Graphs Everyone Should Know and How to Create Them in Stata

Franz Buscha
University of Westminster

A Stata Press Publication
StataCorp LLC
College Station, Texas

Published by Stata Press, 4905 Lakeway Drive, College Station, Texas 77845
Typeset in LaTeX 2$_\varepsilon$
Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

Print ISBN-10: 1-59718-413-6
Print ISBN-13: 978-1-59718-413-7
ePub ISBN-10: 1-59718-414-4
ePub ISBN-13: 978-1-59718-414-4

Library of Congress Control Number: 2024945728

(Pages omitted)

Preface

The art of data visualization has advanced significantly since the turn of the millennium. In particular, accessibility and ease of use have taken center stage. Beautiful, complex, and sophisticated data visualizations are no longer limited to those with highly technical and specialized coding skills. Many software platforms now make it easy for anyone to start creating graphs and visualizing data. Now a much broader user base can find creative ways to reveal patterns and connections in data and, in turn, weave them into their stories and narratives.

Stata exemplifies both ease of use and powerful capabilities in data visualization. Continuous updates to Stata's graphics features, combined with a large and innovative user base, have transformed it from statistical software with basic graphic functions into a true graphics powerhouse. This evolution allows users to create an extensive array of sophisticated and visually compelling graphs with relative ease, leveraging both built-in functionalities and community-contributed commands.

This book aims to unleash that powerhouse and show users the wide range of statistical graphs that can be created with Stata. It is written in an informal style and contains no equations. The aim of the book is not to explain how each graph and statistical command works "under the bonnet", and there is an assumption that users have a basic understanding of statistics. This book is not intended to be linear and read from beginning to end. It is designed to be flipped through casually, during which readers can identify a particular type of graph or a series of graphs they are interested in. They then focus on that content, either reading the text around the graph before applying the relevant code and idea to their own data or reading the entire chapter for a more comprehensive introduction and discussion of that type of graph.

Each graph is directly preceded by the command that generates it. This will help users quickly adapt the code to their own data and create similar-looking graphs. As much as possible, only Stata datasets are used. These come preinstalled with Stata and will already reside on local hard drives. In other instances, examples use online datasets from Stata's official website that are fast to download. Custom or bespoke data sources are avoided because they can be difficult to access or work with. Users will, therefore, notice repetition of certain datasets such as `auto.dta`, `citytemp.dta`, `nlsw88.dta`, or `uslifeexp.dta`. These datasets often have particular variables or data setups that make them useful for teaching purposes, and their repetition throughout the book will help users better understand how different graph types represent the same data.

The book is split into four major parts, of which three are related to statistical concepts around data visualization. The first part looks at graphs that focus on univariate (one variable) data. Here the emphasis is often on presenting distributions of single variables. The second part examines graphs for bivariate (two variables) data. These graphs often examine how two variables are related and how such data are jointly distributed. The third part presents graphs for multivariate (three or more variables) data, which aim to examine how many variables are jointly related. These are often highly specialized graphs that use some trick to insert additional data dimensions. The final part looks at some useful graphic capabilities that users might encounter in data analysis, including how regression results can be easily visualized or how maps can be created.

Each chapter first demonstrates a basic graph, after which a series of important options is highlighted. This book does not aim to present and explain the many options available for each graph command. However, some options matter more than others, and this book tries to highlight how selected options can significantly change a graph. Also note that many graphs have access to generic graph options that change marker colors, line shapes, aspect ratios, axis titles, etc. These options are not explained in detail, because it is assumed that users have some basic knowledge of the graphic capabilities in Stata. However, these options are highlighted and explained when used.

A vital feature of this book is that it goes beyond the standard, official Stata installation. This book aims to represent the wide range of graph commands written by other Stata users and available at the Statistical Software Components Archive. Each chapter will clarify what additional commands or packages need to be installed to be able to follow along with that chapter. All chapters are standalone, and the code will work from top to bottom. The chapters are designed so that readers can execute all the relevant code without worrying about previous or later chapters.

I hope you enjoy this book and keep it close when working with data. Stata learning and teaching have come a long way since I started learning many years ago, and I am glad to provide you with a friendly new resource to augment Stata's extensive documentation. Of course, you are always encouraged to read the relevant help file and examine Stata's manuals in depth for all the nitty-gritty details. However, to rapidly accumulate high-level knowledge of many different processes and abilities, you cannot beat the plethora of Stata Press books now available for users.

Best wishes,
Franz Buscha

(Pages omitted)

Contents

(Pages omitted)

1 Histograms

Advantages

1. Are easy to understand and widely used
2. Are versatile—bin-width choices can generate different visualizations
3. Have fewer parameters to choose from than kernel density plots
4. Make it easy to identify the most frequent bin (tallest)

Disadvantages

1. Use density for default display, which is less intuitive for general users
2. Require careful bin-width selection for proper data analysis
3. Do not provide granularity in small local regions of data
4. Provide minimal statistical summary information such as median, mean, or quartiles

Required community-contributed packages

To install the community-contributed commands that are required to follow along with this chapter completely, type

```
. ssc install eqprhistogram
. ssc install hist3
. ssc install bihist
. ssc install stripplot
```

1.1 Introduction

Histogram plots are the basic "go-to graphs" when plotting univariate distributions for single continuous variables. They are well known, used widely across the corporate and scientific worlds, and taught early at school. Their primary advantage is their simplicity in conveying how one variable is distributed across a range of values. Their use of bars to display the number of observations in a given data range is an effective method to

visually present statistical information. In histograms, both the width and the length of the bar carry numerical information, while in traditional bar charts (which are discussed in chapter 9), the width is irrelevant. This two-dimensional property of histograms (compared with the single dimension of bar charts) is what makes them powerful while also retaining accessibility; they carry more information but not so much to overwhelm general users.

The key to using histograms is bin-width management. Larger bin widths result in bars encapsulating a larger data range, leading to less detail in the graph. Very large bins often result in blocky graphs with little variation and detail. Smaller bins lead to more detail. However, too much detail can result in spiky graphs with large variances and many void areas. One must understand what granularity of data analysis is required and adjust the bin width accordingly.

Another disadvantage of histograms is that they provide minimal statistical summary information. For example, users cannot identify where mean or median values lie on the distribution. In addition, the concept of density may confuse some readers because it refers to relative frequency instead of actual frequency. In basic use cases, one should change the y-axis scale to frequency because it will be more intuitive to nontechnical users.

1.2 A basic histogram

Basic histograms can be created via the `histogram` command in Stata. Only one variable can be specified in this command, so the traditional command for a default histogram is written in the form of `histogram` *varname*. An example is shown in figure 1.1. It uses the city temperature dataset (`citytemp.dta`), which contains temperature information on 956 US cities. The variable `tempjan` is a continuously measured variable of interest.

```
. * Load data
. sysuse citytemp
(City temperature data)
. summarize tempjan
```

Variable	Obs	Mean	Std. dev.	Min	Max
tempjan	954	35.74895	14.18813	2.2	72.6

Creating figure 1.1

```
. * A basic histogram
. histogram tempjan
```

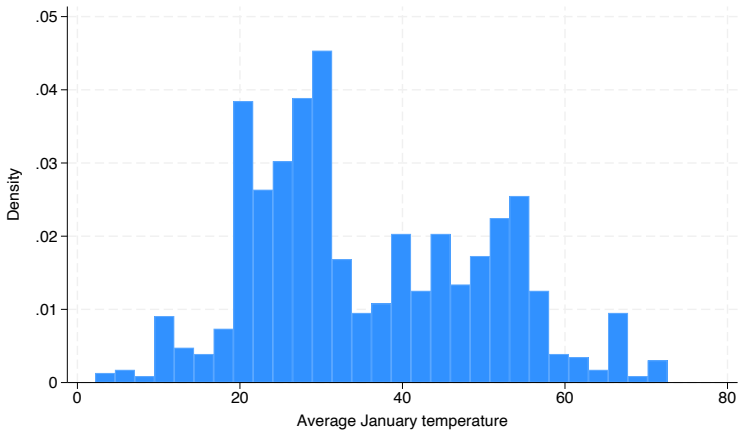

Figure 1.1. A basic histogram

Figure 1.1 presents a basic histogram of average January temperatures of various US cities. The graph suggests that the average temperature in January is possibly bimodal. There are peaks at around 30 degrees and 55 degrees. Few cities experience very cold or very hot days. The y axis is presented as a density by default. Densities are scaled so that the areas of all bins sum to 1. Overall, a user gets an excellent impression of how the numeric temperature values are distributed.

1.2.1 Custom bin widths

The most important options in the `histogram` command relate to adjusting the bin width. This can be done with the `bins()` option, where a numeric value indicates the number of bins on the graph. The `width()` option sets the width of each bin to a numeric value. The `discrete` option is available if data are discrete; this option creates a bin for each unique value in the variable. Examples of each of these options are shown below in figure 1.2. The option `name()` is used to store graphs, and the command `graph combine` is used to merge stored graphs into one graph.

Creating figure 1.2

```
. * Options that change bin sizes
. histogram tempjan, bins(10) name(g1, replace)
. histogram tempjan, bins(50) name(g2, replace)
. histogram tempjan, discrete name(g3, replace)
. histogram tempjan, width(10) name(g4, replace)
. graph combine g1 g3 g2 g4
```

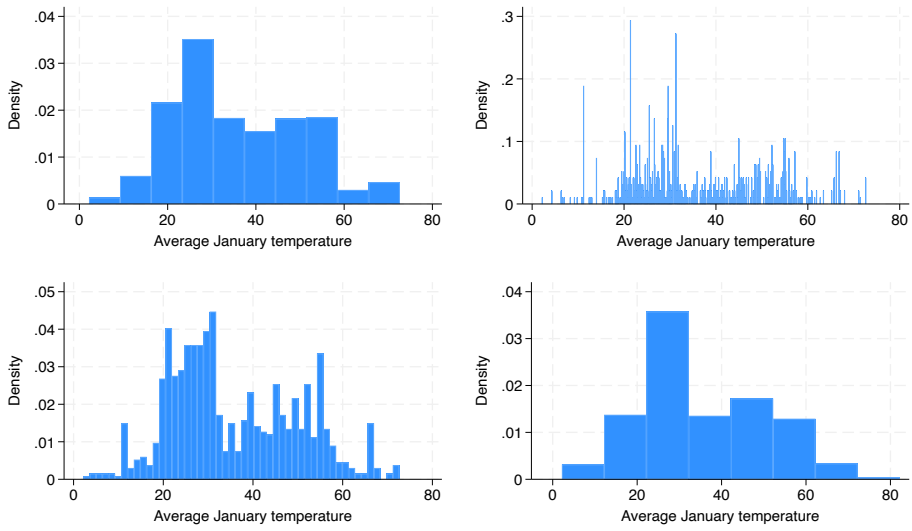

Figure 1.2. Histogram with different bin widths

1.2.2 Changing the y axis

The default y axis of `histogram` is density. Density is a measure of relative frequency, which is frequency divided by the sum of frequencies and then divided by the bin width. This can make interpretation of the y axis difficult for laypeople and those not versed in statistical language. It is possible to transform this axis; for example, multiplying the bin width by the bin height results in proportions, but doing this manually is cumbersome. Luckily, `histogram` has various options that change the default y axis to a different format. The options `density` (default), `fraction`, `frequency`, and `percent` allow users to present density, fractions, frequencies, or percentages on the y axis. Figure 1.3 exemplifies all four options.

Creating figure 1.3

```
. * Options that change the y axis
. histogram tempjan, density name(g1, replace)
. histogram tempjan, fraction name(g2, replace)
. histogram tempjan, frequency name(g3, replace)
. histogram tempjan, percent name(g4, replace)
. graph combine g1 g3 g2 g4
```

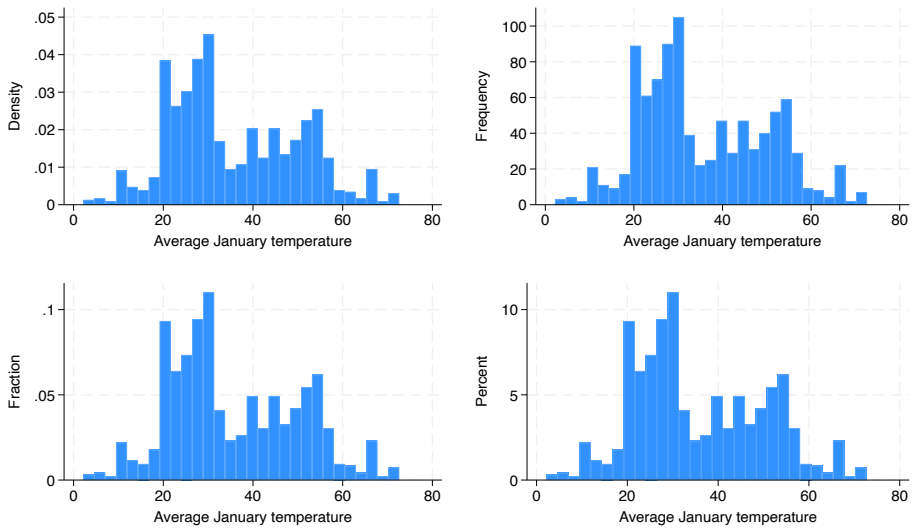

Figure 1.3. Histogram with different y axes

Figure 1.3 highlights the four different y-axis options available to `histogram` users. The graphs look identical, but the y axes contain different values that may be helpful to end users.

1.2.3 Bar labels

Those looking to add more detail to their histograms may want to use the option `addlabels`, which adds height labels to bars and makes individual bar-to-bar comparison substantially easier, especially when bars are of similar sizes. To affect the rendition of the bar labels, the option `addlabopts()` can be used, which takes any options from the *marker_label_options* syntax (see `help marker_label_options`). Figure 1.4 below presents an example of a histogram with marker labels that are angled at 45 degrees, formatted to three decimal places, and very small.

Creating figure 1.4

```
. * Adding bar labels
. histogram tempjan, addlabels      ///
> addlabopts(mlabangle(45) mlabformat(%9.3f) mlabsize(vsmall))
```

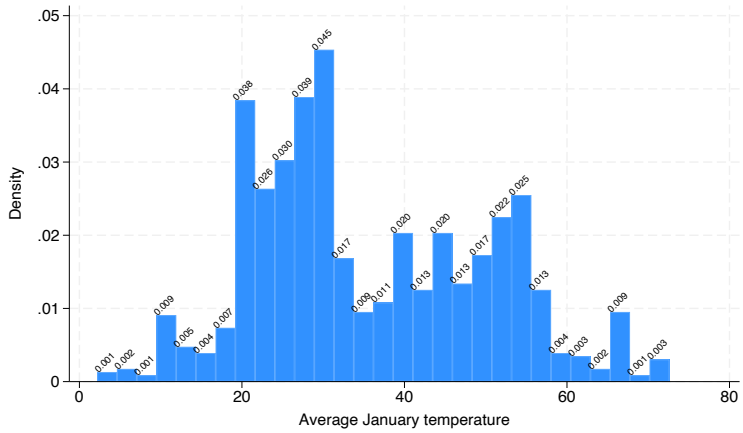

Figure 1.4. Histogram with bar labels

The usefulness of this option will depend on the level of detail required by users. In most cases, histograms are designed to provide general distributional overviews. Individual bar-to-bar comparisons that require bar labels are relatively rare.

1.2.4 Added density plots

Kernel density plots can be added to basic histograms generated with the `histogram` command. The option `normal` adds a normal density to the histogram, while the option `kdensity` adds a kernel density plot to the histogram. The `normal` option is useful for quickly checking whether the histogram mimics a normally distributed variable. The `kdensity` option is useful because kernel density plots provide a smoother visualization of the distribution. Those wishing to have the simplicity of a histogram and the smoothness of a kernel density plot can have both by using this option.

When adding a kernel density plot, the option `kdenopts()` allows for kernel density options to be imposed. Without going into detail, there are many options, of which the most important is `bwidth()`. This option defines the bandwidth of the kernel smoother. One should use bandwidths that are similar to the bin width in the histogram to give some comparability of both plottypes. Figure 1.5 shows an example of this.

Creating figure 1.5

```
. * Adding kernel density plot
. histogram tempjan, width(5) kdensity kdenopts(bwidth(5))
```

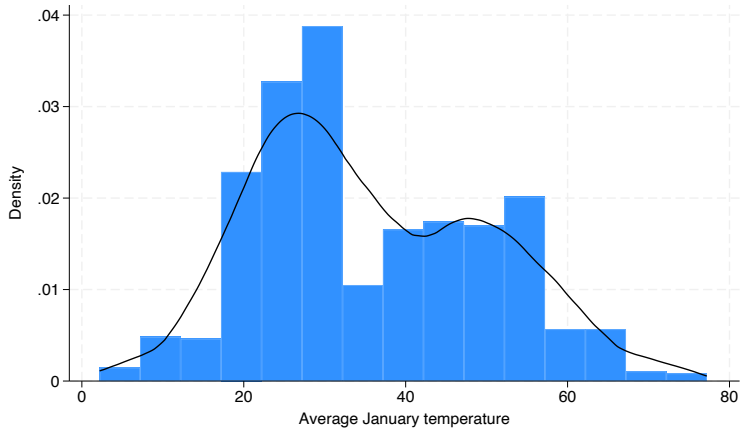

Figure 1.5. Histogram with density plots

1.3 Spike plots

For those wishing to reveal the very fine structure of a data distribution, `spikeplot` offers a convenient alternative to specifying a histogram with a very high bin count (or tiny bin width), for example, `histogram` *varname*`, bins(200)`. A spike plot is essentially a histogram that uses a very high bin count by default, resulting in a highly detailed histogram that reveals the fine structure of an underlying data distribution. An example is shown in figure 1.6.

Creating figure 1.6

```
. * A basic spike plot
. spikeplot tempjan
```

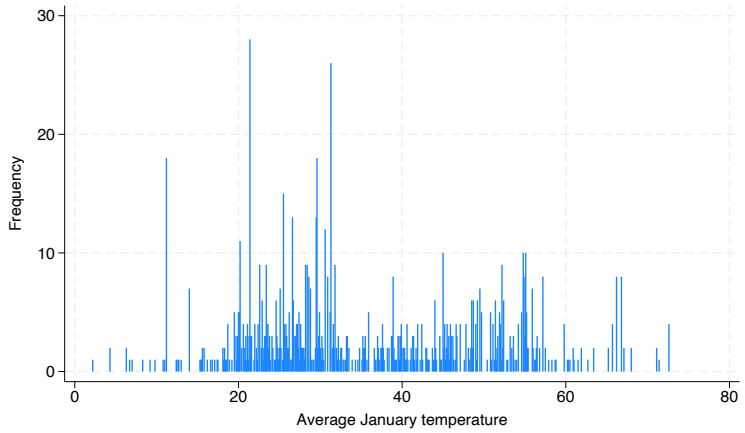

Figure 1.6. A basic spike plot

Figure 1.6 is a default `spikeplot`. It excels at revealing minute data concentrations. However, there are now noticeable gaps in the distribution where no observations occur. Like `histogram`, the `spikeplot` command has a bin-width option that allows users to modify the widths of the spike bars. This can be done via the `round(#)` option, which rounds the variable to the nearest multiple of #. However, unlike `histogram`, the bar widths are unchanged. In other words, the thin nature of the bars in `spikeplot` remains under all circumstances, which can result in spike plots that show gaps between bars, while data in each gap are actually represented in each spike. Figure 1.7 shows an example of this.

Creating figure 1.7

```
. * Spike plot with custom bin width
. spikeplot tempjan, round(1)
```

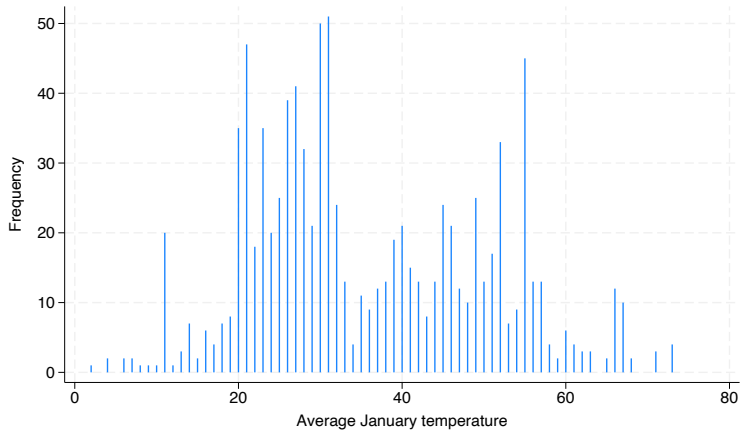

Figure 1.7. Spike plot with custom bin width

Figure 1.7 presents a spike plot that uses integer bins. The gapped nature of this histogram can be useful when seeking to identify which integer values have no temperature observations. However, take care that small gaps are not interpreted as data voids.

It is possible to increase the drawn bar width of spikes in `spikeplot` via the `lwidth()` option. Below is an example that uses a large number in the `round()` option and uses the `fraction` option to change the y axis to fractions. However, one generally should not use `spikeplot` in such a fashion; a normal histogram drawn using the `histogram` command will perform much better unless there is a specific need for blank gaps in a histogram.

Creating figure 1.8

```
. * Spike plot with custom bin width and custom line width
. spikeplot tempjan, round(5) fraction lwidth(*10)
```

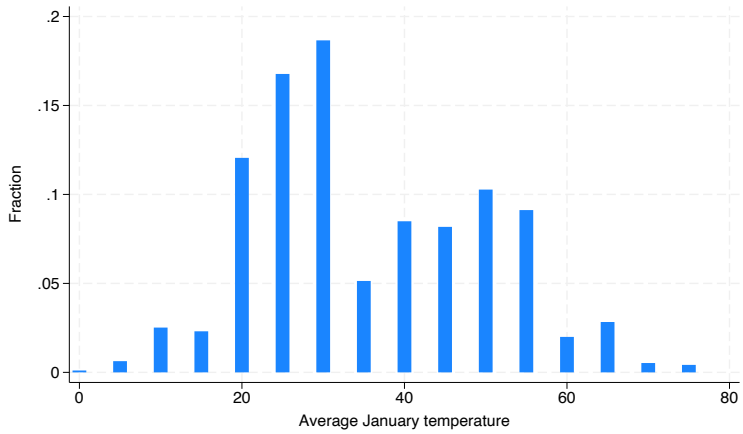

Figure 1.8. Spike plot with custom bin width and custom line width

1.4 Histograms with varying bin widths

Standard histograms have constant bin widths. However, this condition can be relaxed via two community-contributed commands that allow users to modify individual bin widths. The community-contributed command `eqprhistogram` (Cox 2003a) creates equal probability histograms where the widths of the bars vary so that all bars contain equal proportions of the data. This allows users to visualize the percentile distribution in conjunction with the underlying global data distribution. The `bin()` option indicates the number of bins. The default is `bin(8)`, indicating 12.5% of data in each bin. An example with five bins, where each histogram bin captures 20% of the total observations, is shown below in figure 1.9.

Creating figure 1.9

```
. * Equal probability histogram with 5 bins
. eqprhistogram tempjan, bin(5)
```

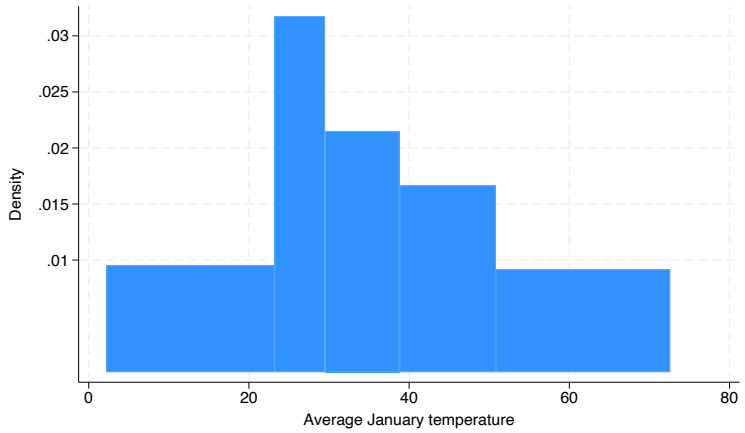

Figure 1.9. Equal probability histogram

The advantage of an equal probability histogram is that it marries distributional analysis with percentile analysis, giving users a better overview of how a distribution is weighted over its observation range. A disadvantage is that areas with little data have large bin widths, denying users details in that particular region. As with all histograms, users should vary the bin size when creating such graphs.

Alternatively, rather than specifying equal probability bins, users can specify custom bins that highlight specific ranges of data. For example, such custom bins might have particular meaning because they belong to an important categorical range group in the underlying continuous data. Such plots can be created with the community-contributed command `hist3` (Kohler and Kuehnel 2000). Unfortunately, this is a relatively old community-contributed command and has not been updated for some time, causing its graphics within Stata to default to a black background with yellow line colors and blue text. However, when these graphs are exported, they default to a black-and-white color scheme. The option `values()` sets the bin cutpoints; note that the start and end cutpoints must be specified. Figure 1.10 presents an example with five custom bins.

Creating figure 1.10

```
. * Histogram with custom bin sizes
. hist3 tempjan, values(2.2, 5, 20, 25, 50, 73)
```

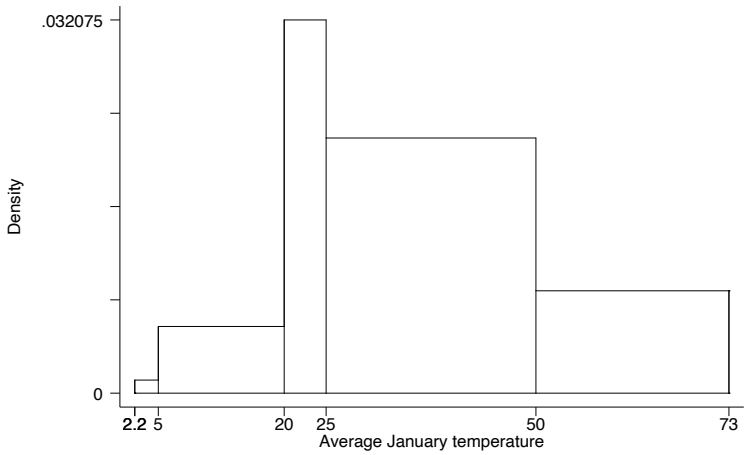

Figure 1.10. Custom bin histogram

Custom bin-width histograms are helpful if users are interested in particular value ranges of the distribution. In this example, a user may be interested in temperatures between 20–25 and 25–50 degrees specifically. However, custom bin-width histograms are generally not often used and require very specific research questions.

1.5 Multiple histograms

Often, users wish to compare multiple distributions. This comparison can be across different continuous variables or subgroups of the same continuous variable. However, histograms are not naturally designed to offer easy comparisons on a single graph, because histograms use significant area shading that causes problems when overlaying multiple histograms. Even when area shading is suppressed, bar lines often continue to overlap, making it hard to identify which histogram belongs to which variable or group. A good rule of thumb is that a kernel density plot should be the first choice when multiple distributions are to be graphed.

Nevertheless, various ways to create multiple histograms exist. The first is via the `by()` option in the official command `histogram`. Users must specify a categorical variable, and Stata will then draw repeated histograms for each subgroup and automatically combine the subgraphs into one graph. An example is shown below in figure 1.11.

Creating figure 1.11

```
. * Multiple histograms with by()
. histogram tempjan, by(region)
```

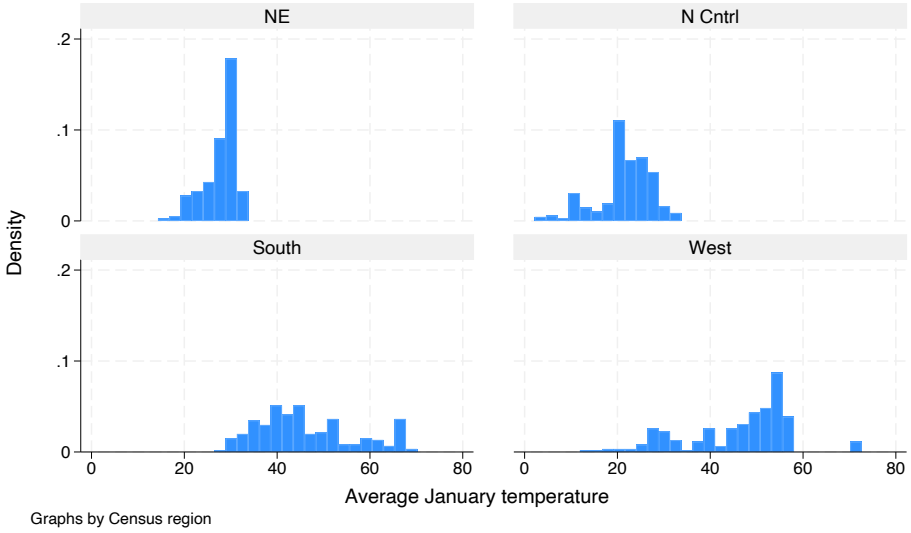

Figure 1.11. Multiple histograms side by side

Figure 1.11 exemplifies use of the `by()` option, which is convenient for a quick overview of how data are distributed across different categorical subgroups. However, side-by-side comparisons are generally useful only for approximate comparison or where there are significant global differences in the distributions. Detailed comparison of local differences is difficult because the graphs are not overlaid.

If detailed local comparisons are required, then the `twoway histogram` command can be used. This command is similar to `histogram` but can be used within the `twoway` wrapper command, which allows multiple Stata graphs to be overlaid on top of each other. The basic command takes the form `twoway (histogram` *var1*`) (histogram` *var2*`)`.

Unfortunately, overlaying multiple `twoway histogram` plots does not automatically change histogram colors. This means that multiple two-way histograms will often visually "block" each other, resulting in a near-useless graph. The trick is to use the `color()` option in each two-way graph and 1) customize histogram colors individually (opposing colors are recommended) and 2) apply opacity via the `%` modifier. A value of `100%` fully hides the background, and a value of `0%` means that the color is fully transparent. An example is shown below in figure 1.12 that uses custom colors from the *colorstyle* list (`help colorstyle`) and 30% opacity. This code uses the `if` qualifier to examine subgroups of the same variable, although different variables could also be used. Note that the legend is not formatted in this graph.

Creating figure 1.12

```
. * Multiple two-way histograms
. twoway (histogram tempjan if region == 1, color(edkblue%30))  ///
> (histogram tempjan if region == 2, color(erose%30))           ///
> (histogram tempjan if region == 3, color(stone%30))           ///
> (histogram tempjan if region == 4, color(emerald%30))
```

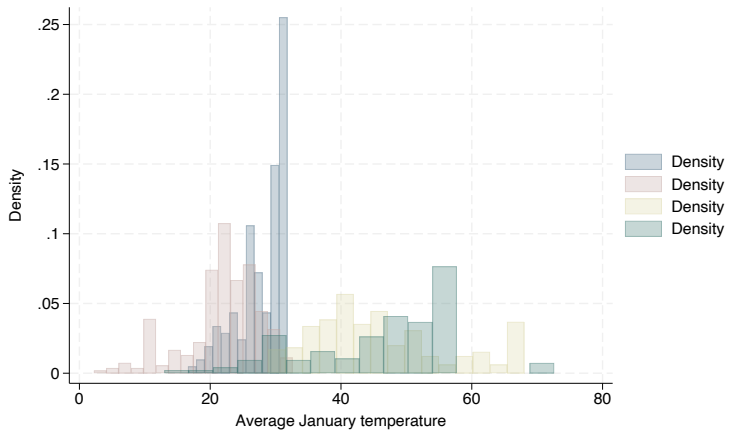

Figure 1.12. Multiple overlaid histograms

1.6 Mirrored histograms

Another approach is to use mirrored histograms. This works only when two histograms must be compared and involves placing one histogram upside down directly below the other histogram. One can manually achieve this by creating two individual histograms and using the `yscale(reverse)` option for one of them. The graphs can then be merged with the `graph combine` command using the `cols(1)` and `xcommon` options, which will result in a mirrorlike merge.

The resulting visual is not perfect, requiring further options to remove margins and align axes. The community-contributed command `bihist` (Nichols 2008) is a more user-friendly method that produces a mirror histogram. The option `by()` must be specified and requires a dichotomous group variable. The `frequency`, `fraction`, and `density` options allow users to choose between different y-axis scales. Figure 1.13 below demonstrates an example with a dichotomous variable and density on the y axis.

```
. * Two histograms with bihist
. generate south = (region==3)
. tabulate south
```

south	Freq.	Percent	Cum.
0	706	73.85	73.85
1	250	26.15	100.00
Total	956	100.00	

Creating figure 1.13

```
. * Mirrored histogram
. bihist tempjan, by(south) density
```

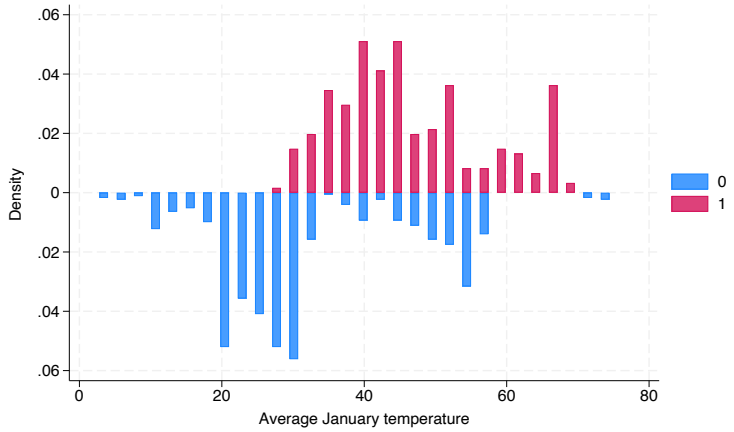

Figure 1.13. Mirrored histogram

Figure 1.13 demonstrates a mirrored histogram. These are particularly useful for comparing two distributions because the distributions do not overlap but still occupy the same graph space and share one x axis, making local comparison fast and easy. A disadvantage of the `bihist` command is that there is no easy way to adjust bin widths. If adjustments are needed, users may need to resort to creating a manual mirrored histogram.

1.7 Ridgeline histograms

If more than two histograms are required, then one can use a ridgeline histogram plot. Rather than merging two histograms onto a common y axis and mirroring one histogram, ridgeline plots graph multiple histograms on separate rows across the y axis. Essentially, the y axis serves the dual purpose of identifying both numeric scale (density, frequency, etc.) and categorical identification. While there is a specific community-contributed command for kernel density ridgeline plots, no such command exists for histograms. However, with some option adjustments, the community-contributed command `stripplot` (Cox 2003d) can be used to create horizontal ridgeline histograms. Alternatively, the official command `dotplot` can be used to create ridgeline plots but only in a vertical orientation. Therefore, `stripplot` is the preferred command to create ridgeline plots.

The `stripplot` command graphs data by placing several marks along a single magnitude axis and can be considered an advanced version of the official command `dotplot`. It can take either multiple variables (for example, `stripplot` *var1 var2*) or one variable when used with the `over()` option (for example, `stripplot` *var1*`, over(`*var2*`)`; this graphs *var1* over *var2* subgroups). Note that the `over()` variable should contain categorical values. The key to using `stripplot` for multiple histograms is to use the `stack`, `width()`, and `msymbol()` options. The `stack` option specifies that data points with identical values are to be stacked, while the `width()` option specifies that values are to be rounded in classes of specified width. The `msymbol(S)` option is used to display square markers. Combining it with the `stack` option results in many stacked squares that are ultimately depicted as vertical bars that identify frequency or density. This is demonstrated in figure 1.14 below. The `msize()` option is used to adjust bar sizes in the graph, although this is only a visual effect.

Creating figure 1.14

```
. * Ridgeline histogram using stripplot command
. stripplot tempjan, over(division) stack msymbol(S) msize(small) width(1)
```

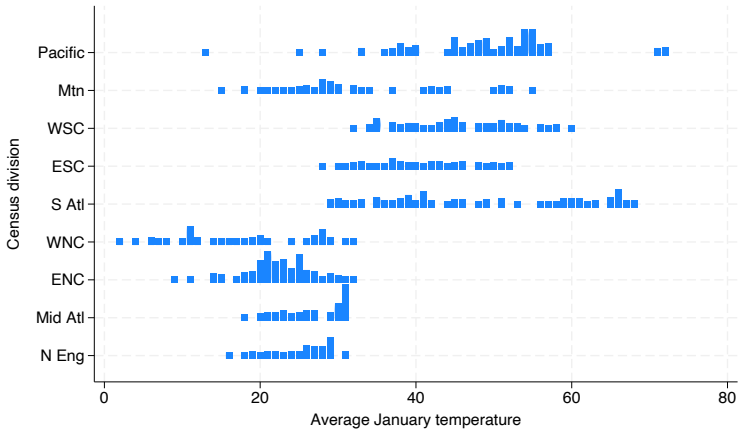

Figure 1.14. Ridgeline histogram using `stripplot`

Figure 1.14 demonstrates how several histograms can be plotted on a single graph. While individual detail in the y axis is lost, global and local distributional comparisons across the different categories are easy and fast. For example, it is easy to identify that the last four regions in the above graph have much colder January temperatures than the first five.

For those unable to download and install community-contributed commands, the official command `dotplot` is an alternative method to produce ridgeline histograms. `dotplot` is based on the idea of showing a point symbol for each data value. The operation of `dotplot` is similar to that of `stripplot`; users specify a continuous variable (*var1*) and use the `over()` option to display a column dot plot for each value of the group variable (*var2*). The `msymbol()` option is used to convert standard circle markers to square markers, and the `msize()` option is used to adjust marker sizes for the graph. The `ny()` option is analogous to the `bins()` option in `histogram`. Figure 1.15 highlights an example with 50 bins.

Creating figure 1.15

```
. * Ridgeline histogram using dotplot command
. dotplot tempjan, over(division) msymbol(S) msize(small) ny(50)
```

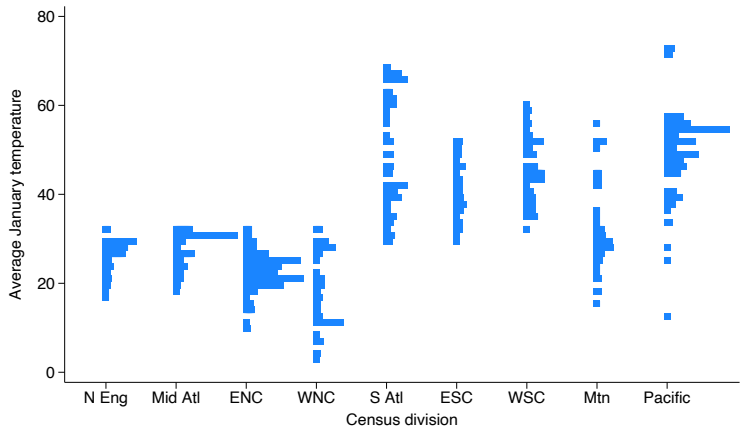

Figure 1.15. Ridgeline histogram using `dotplot`

Figure 1.15 presents an alternative method to create ridgeline histograms. It is similar in style and code to figure 1.14 but can be created using the official command `dotplot`. As mentioned earlier, `dotplot` only creates a vertical graph; it is not possible to rotate the graph and make it horizontal. When using `stripplot`, however, one can simply add the `vertical` option to rotate the graph.

Remember that figures 1.14 and 1.15 are created by careful manipulation of the marker size and bin-width options. Both options must be adjusted synchronously to create an image that has no overlapping bars or wide empty spaces between bars. Some trial and error is required to find the "right" values for both options.