Printing (PDF)

Basic printing

If a tabulation function is called from the top level, it should print out its table(s) on its own.

As usual, first, let's start up the package and pick a survey to analyze:

```
library(surveytable)
set_survey(namcs2019sv)
```

Table 1: Survey info {NAMCS 2019 PUF}

Variables	Observations	Design
33	8,250	Stratified 1 - level Cluster Sampling design (with replacement) With (398) clusters. namcs2019sv = survey::svydesign(ids = ~CPSUM, strata = ~CSTRATM, weights = ~PATWT , data = namcs2019sv_df)

Now, when a tabulation function is called from the top level, it prints. You don't need to do anything extra.

```
tab("AGER")
```

Table 2: Patient age recode {NAMCS 2019 PUF}

Level	n	Number	SE	LL	UL	Percent	SE	LL	UL
Under 15 years	887	117,916,772	14,097,315	93,228,928	149,142,177	11.4	1.3	8.9	14.2
15-24 years	542	64,855,698	7,018,359	52,386,950	80,292,164	6.3	0.6	5.1	7.5
25-44 years	1,435	170,270,604	13,965,978	144,924,545	200,049,472	16.4	1.1	14.3	18.8
45-64 years	2,283	309,505,956	23,289,827	266,994,092	358,786,727	29.9	1.4	27.2	32.6
65-74 years	1,661	206,865,982	14,365,993	180,480,708	237,108,637	20.0	1.2	17.6	22.5
75 years and over	1,442	167,069,344	15,179,082	139,746,193	199,734,713	16.1	1.3	13.7	18.8

N = 8250.

If a tabulation function is called not from the top level, such as from within a loop or another function, you do need to call `print()` explicitly for it to print. For example:

```
for (vr in c("AGER", "SEX")) {
  print( tab_subset(vr, "MAJOR", "Preventive care") )
}
```

Table 3: Patient age recode (Major reason for this visit = Preventive care) {NAMCS 2019 PUF}

Level	n	Number	SE	LL	UL	Percent	SE	LL	UL
Under 15 years	300	50,700,892	8,555,609	36,351,714	70,714,146	22.7	3.5	16.1	30.4
15-24 years	121	18,196,389	2,888,616	13,246,305	24,996,296	8.1	1.2	5.9	10.9
25-44 years	370	50,573,223	6,834,740	38,749,084	66,005,455	22.6	2.5	17.8	28.0
45-64 years	355	53,804,610	9,477,599	37,982,129	76,218,371	24.1	3.2	17.9	31.1
65-74 years	225	27,985,400	4,668,693	20,072,754	39,017,198	12.5	1.8	9.2	16.5
75 years and over	197	22,363,158	3,804,827	15,925,231	31,403,678	10.0	1.7	6.9	13.8

N = 1568.

Table 4: Patient sex (Major reason for this visit = Preventive care) {NAMCS 2019 PUF}

Level	n	Number	SE	LL	UL	Percent	SE	LL	UL
Female	1,014	139,091,345	11,844,812	117,664,165	164,420,512	62.2	2.9	56.2	68.0
Male	554	84,532,326	10,593,549	66,039,112	108,204,272	37.8	2.9	32.0	43.8

N = 1568.

Create HTML or PDF tables

Using a Quarto document, you can create tables in many different formats, such as HTML or PDF. Here is a straightforward example of what a Quarto document might look like:

````
---
title: "My tables"
author: "Me"
format: pdf
---

# Welcome

As usual, first, let's start up the package and pick a survey to analyze:

```{r, results='asis'}
library(surveytable)
set_survey(namcs2019sv, output = 'auto')
```

# Tables

Take a look at this table:

```{r, results='asis'}
tab("AGER")
```
````

Note the `format` setting, which specifies that this document will create PDF tables. Also note that you do have to add the `results='asis'` argument to the code chunks that print tables.

Print using various table-making packages

You can change the package that `surveytable` uses for printing. `surveytable` comes with code for using one of these packages: `huxtable` (default), `gt`, and `kableExtra`. There is also the option to automatically select one of these packages depending on whether the output is to the screen (`huxtable`), HTML (`gt`), or PDF (`kableExtra`). In addition, you can supply custom code to use any table-making package of your choice.

Changing the table-making package has a couple of uses:

- Use `as_object()` to generate an object from your favorite table-making package, customize this object, and then finally print it, so the table looks exactly the way you want it to look.
- Print to destinations other than the screen, such as HTML or PDF.

kableExtra

By default, `surveytable` prints using `huxtable`. However, at this point, we have only implemented PDF printing with `kableExtra`. This is the reason that, in a PDF document, you do need to switch to `kableExtra` output:

```
set_opts(output = "kableExtra")
#> * Printing with kableextra.
```

Once you do that, produce PDF tables like so:

````
```{r, results='asis'}
tab("AGER")
```
````

Table 5: Patient age recode {NAMCS 2019 PUF}

Level	n	Number	SE	LL	UL	Percent	SE	LL	UL
Under 15 years	887	117,916,772	14,097,315	93,228,928	149,142,177	11.4	1.3	8.9	14.2
15-24 years	542	64,855,698	7,018,359	52,386,950	80,292,164	6.3	0.6	5.1	7.5
25-44 years	1,435	170,270,604	13,965,978	144,924,545	200,049,472	16.4	1.1	14.3	18.8
45-64 years	2,283	309,505,956	23,289,827	266,994,092	358,786,727	29.9	1.4	27.2	32.6
65-74 years	1,661	206,865,982	14,365,993	180,480,708	237,108,637	20.0	1.2	17.6	22.5
75 years and over	1,442	167,069,344	15,179,082	139,746,193	199,734,713	16.1	1.3	13.7	18.8

N = 8250.

auto

This option automatically selects one of the above packages depending on whether the output is to the screen (`huxtable`), HTML (`gt`), or PDF (`kableExtra`).

```
set_opts(output = "auto")
#> * Printing with huxtable for screen, gt for HTML, or kableExtra for LaTeX.
```

PDF output (this should use `kableExtra`):

```{r, results='asis'}
tab("AGER")
```

Table 6: Patient age recode {NAMCS 2019 PUF}

Level	n	Number	SE	LL	UL	Percent	SE	LL	UL
Under 15 years	887	117,916,772	14,097,315	93,228,928	149,142,177	11.4	1.3	8.9	14.2
15-24 years	542	64,855,698	7,018,359	52,386,950	80,292,164	6.3	0.6	5.1	7.5
25-44 years	1,435	170,270,604	13,965,978	144,924,545	200,049,472	16.4	1.1	14.3	18.8
45-64 years	2,283	309,505,956	23,289,827	266,994,092	358,786,727	29.9	1.4	27.2	32.6
65-74 years	1,661	206,865,982	14,365,993	180,480,708	237,108,637	20.0	1.2	17.6	22.5
75 years and over	1,442	167,069,344	15,179,082	139,746,193	199,734,713	16.1	1.3	13.7	18.8

N = 8250.

Advanced printing

The proper approach

Advanced users can add functionality to use **any** table-making package that they want. For more information, see `help("surveytable-options")`.

The "quick-and-dirty" approach

The tabulation functions return either:

- for a single table, a data frame, with certain attributes set; or
- for more than one table, a list of such data frames.

You can convert a single table to a `data.frame` with `as.data.frame()`, like so:

```
tab("AGER") |> as.data.frame()
#>                Level    n    Number       SE       LL        UL Percent  SE
#> 1    Under 15 years  887 117916772 14097315 93228928 149142177    11.4 1.3
#> 2       15-24 years  542  64855698  7018359 52386950  80292164     6.3 0.6
#> 3       25-44 years 1435 170270604 13965978 144924545 200049472    16.4 1.1
#> 4       45-64 years 2283 309505956 23289827 266994092 358786727    29.9 1.4
#> 5       65-74 years 1661 206865982 14365993 180480708 237108637    20.0 1.2
#> 6 75 years and over 1442 167069344 15179082 139746193 199734713    16.1 1.3
#>     LL   UL
#> 1  8.9 14.2
#> 2  5.1  7.5
#> 3 14.3 18.8
#> 4 27.2 32.6
#> 5 17.6 22.5
#> 6 13.7 18.8
```

Alternatively, you can pass this data frame to your favorite table-making package. This example passes it to `gt`:

Level	n	Number (000)	SE (000)	LL (000)	UL (000)	Percent	SE	LL	UL
Under 15 years	887	117917	14097	93229	149142	11.4	1.3	8.9	14.2
15-24 years	542	64856	7018	52387	80292	6.3	0.6	5.1	7.5
25-44 years	1435	170271	13966	144925	200049	16.4	1.1	14.3	18.8
45-64 years	2283	309506	23290	266994	358787	29.9	1.4	27.2	32.6
65-74 years	1661	206866	14366	180481	237109	20.0	1.2	17.6	22.5
75 years and over	1442	167069	15179	139746	199735	16.1	1.3	13.7	18.8

```
set_opts(count = "1k")
#> * Rounding counts to the nearest thousand.
tab("AGER") |> gt::gt()
```

(Because of how LaTeX works, the table is likely not here, but elsewhere on the page.)

The reason that this is the "quick-and-dirty" approach is that the output it creates is not as nice as conventional tables, described above. The output does not have table title (which has important information about the variable and the survey), table footer (which has important information about sample size and low-precision estimates), and it does not format the estimates. Nevertheless, there could be situations in which this approach is helpful, such as

- extracting an exact value from a table using `as.data.frame()`; or
- quickly using your favorite table-making package.

Save the tables

Save to a CSV file

All tabulation functions have an argument called `csv`. Use it to specify the name of a CSV (comma-separated values) file, like so:

```
tab("AGER", csv = "myfile.csv")
```

Table 7: Patient age recode {NAMCS 2019 PUF}

Level	n	Number (000)	SE (000)	LL (000)	UL (000)	Percent	SE	LL	UL
Under 15 years	887	117,917	14,097	93,229	149,142	11.4	1.3	8.9	14.2
15-24 years	542	64,856	7,018	52,387	80,292	6.3	0.6	5.1	7.5
25-44 years	1,435	170,271	13,966	144,925	200,049	16.4	1.1	14.3	18.8
45-64 years	2,283	309,506	23,290	266,994	358,787	29.9	1.4	27.2	32.6
65-74 years	1,661	206,866	14,366	180,481	237,109	20.0	1.2	17.6	22.5
75 years and over	1,442	167,069	15,179	139,746	199,735	16.1	1.3	13.7	18.8

N = 8250.

Open this CSV file in Excel or your favorite text editor or spreadsheet.

Save to an R data file

Use the built-in `saveRDS()` function to save a table to an R data file:

```
tab("AGER") |> saveRDS("myfile.rds")
```

You can later load this data file back into R. To print the table, just load the file, like so:

```
readRDS("myfile.rds")
```

Table 8: Patient age recode {NAMCS 2019 PUF}

Level	n	Number (000)	SE (000)	LL (000)	UL (000)	Percent	SE	LL	UL
Under 15 years	887	117,917	14,097	93,229	149,142	11.4	1.3	8.9	14.2
15-24 years	542	64,856	7,018	52,387	80,292	6.3	0.6	5.1	7.5
25-44 years	1,435	170,271	13,966	144,925	200,049	16.4	1.1	14.3	18.8
45-64 years	2,283	309,506	23,290	266,994	358,787	29.9	1.4	27.2	32.6
65-74 years	1,661	206,866	14,366	180,481	237,109	20.0	1.2	17.6	22.5
75 years and over	1,442	167,069	15,179	139,746	199,735	16.1	1.3	13.7	18.8

N = 8250.

Suppress printing

There are times when you might want to prevent the tabulation functions from printing tables. If you are saving the tables to a CSV file anyway, you might not need screen printing.

As mentioned above, if the tabulation functions are called from within a loop without using the `print()` command, they won't print.

An easy way to suppress printing when the tabulation functions are called from the top level is to assign the output to some variable. For example, this will save the table to a CSV file, but won't print it to the screen:

```
tmp = tab("AGER", csv = "myfile.csv")
```