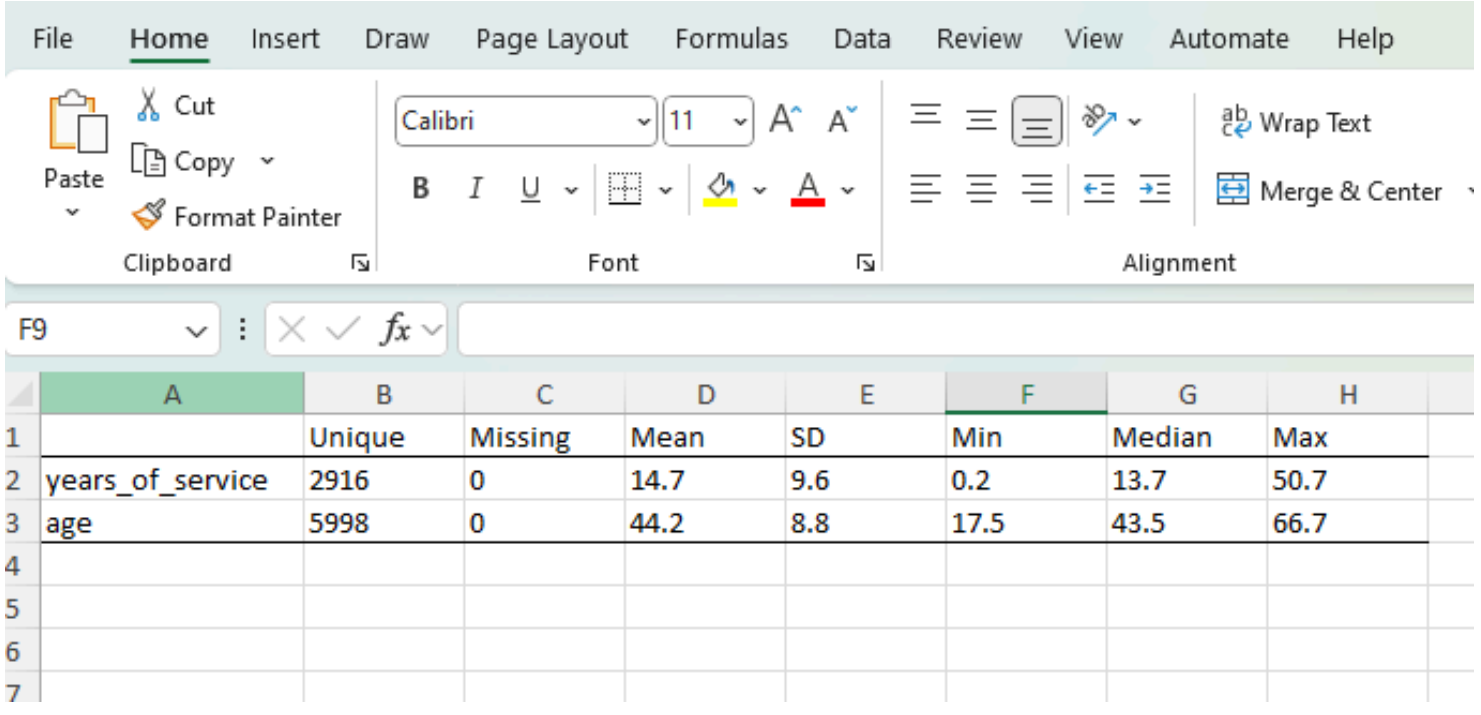
Exporting tables

Now the result will show in your `Documents` folder



Exporting tables

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



	A	B	C	D	E	F	G	H
1		Unique	Missing	Mean	SD	Min	Median	Max
2	years_of_service	2916	0	14.7	9.6	0.2	13.7	50.7
3	age	5998	0	44.2	8.8	17.5	43.5	66.7
4								
5								
6								
7								

Exporting tables

```
# Store the table in a new object
stats_table <- datasummary_skim(department_staff, 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

Customizing table outputs

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

```
# 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_blue()
```

	Unique	Missing Pct.	Mean	SD	Min	Median	Max
years_of_service	2916	0	15	10	0	14	51
age	5998	0	44	9	18	44	67

Customizing table outputs

Exercise 6: Export a customized table to Excel

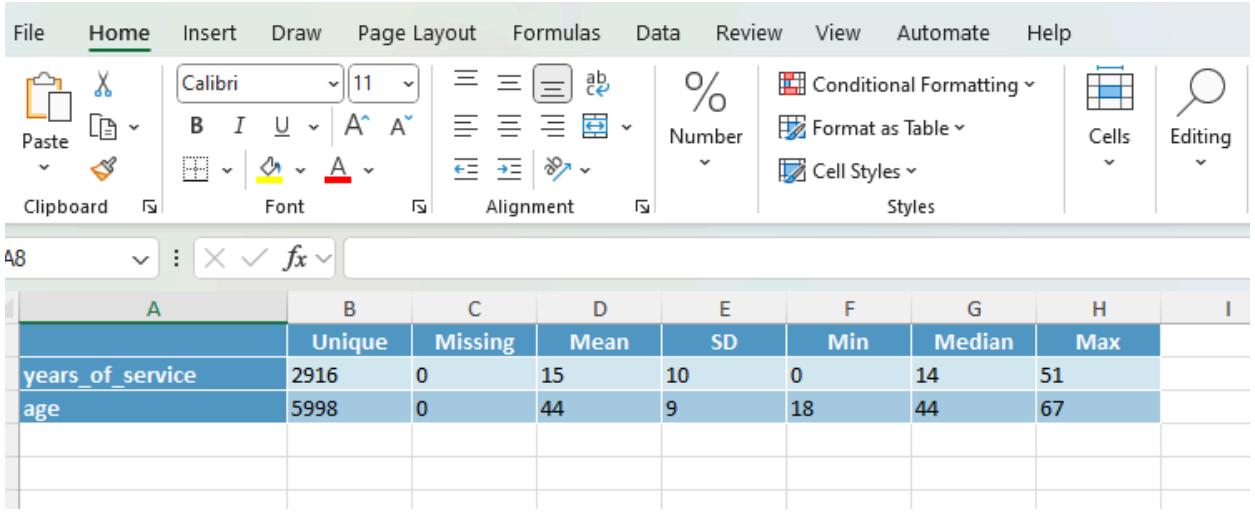
1.- Customize `stats_table` in a new object called `stats_table_custom`

```
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_blue()
```

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



	Unique	Missing	Mean	SD	Min	Median	Max
years_of_service	2916	0	15	10	0	14	51
age	5998	0	44	9	18	44	67

Customizing table outputs

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

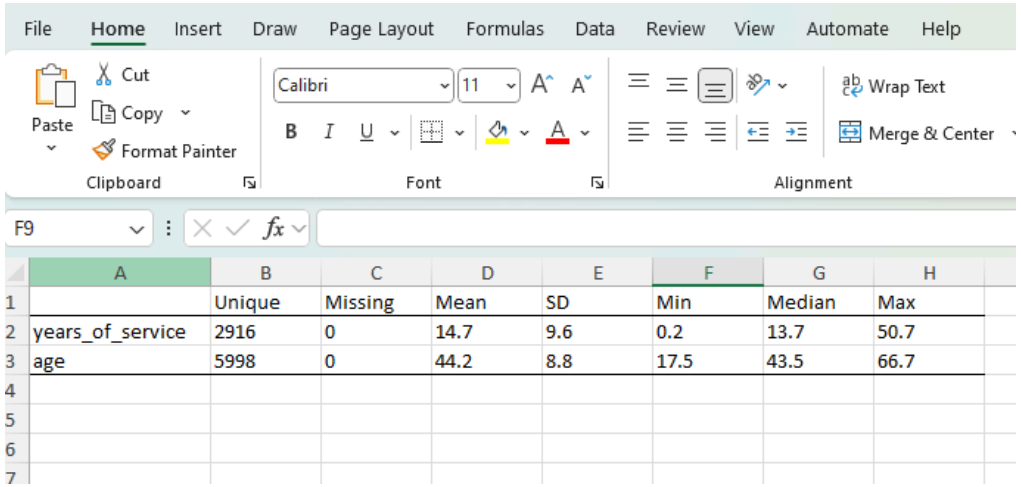
```
stats_table_custom <- stats_table %>%  # <---- here
  set_header_rows(1, TRUE) %>%
  set_header_cols(1, TRUE)  %>%
  set_number_format(everywhere, 2:ncol(.), "%9.0f") %>%
  set_align(1, everywhere, "center") %>%
  theme_blue()
```

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

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

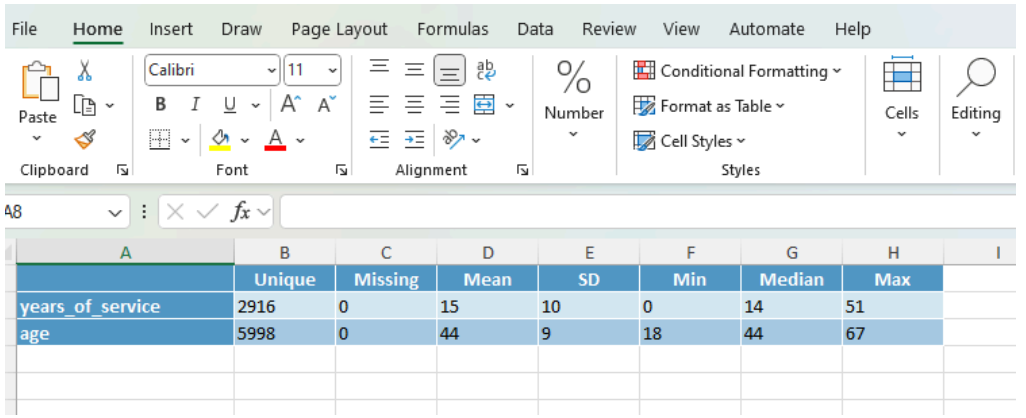
Customizing table outputs

Before:



	A	B	C	D	E	F	G	H
1		Unique	Missing	Mean	SD	Min	Median	Max
2	years_of_service	2916	0	14.7	9.6	0.2	13.7	50.7
3	age	5998	0	44.2	8.8	17.5	43.5	66.7
4								
5								
6								
7								

After:



	A	B	C	D	E	F	G	H	I
		Unique	Missing	Mean	SD	Min	Median	Max	
	years_of_service	2916	0	15	10	0	14	51	
	age	5998	0	44	9	18	44	67	

Customizing table outputs

We used `theme_blue()`. 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%

Use it on your work

Key Takeaways:

- This was a **basic example** with a few variables from your staff list, but the **possibilities are endless**.
- With this and the contents from yesterday's session, you can create **summaries** of **anything you can think of**.

Real-World Example:

Annual Report: Programmed vs. Billings for 2023 (In GH¢)

Table 9: Programmed versus Billings for 2023 (In GH¢)

Month	Programme	Actual Billings	Variance	%Variance
Jan	65,281,220.45	56,009,815.06	(9,271,405.39)	-14.20%
Feb	61,643,963.71	53,401,114.00	(8,242,849.71)	-13.37%
Mar	61,235,701.47	65,176,751.96	3,941,050.49	6.44%
Apr	65,961,254.09	60,430,666.88	(5,530,587.21)	-8.38%
May	64,148,555.19	66,670,601.74	2,522,046.55	3.93%
Jun	67,795,747.84	60,291,190.36	(7,504,557.48)	-11.07%
Jul	68,556,101.27	63,948,059.60	(4,608,041.67)	-6.72%
Aug	66,487,261.44	64,451,574.60	(2,035,686.84)	-3.06%
Sep	64,368,381.44	59,259,571.44	(5,108,810.00)	-7.94%
Oct	64,001,313.35	59,850,932.04	(4,150,381.31)	-6.48%
Nov	74,594,455.37	68,070,927.90	(6,523,527.47)	-8.75%
Dec	78,090,436.49	66,271,765.20	(11,818,671.29)	-15.13%
Total	**802,164,392.11**	**743,832,970.78**	**(58,331,421.33)**	**-7.27%**

Source: NPA

- If we have the data, you can easily create summaries like this directly in R.
- Once written, the code can be re-used for the next year or quarter.

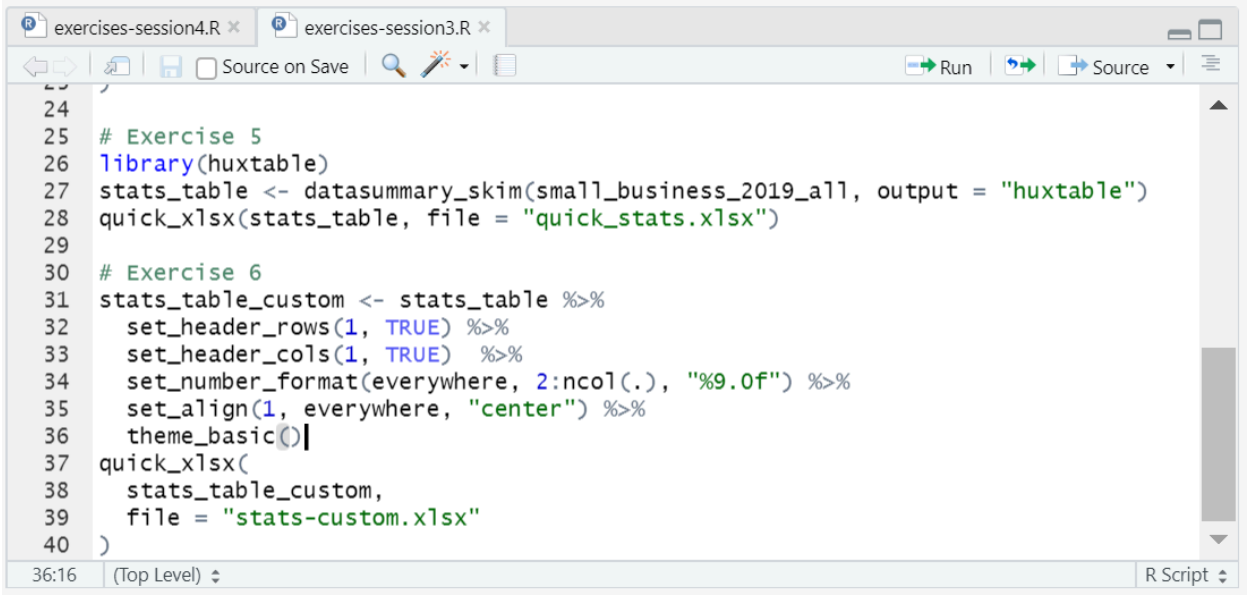
Questions?

Wrapping up

Wrapping up

Save your work!

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



Wrapping up

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

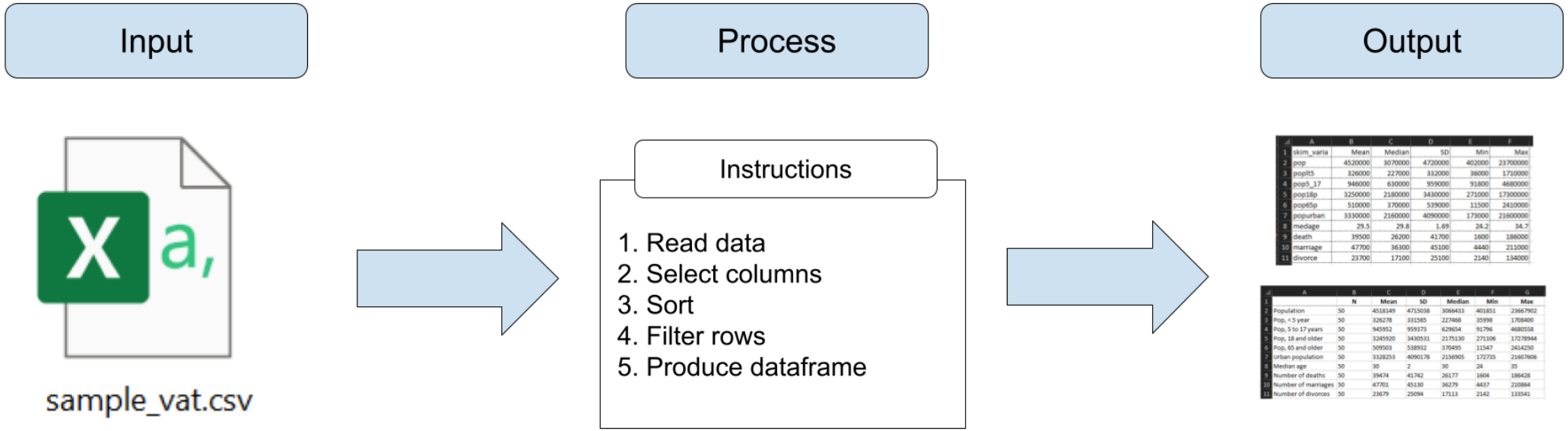
What else is available?

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

This session

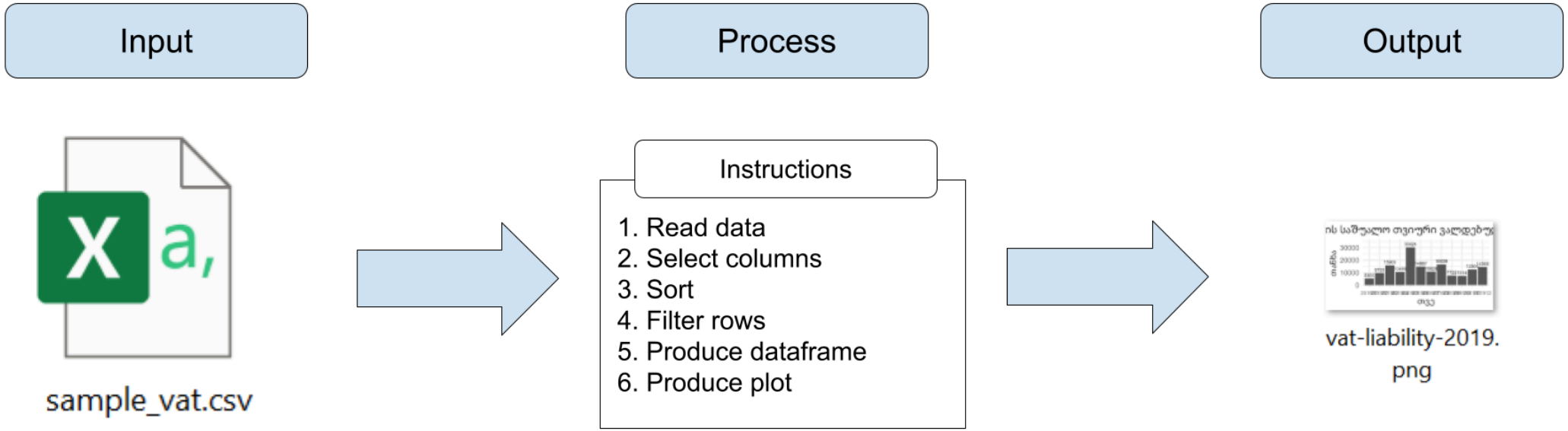


	A	B	C	D	E	F
1	skim_varia	Mean	Median	SD	Min	Max
2	pop	4520000	3070000	4720000	402000	23700000
3	poplt5	326000	227000	332000	36000	1710000
4	pop5_17	946000	630000	959000	91800	4680000
5	pop18p	3250000	2180000	3430000	271000	17300000
6	pop65p	510000	370000	539000	11500	2410000
7	popurban	3330000	2160000	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

Next session (last one)



Thanks! // ¡Gracias! // Obrigado!