`waffle`: A new command for creating waffle charts in Stata

Jared Colston
University of Wisconsin-Madison
colston@wisc.edu

Abstract. In this article, I describe `waffle`, a command to produce waffle charts in Stata. `waffle` is a wrapper for `twoway scatter` that produces a 10x10 or 5x20 grid of squares that are conditionally colored based on specified proportions or percentages. I illustrate the use of `waffle` first by recreating the charts in Asjad Naqvi's blog post introducing the method, and then expanding upon this to illustrate the range of options and additional features I have added to the program using Stata's preset `pop2000` dataset.

Keywords: waffle, data visualization, graphics, twoway

1 Introduction

Waffle charts allow for a new way to present proportions and percentages. Similar to stacked bar charts, pie charts, or donut charts, waffle charts provide a useful way of describing proportional data and presenting it in an interesting and aesthetically pleasing way. They add just another tool to the toolkit of analysts who are wishing to improve their data communication and make their reporting less table-intensive and more appealing to broader audiences.

Waffle charts are a recent development in descriptive statistics and data visualization. Currently, both R and Python have packages that construct waffle charts based on the percentages or proportions that you indicate. This package is intended to introduce that functionality into Stata. In October 2021, Dr. Asjad Naqvi introduced the method through which to calculate waffle charts and plot them using `twoway scatter` commands. The methodology section of this article will discuss his process as well as some minor changes I have made to allow for multiple categories in a single waffle chart.

This article is organized as follows. In section 2, I describe the methodology first discussed in Dr. Naqvi's web post and add the additional calculations for multi-category waffle charts. In section 3, I describe the `waffle` command. In section 4, I illustrate the use of `waffle` by briefly recreating Dr. Naqvi's graphs and then expanding upon that using Stata's preset `pop2000` dataset. In section 5 I conclude.

2 Methodology

Waffles in Stata are generated by creating a grid of observations in the data frame. This is done by storing the desired number of rows and columns as local macros and expanding the

observations based on those macros. In `waffle`, the standard grid is 10x10, so the underlying code looks like this:

```
local rows = 10
local cols = 10
local obsv = `cols' * `rows'
expand `obsv'
```

In `waffle`, if the option `wide` is specified, then this code changes to:

```
local rows = 20
local cols = 5
```

The reason that rows are 20 and not 5 stems from how we want the waffle to sequentially fill in the colors of our proportion further down in the process. By reversing the 5x20 to 20x5 here, we can reverse it again when plotting `twoway scatter y x` to `twoway scatter x y` and still get the 5x20 grid we desire with the colors filling in correctly.

After this, we generate variables **x** and **y** using the sequence command in `egen` based on our `cols` macro:

```
egen y = seq(), block(`cols')
egen x = seq(), to(`cols')
```

Optionally, if a categorical variable is specified in `by()`, then the sequential code takes that into account, creating a grid for each category:

```
bysort `byvar' :  egen y = seq(), block(`cols')
                  egen x = seq(), to(`cols')
```

After we create the waffle structure, we need to indicate how we want the grid to be colored. First, we generate an `id` variable equal to the row number in our data frame and generate a `dot` variable that will be assigned a 1 if our proportion meets the threshold:

```
gen id = _n
gen dots = .
```

Optionally, if we have specified a `by()` variable, then we also create a tag conditional on the grid in question (because previously we created a grid based on each category in `by()`):

```
by `byvar': gen id = _n
egen tag = tag(`byvar')
gen dots = .
```

Next, we tell Stata to summarize and obtain the mean value of our proportion; the mean is obtained simply because there should be no variation in the proportion variable. We then compare that mean value to our previously generated `id` variable using a Boolean, where our `dots` variable is assigned a 1 if our `id` variable is less or equal to our mean proportion value:

```
summarize `proportion_var'
local share = `r(mean)'
```

```
summarize id
replace dots = id <= int(`share' * `r(max)')
```

If `by()` is specified then we perform this calculation for each of our categorical grids:

```
levelsof `byvar', local(lvls)

foreach x of local lvls {
      summarize `proportion_var' if `byvar' == `x' & tag == 1
      local share = `r(mean)'

      summarize id if `byvar' == `x'
      replace dots = id <= int(`share' * `r(max)') if `byvar' == `x'
}
```

In this code, the local macro `'x'` is not surrounded by double quotes, indicating that the categorical `by()` variable must be a numeric variable. However, in the `waffle` program, this is accounted for by identifying string `by()` variables beforehand and temporarily converting them the numeric so that they can still be used.

With this code, we now have our single-color waffle charts with potentially several categories specified in `by()`. There is also the option, however, to specify several categories in a single waffle chart, showing multiple categories within a single proportion.

To produce waffle charts with multiple categories, we must generate two additional variables that will define how squares are colored. Prior to generating the sequential **y** and **x** variables that define the waffle structure, we will create:

```
gen seq_cat1 = `proportion_var1'
gen seq_cat2 = seq_cat1 + `proportion_var2'
```

This process can be repeated for as many categorical variables that have been specified. I have limited the maximum number of allowable categories to 5 for visualization's sake. What this process does is creates sequential color categories that are equal to each proportion we have, but sets them to be additive. If our first proportion variable is .12 and our second proportion variable is .10, our current method would prevent the second proportion variable from showing up in the waffle at all. This additive process transforms our second proportional variable to .22 (.12 + .10) to allow it to show on top of the .12 proportion. After we have created these additive variables, we generate empty color categories in place of the aforementioned `dots` variable:

```
gen id = _n
gen color_cat1 = .
gen color_cat2 = .

summarize seq_cat1
local share = `r(mean)'
summarize id
replace color_cat1 = id <= int(`share' * `r(max)')

summarize seq_cat2
local share = `r(mean)'
```

```
summarize id
replace color_cat2 = id <= int(`share' * `r(max)')
```

After this, we combine these separate Booleans into a single categorical variable called `category` where the categories do not overlap:

```
gen category = .
replace category = color_cat1
replace category = 2 if color_cat2 == 1 & category == 0
```

The same logic of this process can be repeated with up to 5 proportions, with the resulting code:

```
gen category = .
replace category = color_cat1
replace category = 2 if color_cat2 == 1 & category == 0
replace category = 3 if color_cat3 == 1 & category == 0
replace category = 4 if color_cat4 == 1 & category == 0
replace category = 5 if color_cat5 == 1 & category == 0
```

When our waffle structure is set, it is simply a matter of plotting several `twoway scatter` plots together conditional on either the `dots` or `category` variable:

```
twoway      (scatter y x if dots == 1) ///
            (scatter y x if dots == 0)
```

or:

```
twoway      (scatter y x if category == 1) ///
            (scatter y x if category == 2) ///
            (scatter y x if category == 3) ///
            (scatter y x if category == 4) ///
            (scatter y x if category == 5)
```

Additional formatting of these plots are built into the `waffle` command and will be discussed in the following sections.

3 The `waffle` command

3.1 Syntax

The syntax for the `waffle` command is

```
waffle varlist [if] [in], [wide by(varlist) colors(colorlist)
      emptycolors(colorlist) outlinecolors(colorlist)
      emptyoutlinecolors(colorlist) markersize(#) scheme(schemename)
      title(tinfo) note(tinfo) name(name [, replace])
      legend([contents] [location])]
```

3.2 Options

Waffle Design

`wide` transforms the 10x10 grid default to a 5x20 grid for all waffles being drawn.

`by(varlist)` produces a waffle chart for each category in *varlist*. This option is not allowed if more than one variable is listed in `by()` and is also not allowed with more than one percent/decimal specified. If you would like multiple waffle charts with multiple color categories, I recommend drawing multiple individually and combining using `graph combine` or the user-written command `grc1leg`. Can be used with select `twoway by()` options, such as `rows()`.

`colors(colorlist)` allows you to specify the colors of your percent/decimal variables. Colors will be assigned in order of your specified *varlist*. Default colors are based on the Tableau scheme from `schemepack`.

`emptycolors(colorlist)` allows you to specify the colors of the "empty" squares in the waffle. The empty squares are all squares up to 100 not included in your defined percent/decimal variables. The default color is *gs14*.

`outlinecolors(colorlist)` allows you to specify the outline colors of your filled in waffle squares. The default is *none*.

`emptyoutlinecolors(colorlist)` allows you to specify the outline colors of the "empty" squares in the waffle. This is separated from `outlinecolors` in the event that you would like to make the empty squares white with an outline, etc.

`markersize(#)` allows you to alter the size of the markers. The default sizes are based on other options, designed to adjust for number of categories in `by()` and whether or not `wide` is specified. The size is also dependent on the aspect ratio, which changes depending on the `wide` specification.

`scheme(schemename)` allows you to change the scheme of the chart. Because other graph aspects, such as axes, are not allowed, this mostly determines the look of the background and the legend. A recommended scheme is `white_tableau` from `schemepack` (available on ssc).

Add Information

`title(tinfo)` allows you give your chart a title. If by()`is specified, this will be the overall title, with individual waffle titles being identified by the value labels in`by(varlist)`.

`note(tinfo)` adds a note at the bottom left of the chart.

`name(name [, replace])` gives the chart a name Stata will recognize, which is important if you would like to `combine` multiple graphs.

legend([contents] [location]) determines the look of a legend. All legend options allowed in twoway are allowed here. This option is only available if multiple categories are specified.

4 Examples

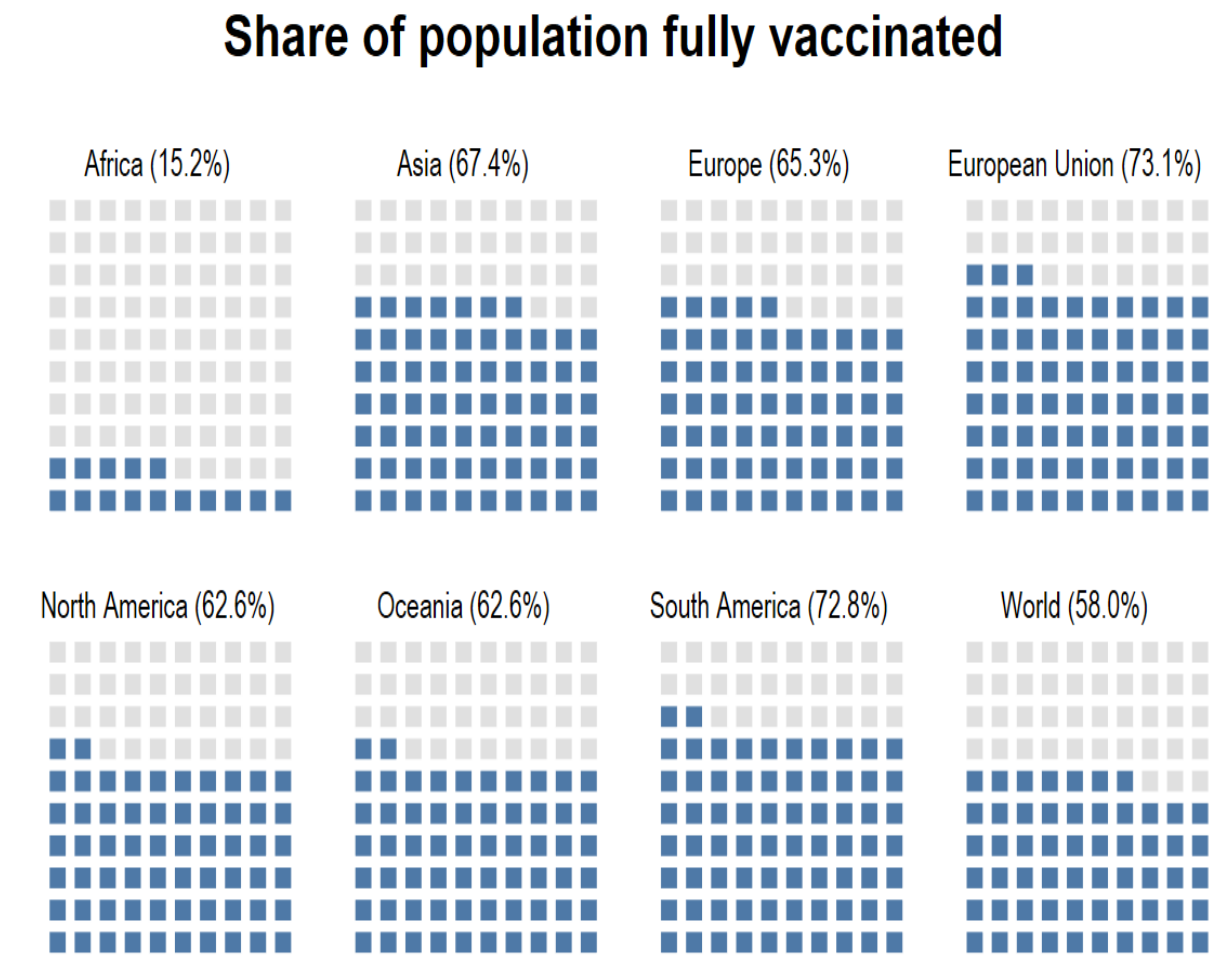
Example 1: Recreate chart from Dr. Naqvi's post

For the first example of waffle, I will recreate the chart made by Dr. Naqvi in his previous web post. First, we will import and prep the data in the same manner as him. For further discussion of what this data prep code is accomplishing, I recommend reading his linked post. The waffle command has also been added to the SSC repository, so we can install it from there.

```
        ssc install waffle, replace
        ssc install schemepack, replace
        set scheme white_tableau
        graph set window fontface "Arial Narrow"
        import delim using "https://covid.ourworldindata.org/data/owid-covid-
data.csv", clear
        gen date2 = date(date, "YMD")
        format date2 %tdDD-Mon-yy
        drop date
        ren date2 date
        ren location country
        keep iso_code continent country date people_fully_vaccinated
population
        keep if length(iso_code) > 3
        tab country
        drop if inlist(iso_code,"OWID_KOS","OWID_CYN","OWID_HIC")
        drop if inlist(iso_code,"OWID_LIC","OWID_LMC","OWID_UMC")
        bysort country: egen last = max(date) if people_fully_vaccinated != .
        keep if date == last
        summ date
        global dateval: di %tdd_m_y `r(max)'
        di "$dateval"
        gen share = people_fully_vaccinated / population
        gen country2 = country + " (" + string(share * 100, "%9.1f") + "%)"
```

Now we can use waffle to create the chart from the web post. Note that the subtitles over each chart are determined by the value labels of the categorical by() variable:

```
      waffle share, by(country2, rows(2)) mark(2) ///
            title("{fontface Arial Bold:Share of population fully
vaccinated}", margin(medlarge)) ///
            note("Source: Our World in Data. Data updated: $dateval.")
```



Example 2: Use `pop2000` to showcase the range of the `waffle` command

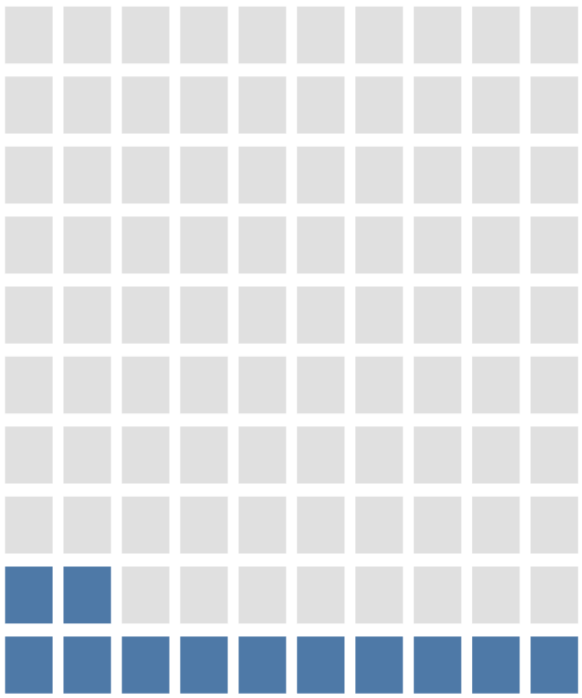
We can now turn to a preset data file in Stata to demonstrate the other options and functionality of `waffle`. First, we need to load and prep the data:

```
// Pull in the data
        sysuse pop2000, clear
// Create a smaller category for examples
        gen small_age_cat = 0
        replace small_age_cat = 1 if inrange(agegrp,1,6)
        replace small_age_cat = 2 if inrange(agegrp,7,13)
        replace small_age_cat = 3 if inrange(agegrp,14,17)
        label define age_cat 1 "Under 30" 2 "30-64" 3 "65 & up"
        lab val small_age_cat age_cat
// Create total percents for each race/ethnic category
        foreach x in black white asian total {
          egen tot_`x' = sum(`x')
        }
        gen pct_tot_black = tot_black / tot_total
```

```
                gen pct_tot_white = tot_white / tot_total
                gen pct_tot_asian = tot_asian / tot_total
        // Create category-specific percent variables
                egen small_age_tot_total = sum(total), by(small_age_cat)
                foreach x in white black asian {
                  gen pct_`x'_age = `x' / total
                  egen small_age_tot_`x' = sum(`x'), by(small_age_cat)
                  gen pct_`x’_small_age = small_age_tot_`x' /
small_age_tot_total
                  drop small_age_tot_`x'
                }
        // Only keep the variables we are interested in
                drop total-femisland tot_black-tot_total small_age_tot_total
```

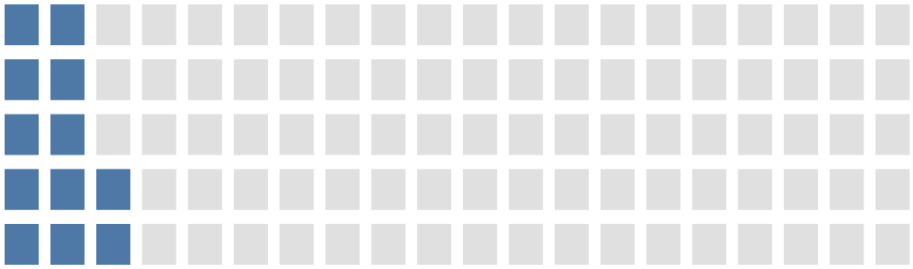
Now we can create a simple waffle chart of the total share of Black population in the 2000 Census:

```
waffle pct_tot_black
```



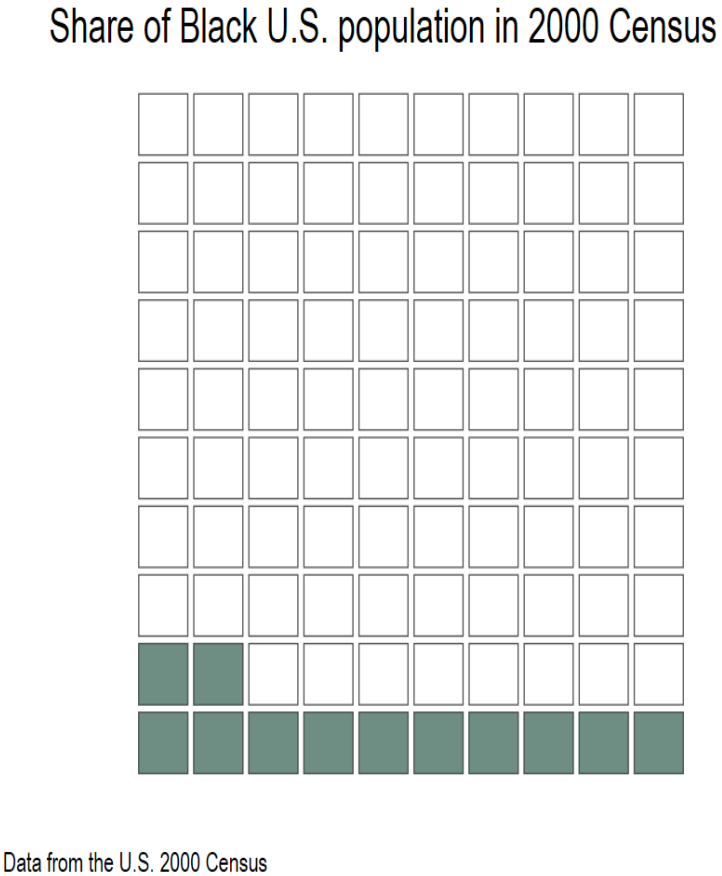
Next, we can alter the structure of the waffle by specifying that we would like to use `wide` mode:

```
waffle pct_tot_black, wide
```



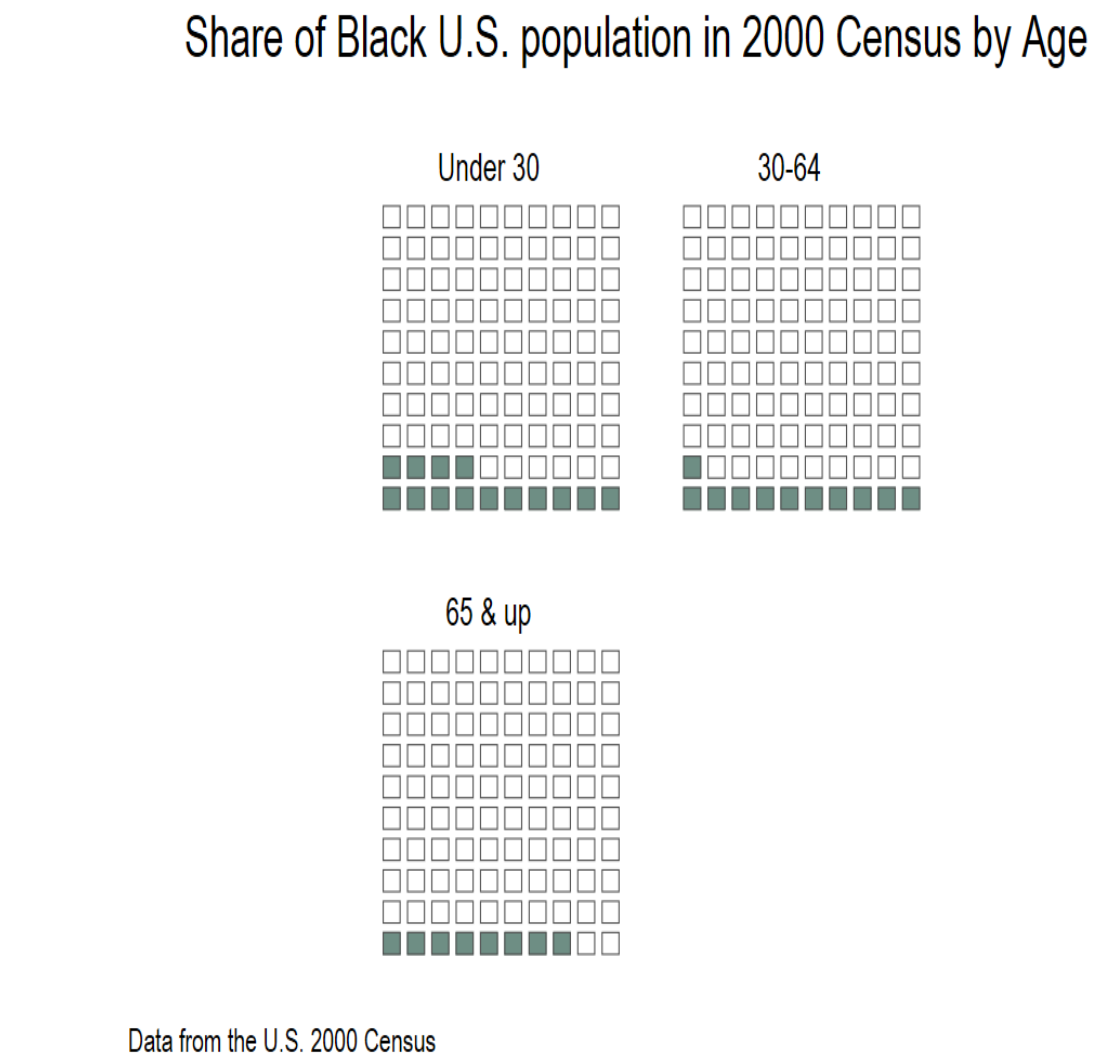
Turning back to standard mode, we can add additional elements to our chart to improve readability and alter some design features. Here, we will add a title, a note on the data source, and adjust the colors and outlines of the squares. Notice that we also slightly adjust the marker size. The reason for this is that adding additional elements to the chart can alter the aspect ratio, so occasionally the marker sizes must be adjusted following any additions:

```
      waffle pct_tot_black, ///
            title("Share of Black U.S. population in 2000 Census",
margin(medlarge)) ///
            note("Data from the U.S. 2000 Census") ///
            markersize(6) colors(teal) emptycolors(white) outlinecolors(gs5)
emptyoutlinecolors(gs5)
```



We can also incorporate the `by()` option to look at the share of respondents that are Black by age group. Note that in addition to specifying the `by()` option we have also chosen the group-specific variable `pct_black_small_age` rather than `pct_tot_black`, which is necessary for `waffle` to function properly:

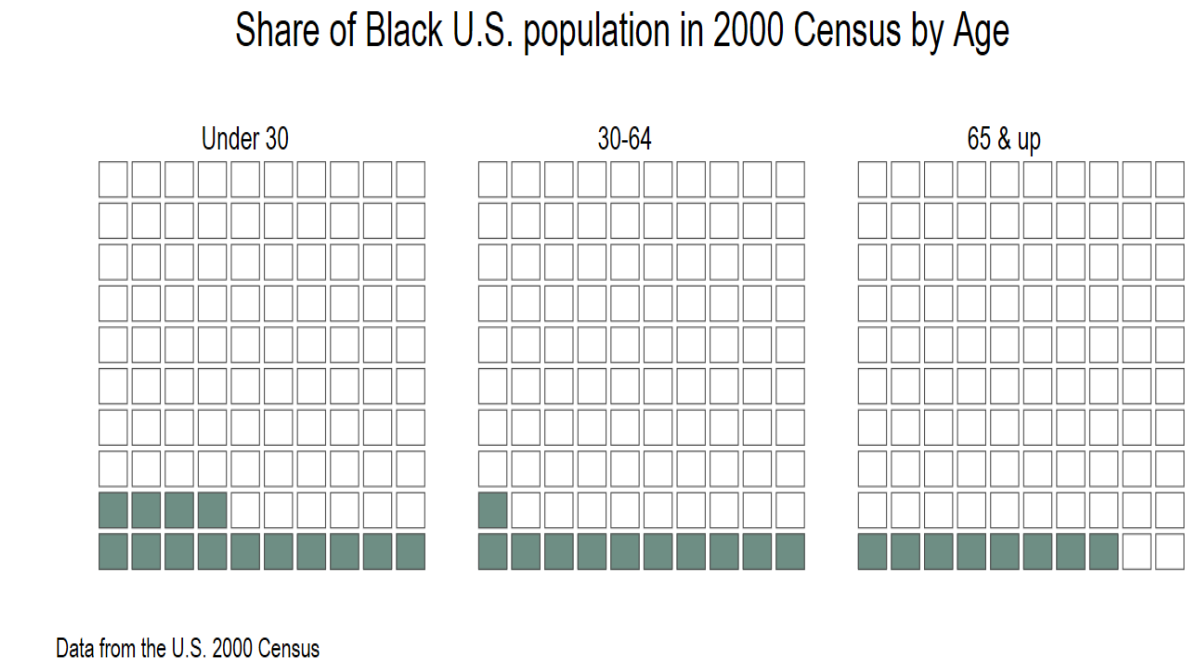
```
      waffle pct_black_small_age, by(small_age_cat) ///
            title("Share of Black U.S. population in 2000 Census by Age",
margin(medlarge)) ///
            note("Data from the U.S. 2000 Census") ///
            markersize(2) colors(teal) emptycolors(white) outlinecolors(gs5)
emptyoutlinecolors(gs5)
```



We can adjust how the grid of waffle charts looks, such as placing them all on a single row:

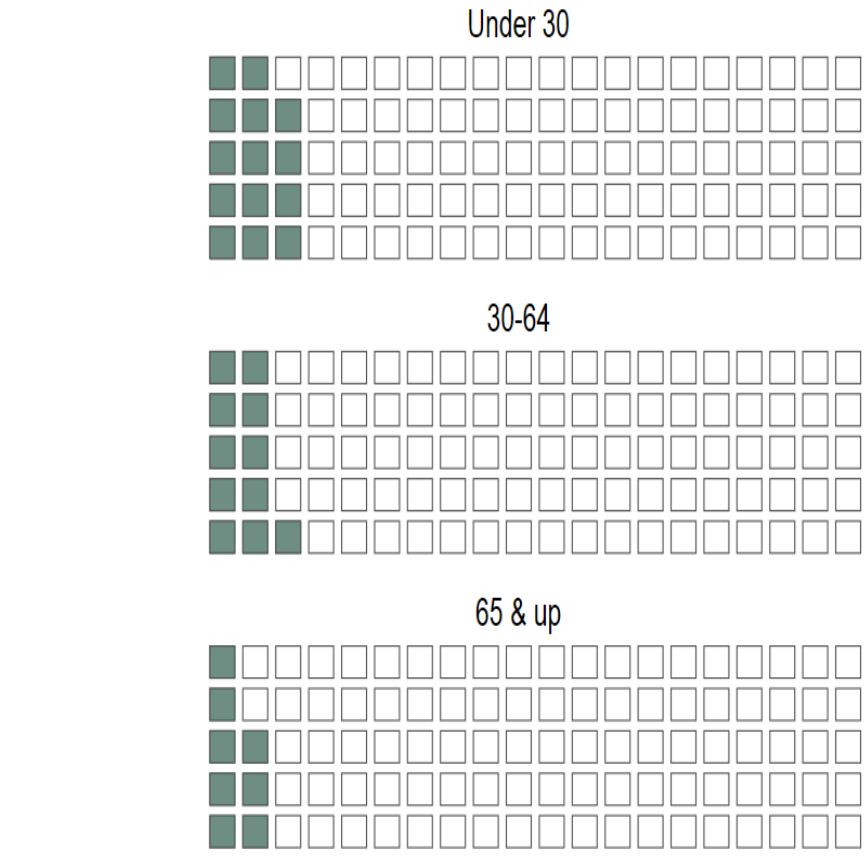
```
      waffle pct_black_small_age, by(small_age_cat, rows(1)) ///
            title("Share of Black U.S. population in 2000 Census by Age",
margin(medlarge)) ///
```

```
            note("Data from the U.S. 2000 Census") ///
            markersize(4) colors(teal) emptycolors(white) outlinecolors(gs5)
emptyoutlinecolors(gs5)
```



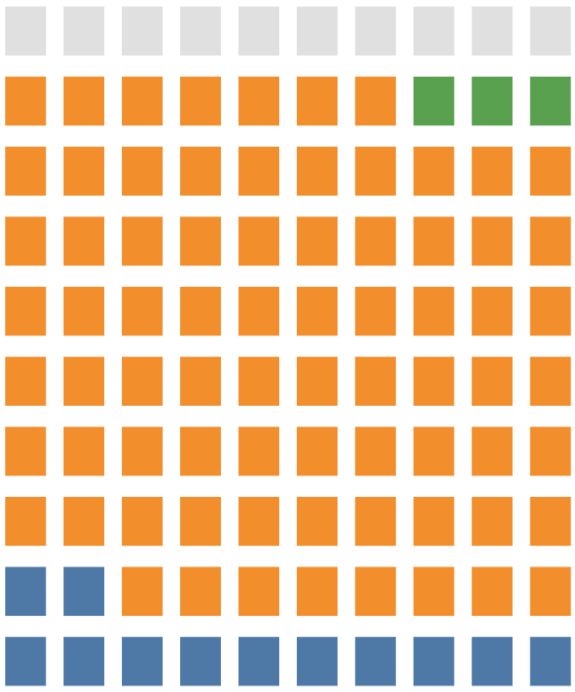
Adjusting the look of the chart grid also works in `wide` mode:

```
      waffle pct_black_small_age, by(small_age_cat, rows(3)) wide ///
            title("Share of Black U.S. population in 2000 Census by Age",
margin(medlarge)) ///
            note("Data from the U.S. 2000 Census") ///
            markersize(3) colors(teal) emptycolors(white) outlinecolors(gs5)
emptyoutlinecolors(gs5)
```



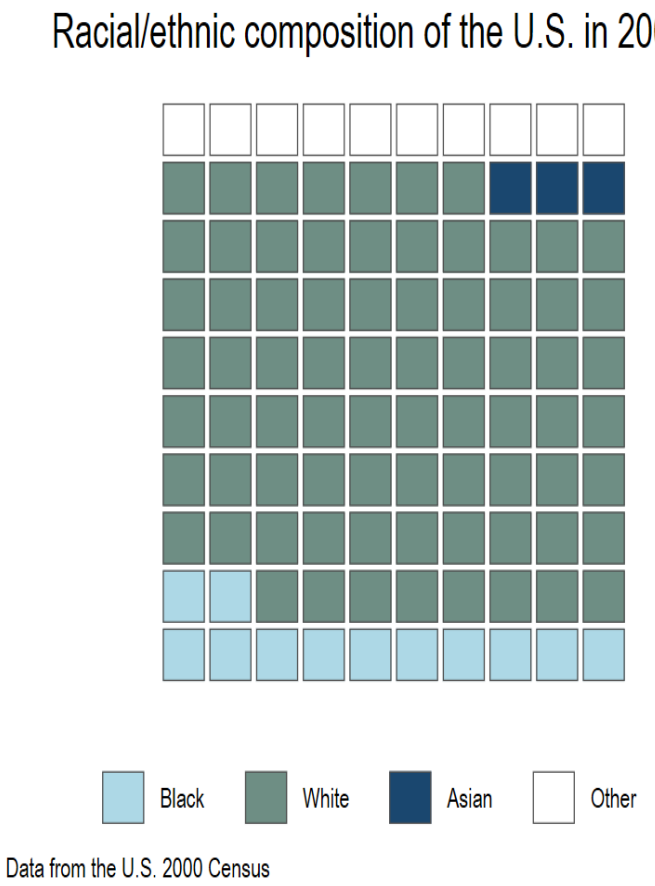
An important extension of waffle charts that has been incorporated into the `waffle` command is the creation of multi-category waffle charts. Here we create a simple waffle chart using each of the race category proportions we created above:

```
waffle pct_tot_black pct_tot_white pct_tot_asian
```



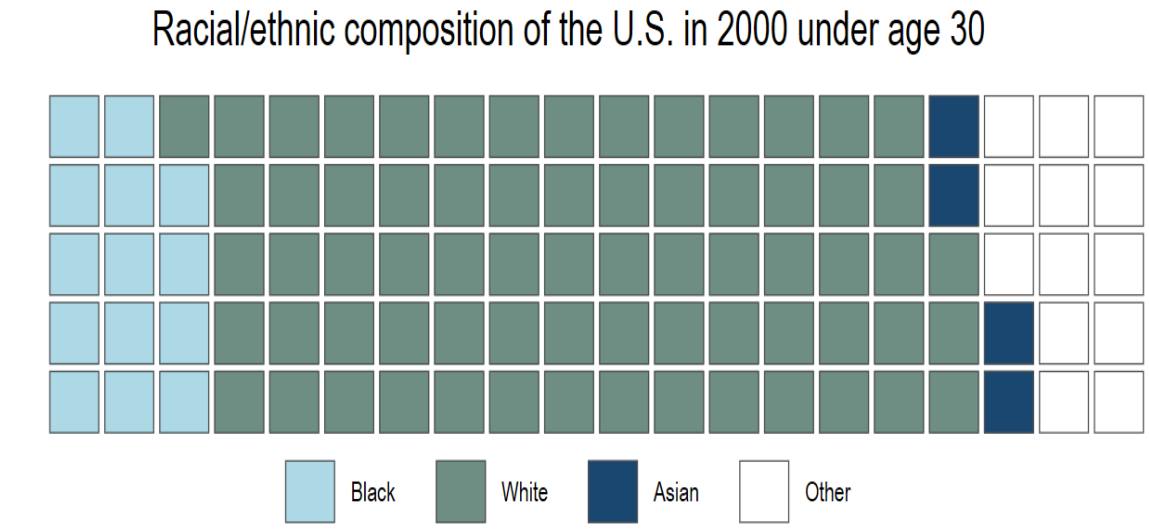
We can add the same graph details as above to this, though we will also add a legend to identify the different categories. Notice that the colors specified in `colors()` define the color in order of your specified *varlist*:

```
      waffle pct_tot_black pct_tot_white pct_tot_asian, ///
            title("Racial/ethnic composition of the U.S. in 2000",
margin(medlarge)) ///
            note("Data from the U.S. 2000 Census") ///
            markersize(5) colors(ltblue teal navy) emptycolors(white)
outlinecolors(gs5) emptyoutlinecolors(gs5) ///
            legend(order(1 "Black" 2 "White" 3 "Asian" 4 "Other") pos(6)
rows(1))
```



Next, we can display this multi-category waffle chart in `wide` mode but also use an `if` statement to only display percentages for a single age category. Notice again that the proportion variables we specify must be unique to the age category intended:

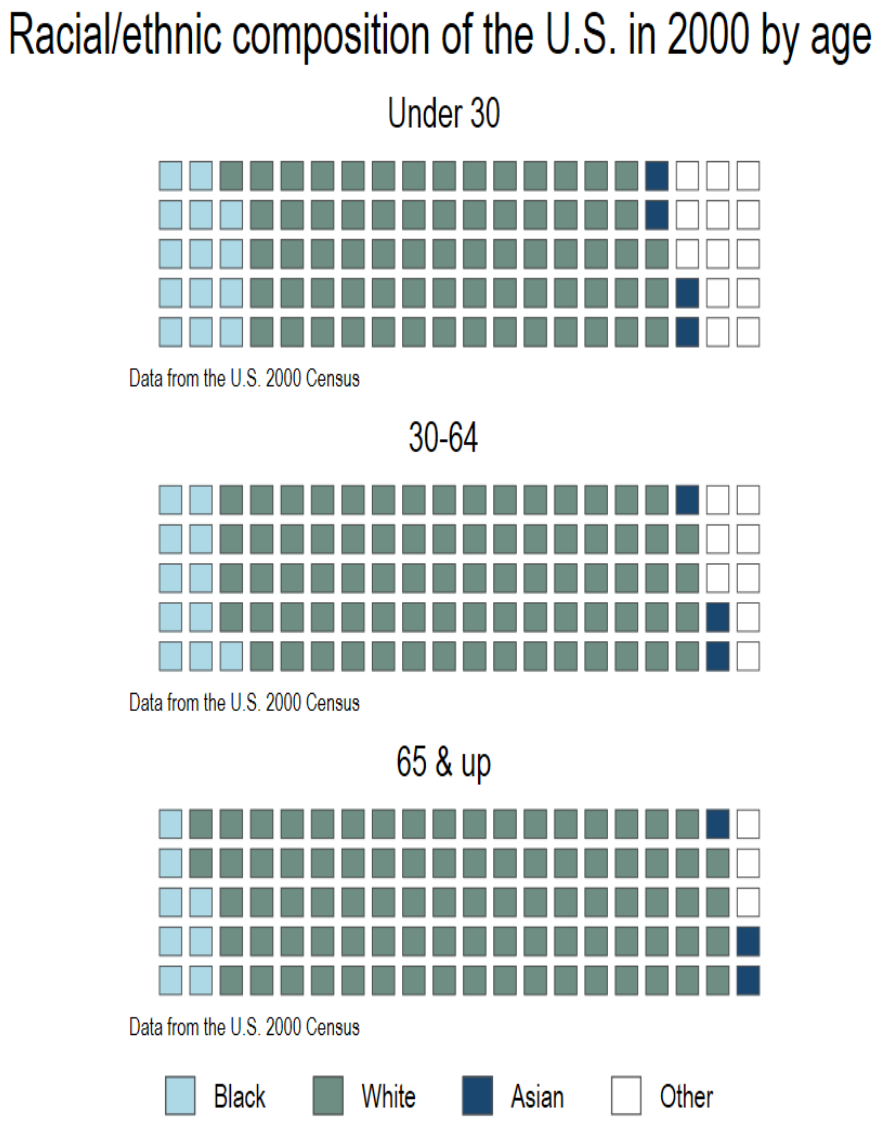
```
      waffle pct_black_small_age pct_white_small_age pct_asian_small_age if
small_age_cat == 1, ///
            title("Racial/ethnic composition of the U.S. in 2000 under age
30", margin(medlarge)) ///
            note("Data from the U.S. 2000 Census") ///
            markersize(6) colors(ltblue teal navy) emptycolors(white)
outlinecolors(gs5) emptyoutlinecolors(gs5) ///
            legend(order(1 "Black" 2 "White" 3 "Asian" 4 "Other") pos(6)
rows(1)) wide
```



Finally, we can create a chart grid of multi-category waffle charts by creating the charts individually and combining them using either `graph combine` or the user-written command `grc1leg`. `grc1leg` allows the charts to share a single legend, which improves readability. While you can make each one individually and name it based on the category name, you can also use local macros to define the titles of each, like so:

```
      preserve
            duplicates drop pct_black_small_age pct_white_small_age
pct_asian_small_age small_age_cat, force
            forvalues i = 1/3 {
                  local l = small_age_cat[`i']
                  local lbe : value label small_age_cat
                  local title : label `lbe' `l'
                  waffle pct_black_small_age pct_white_small_age
pct_asian_small_age if small_age_cat == `i', ///
                  title("`title'", margin(medsmall)) note("Data from the U.S.
2000 Census") ///
                  markersize(3) colors(ltblue teal navy) emptycolors(white)
outlinecolors(gs5) emptyoutlinecolors(gs5) ///
                  legend(order(1 "Black" 2 "White" 3 "Asian" 4 "Other")
pos(6) rows(1)) wide name(waffle_`i', replace)
            }
      restore

      net install grc1leg, replace
      grc1leg waffle_1 waffle_2 waffle_3, ///
            cols(1) imargin(zero) title("Racial/ethnic composition of the
U.S. in 2000 by age")
```



5 Conclusion

Data visualization and description is essential for effective data science and analysis. Waffle charts provide another tool in the kit of analysts for producing clear and appealing graphics to best convey information.

In this article, I presented the `waffle` command, which allows users to create these charts simply by specifying their desired percent or proportion variable and adding elements to the charts using the available options.

6 Acknowledgements

I would like to thank Asjad Naqvi for first introducing the method for making waffle charts in Stata. Without that foundation the creation of the `waffle` command would not have been possible.