Contents

Abstract

The *stem-and-leaf* plot — a very powerful data visualization tool — first appeared in 1970 in John Tukey's early edition of *Exploratory Data Analysis*. An advantage of the stem-and-leaf plot is in how much more information it can condense in the same space as a histogram, and another is in how easy it is to do by hand without the need for a computer. For this reason, since the 1980s, it has been used as a teaching tool for students as early as the 4^{th} or 5^{th} grade. Yet, in recent years the classic text (semi-graphic) stem-and-leaf plot has seen little use outside of classrooms and textbooks due to the limited visual appeal of the purely text based output and the inadequate handling of more than around 300 values.

Stemgraphic is a solution to these limitations, providing horizontal, vertical or mirrored layouts, sorted in ascending or descending order, with sane default settings for the visuals, legend, median and outliers. *Stemgraphic* also handles very large data sets through scaling, sampling, trimming and other techniques. Inmar's IVANE (Inmar Visualization & Analytics Notebook Environment) platform integrates this python module, providing a superior alternative to histograms. It is used for exploratory data analysis and communication. On the horizon are further improvements to *stemgraphic*: compact multiple plots and interactivity.

Stemgraphic is available on *pypi.python.org* and can be installed using *pip*. Source is available on Github at *https://github.org/fdion/stemgraphic*.

Keywords

Stem-and-leaf, Tukey, semi-graphic, distribution plot, visualization, python, matplotlib, graphic, histogram, bar charts, exploratory data analysis

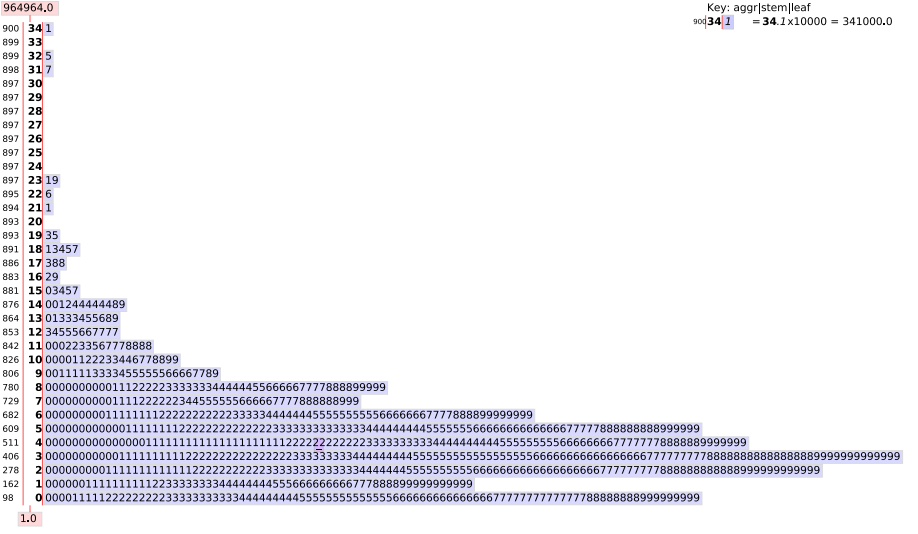


Figure 0.1: *Stem_graphic*: stem-and-leaf plot for 2016 Florida public university salaries [12], a sample of 900 ranging from less than $500 to $341,000, along with outliers (in red boxes) of $1.00 and $964,964 and an estimated median of $42,000 (underlined purple square). An aggregate count is provided to the left of the stem, the difference with the previous line indicates the number of leaves on the right.

Chapter 1

Introduction

The stem-and-leaf plot is one of the most powerful tools *not* found in a data scientist or statistician's toolbox. If we go back in time thirty some years we find the exact opposite. What happened to the stem-and-leaf plot? Finding the answer led me to design and implement an improved graphical version of the stem-and-leaf plot as a python module and integrate it into IVANE (*Inmar's Visualization and Analytics Notebook Environment*), a secure, scalable notebook environment built on top of a Hadoop cluster. To get to that point required many iterations and a solid understanding of the reasons for the rise and fall of the stem-and-leaf plot for exploratory data analysis (EDA).

Starting in the 1970s when John Tukey first published his description of the stem-and-leaf plot, this tool was prominent in the literature on data analysis and a popular tool in the visualization toolbox. Toward the latter part of the 1980s, while the usage in EDA declined, the stem-and-leaf plot was used in college textbooks [11], then in high school curriculums and eventually found its way in some middle and elementary schools [27].

Doing stem-and-leaf plots by hand might be fine for school assignments, but for real world exploratory data analysis and visualization, a way to automate some of the work is needed. From a single Fortran program, the field has grown to many statistical software packages that include stem-and-leaf plots. Some are fairly close to Tukey's design; a few add basic adjustment of font color or size [20], while others cover only a

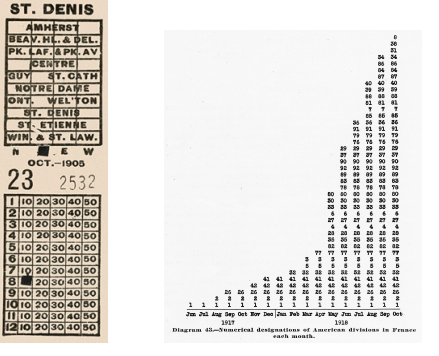
bare minimum of functionality. Still, the rise in popularity of infographics and information visualizations, along with the ever increasing size of datasets, have relegated the seemingly boring stem-and-leaf plot to the past, until now.

Stemgraphic is a graphical stem-and-leaf plot with sane default settings for visuals, scale, legend, median and outliers. It has proven to be very beneficial for EDA work, replacing the histogram in most cases. It also has been well received in communication work in team meetings as well as in printed form.

Before getting into the design and implementation of *stemgraphic*, chapter 2 will cover background information on the stem-and-leaf plot and some lesser known variations, followed by a deeper look at the use, strengths and limitations of the original design in the early period (2.1) and more recent period (2.2). The design and implementation (chapter 3) will cover previous implementations of the stem-and-leaf plot, as well as the design decisions, core logic and implementation details of *stem_text* (3.1) for the scale, large datasets and command line access. The chapter will continue with *stem_graphic* (3.2) detailing mirroring, sorting and inverted axes, legend design and placement considerations, secondary plots, persistence and finally, the selection of sane defaults. Chapter 4 provides qualitative evaluations of *stemgraphic* for smaller datasets compared to the corpus of published stem-and-leaf plots (4.1) and both qualitative and quantitative evaluations on larger datasets compared to text stem-and-leaf plot and histograms (4.2) and comparison of usability of the default settings in daily EDA tasks (4.3). We conclude this paper in Chapter 5, where we highlight the successes and best features of *stemgraphic*. We also cover the current limitations and provide a glimpse at future research and future areas of improvements for multivariate dataframes, compact layouts and interactivity.

Chapter 2

Background



(a) Bus transfer (1905) (b) US Divisions (1919)

Figure 2.1: Tables and charts sometimes misidentified as stem-and-leaf plots

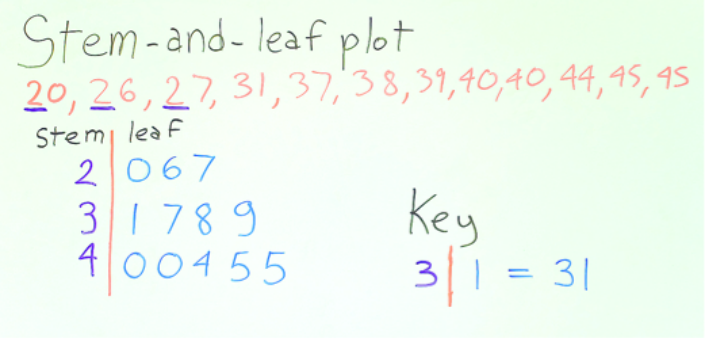


Figure 2.2: Stem-and-leaf made by hand on a whiteboard, in classrooms since the late 1980s, part of the education of millennials.

As we are considering the stem-and-leaf plot created by John Tukey, we need to review his definition of a few terms (from the glossary of his 1977 book *Exploratory Data Analysis* [34]):

"**stem-and-leaf display**: a generalized two digit display, in which the lefthand portion of the values displayed is given by a stem value, while the righthand portion makes up a leaf (leaves follow stem juxtaposed if uniformally one digit, follow separated by commas if some or all involve 2 or more digits)

squeezed stem-and-leaf display: a display with 5 stems (usually labelled '*', 'T', 'F', 'S' and '.') for each stem number.

stretched stem-and-leaf display: a stem-and-leaf display, with 2 stems per value, these being distinguished with '*' and '.'"

Nowadays, the stem-and-leaf display is better known as the stem and leaf plot or stemplot, although the latter can be confusing as there are stemplots [22] that are not stem-and-leaf plots. There are also many unrelated tables (such as the bus transfer [31] in *Figure 2.1a*, train schedules and other timetables) and semi-graphic charts (*Figure 2.1b* [2]) that are mistakenly identified as stem-and-leaf plots, but fail to meet Tukey's definition and/or do not serve as indicators of distributions.

The basic form of the stem-and-leaf display is simple enough for chil-

dren of primary school age to grasp (*Figure 2.2*). There is a large segment of population (a significant portion of 83 million millennials just in the US [37]) that has been exposed to the basic stem-and-leaf plot at some point in school, and in some cases, multiple times. *Figure 2.3* shows the querying on Google of the stem-and-leaf plot and histograms (for reference), both reaching a peak at the beginning of every school semester (September and February) and reaching a low in December and July. It is highly likely that much of the activity is due to some exposure to these distribution plots in an academic context. Of course, previous exposure does not equate to perfect understanding of visualization, as Kaplan *et al.* [18] identified several misconceptions of histograms in students both before and after having received instructions on them. In that regard, these misconceptions are probably not a differentiator between the stem-and-leaf and histogram and further research is needed.

Regarding the three definitions at the beginning of Chapter 2, we have covered the first. As for the other two variations on the basic plot (squeezed and stretched versions), they add some level of complexity both in implementation and in perception and are not as well known or understood as the basic version. Even with these reservations, the stretched plot was added, and section 3.1 (under *Number of stems and leaves*) explains the reason why. Let us now have a look at how the stem-and-leaf plot has done over the last 46 years.

2.1 The paper era

Possibly drawing some inspiration from John Dudley's tallying table [10], Tukey first published his idea for the stem-and-leaf plot in 1970 [33]. Although computers had been invented, they were not yet widely available. Besides, no software implementation of the plot existed in those early days. Up until 1972, the stem-and-leaf plot was probably unknown to people outside of Tukey's classrooms, at which point it gained a larger audience in his paper Some Graphic and Semi-Graphic Displays [35] where he stated: "the editorial principle that nothing should be given both graphically and in tabular form has to become unacceptable".

He further suggested that these types of plots would be quite effective at communicating the coarse and the fine in one display. This was a message full of promise that would be validated over the following years

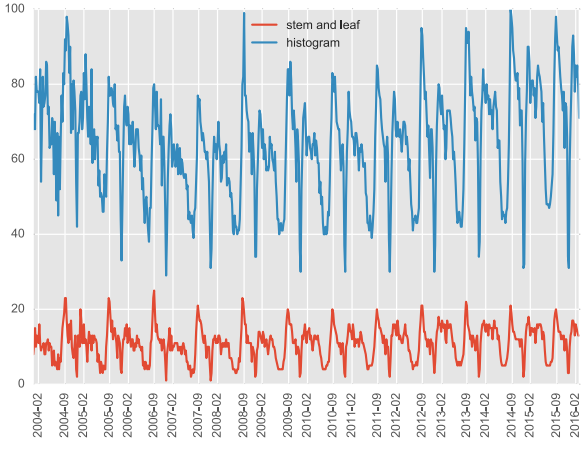


Figure 2.3: Google Trends search for “stem and leaf” and “histogram” in percentage. Cycle in both aligns to a peak in September and February and a dip in December and July.

as usage increased. In 1975, Hoaglin and Wasserman wrote software [15] to generate a text stem-and-leaf plot with a good scale calculation. While their approach proved good enough in many of IVANE’s use cases, using functional and qualitative testing over a wide variety of actual datasets provided *stemgraphic* with improvements to both the algorithm and the results.

In 1977, Tukey published the seminal work *Exploratory Data Analysis* [34]. The stem-and-leaf was not only introduced in the first chapter, it was prominently displayed on the cover of the book. This book covers in great detail the motivation behind it. It also details multiple variations, including categorical plots using an alphanumeric translation code. From that moment on, we see the stem-and-leaf plot appearing in more publications, in the US and abroad. As an example, Transport Canada [21]

started using the stem-and-leaf plot in publications in 1978.

For several years after that, key publications continued to discuss the stem-and-leaf plot as an important tool in exploratory data analysis and information visualization. Specifically, in 1983 we find extensive coverage of the subject throughout the book *Understanding Robust and Exploratory Data Analysis* [14] by Hoaglin, Mosteller and Tukey, where we can read on p.29: "Many of the chapters in this volume use the stem-and-leaf display at several stages in understanding the structure of a batch of data or a batch of residuals. As we encounter these applications of the stem-and-leaf display, its utility and its ability to illuminate will become more evident".

The book also brings up the problem of sorting and of performance, something that has to be taken into consideration for larger datasets, as we will quantify in Chapter 4. Also published in 1983, *Graphical Methods for Data Analysis* [4] by Chambers, Cleveland *et al.* explains both single and multivariate use of the stem-and-leaf. Wrapping up that year, Edward Tufte's *The Visual Display of Quantitative Information* [32] also explored the concept of the stem-and-leaf plot, using it as an example for making every graphical element effective. Specifically, the digit as a graphical mark was key in differentiating the stem-and-leaf with the histogram.

In 1985, we can see signs of the stem-and-leaf plot losing ground as a popular distribution plot. That year, *The Elements of Graphing Data* by W. S. Cleveland [5] makes absolutely no mention of the stem-and-leaf plot. In fact, just two years after *Graphical Methods for Data Analysis*, he completely eschews any text or semi-graphic displays. This will be touched on in more detail in the next section. Still, this was not the end for the stem-and-leaf plot in EDA. In *Graphical Exploratory Data Analysis*, du Toit *et al.* [6] introduce the plot and two programs to generate basic and stretched stem-and-leaf plots.

Finally, in *Mathematical Statistics and Data Analysis* (1988) John Rice [28] introduces the stem-and-leaf plot along with the histogram and density curves. He does mention a problem with the stem-and-leaf plot: "Straightforward stem-and-leaf plots do no work well for data that range over several order of magnitude. In such a situation, it is better to make the stem-and-leaf plot of the logarithms of the data", something that makes the plot less readable to many people. An alternative to this is to provide a secondary plot to retain the fine, yet provide the bigger picture, as we will cover in section 3.2. Wrapping up 1988, W. S. Cleveland edited

the fifth volume of *The Collected Works of John Tukey* [36], which covers graphics related papers from 1965 to 1985, including the stem-and-leaf plot.

From this point forward, the publications covering the stem-and-leaf plot are mostly basic statistics handbooks, from *Complete Business Statistics* [1] in 1989, all the way to *Introduction to the Practice of Statistics, 8th edition* [24] in 2015. One notable exception is Harris' illustrated reference from 1996, *Information Graphics* [13], where he devotes 2 pages to the stem-and-leaf-plot, scaling and splits, along with an example of a grid for generating hand-written charts.

In parallel to this, a push to introduce the stem-and-leaf plot to an ever-younger audience appeared. One of the first papers on the subject, *Stem-and-Leaf Plots in the Primary Grades* was published in 1989 [26]. In it Mendoza and Dunkels make the case that the stem-and-leaf plot can be used as a major tool in young students' educational development. This is not unusual since the plot was designed initially for pen and paper. At any rate, the idea grew and the stem-and-leaf plot is now included in many STEM programs, as early as the 4th grade [30].

2.2 The big data era

The words "big data" have a strong connotation for people doing information visualization. When it comes to the original stem-and-leaf plot however, we are talking about anything over 300 or so data points [15]. As long as everything was done by hand, these kinds of datasets were not common or popular. But as personal computers spread in the marketplace in the mid 80s, things changed radically, spelling trouble for the stem-and-leaf plot in two major ways:

1) It was now easy to generate large sets of random numbers, to store thousands of data points on a floppy disk. Stem-and-leaf plots could not display all of this data at once.

2) Inexpensive computers now provided high-resolution graphics while the text console standardized on 80 columns. Graphics were perceived as state-of-the-art, while green screens were not.

In the previous section, we highlighted a major advantage of the stem-and-leaf plot, providing a coarse and fine level of detail all at once. On this subject, in *La Sémiologie Graphique*, Jacques Bertin [3] compared a

good graphic to the representation of a tree: it is not sufficient for it to show the leaves of a tree; it has to show the limbs and the tree as a whole. That is the expectation we have of modern EDA tools, yet we rarely get to experience the big picture and the fine details all at once. We end up with either a fuzzy big picture, a form of aggregate, or fine details of a very focused subset of our data. If the stem-and-leaf plot can properly handle large data sets, we can once again get the big picture and the fine details together. It is important to note that the R language does provide a *stem()* function that can handle large datasets, limited only by the memory of the computer executing the function. However, it does so by combining aggregation with partial leaves. We will cover this option and its limitations in more detail in section 3.1 (under Number of Data Points). Regarding the issue of graphic appeal, a direct transposition of the text stem-and-leaf plot would not be sufficient to achieve a proper solution. Some flexibility in regards to the various graphical attributes of the plot is the norm in popular graphing systems and toolkits [17][23][38][39] and this has to be part of the design process and solution.

We have now identified two more design challenges, but we are not done yet. The ease of use also has to be addressed, particularly when it comes to minimizing the number of steps and of options that have to be specified to obtain usable results.

In *Visualize this* (2011), Nathan Yau [40] describes how to generate a stem-and-leaf plot with R using the previously mentioned *stem()* function and how to manually style it by a copy-paste operation into something like a word processor (spreadsheet based variants are also possible [16]). While the subheading of "Old School Distribution" hints at how the plot is perceived, the author points out that "there are easier and faster ways to look at distributions today". R's ggplot2 [39] *qplot()* and *geom_histogram()* and Python's Pandas [23] *hist()* and Seaborn [38] *distplot()* are good examples of easy and fast ways to display distribution plots, and should be a benchmark for UX, even though the resulting plots can be inconsistent or even unhelpful at times. A properly designed graphical stem-and-leaf plot with good default settings should be as easy and fast to use as these tools, but provide more insight than these histogram plots in many situations.

Chapter 3

Design and Implementation

Although the initial goal was to have a graphical representation of a stem-and-leaf display integrated into IVANE, the first step was to create a text based (*stem_text*, *stem_data*) version that would provide the right information to the function rendering the graphical display (*stem_graphic*). These were then combined into a Python module along with a text output and a command line wrapper. This allowed quick use of both the conventional text stem-and-leaf plot and the more visually appealing graphical stem-and-leaf plots. It also enabled various forms of testing, both functional and qualitative.

3.1 Stem_text

As noted earlier, the first published implementation of stem-and-leaf plots was Hoaglin/Wasserman in Fortran [15]. It is complete as far as text functionality is concerned, but it is now 41 years old, and would be difficult to use as is. There are also 49 implementations of a basic stem-and-leaf algorithm on Rosetta Code [29], in languages ranging from ACL2 to zkl. They are mostly hard-coded implementations assuming many things about the dataset, such as supporting only positive integers. A proper solution needs to support positive and negative decimal numbers at a minimum (Tukey also did *categorical* stem-and-leaf plots). The above functionality

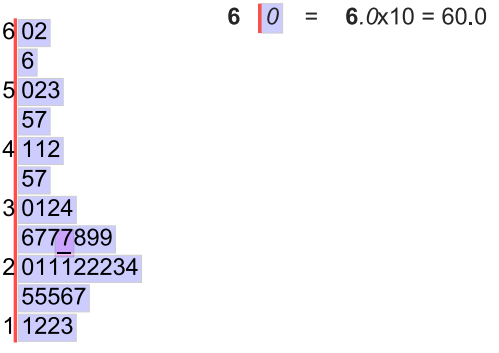


Figure 3.1: Stem-and-leaf based on data from Aczel showing a stretched version, 2 lines per stem. Generated by default with *stem_graphic* (with *aggregation=False*, *alpha=0.2* and font and style changes by using *import seaborn*).

is a given including the ability to display the optimum number of stems and leaves through a proper scaling algorithm. One entry found under the R language is one line, *stem(x)* due to the fact that it is incorporated into the language (*stem()* is an R wrapper to *C_StemLeaf*) and is the better solution of the 49. It has the advantage of already being integrated in IVANE, but has several limitations that prevent us from using it as is. The solution needs to be able to handle more than a few hundred values as IVANE is used on a daily basis to explore commercial transaction datasets with tens or hundreds of millions of data points.

Number of stems and leaves

The first problem is identifying how many stems and leaves to display. Regarding the stems, several papers have been written on bin size for histograms and stem-and-leaf displays. Some of the earliest approaches are Dixon and Kronmal's [8] and Velleman's [15]. In that regard, we will not be proposing a new equation to replace them. Instead we are planning to use them together with the following logic – given n data points to display, we will try to aim for a number of stems around *lines* using Dixon and Kronmal [8] for a small number of data points and Velleman [15] for larger sets.

```
if n <= 300:
    # Dixon
    lines = floor(10 * log(n,10))
else:
    # Velleman
    lines = floor(2 * sqrt(n))
```

When n is very small (about 20 or so), the above does not provide an optimum display and we will want to further adjust our scale, as we will cover presently. Once we have a target number of stems, we follow the approach of Hoaglin and Wasserman [15]:

```
spread = xmax - xmin
scale_factor = pow(10, ceil(log(spread/lines,10)))
check = xmax/scale\_factor - xmin/scale_factor+1
```

The value of *check* is expected to be less than *lines*. After that, the divergence from the original scaling algorithm is as follows:

```
if check > lines:
    scale_factor *= 10
elif (check < 7 and n >= 45) or check < 3:
    scale_factor /= 10
elif floor(check)*2 <= lines and split is None:
    split=5
```

If we are above our calculated threshold number of lines, we multiply the scale by 10. Conversely, if we have very few lines (7 or less) and have a sufficient number (45) of data points to display or we only have fewer than 3 stems, we divide the scale by 10. This heuristic was derived experimentally, through repeated functional test cycles. Finally, if we are in neither of these categories, we will increase our number of lines if we calculated that less than half the optimum number of lines will be generated using the current scale. We do this by introducing a break in the

leaves at 5, meaning that leaves 0 to 4 will be on a line, and 5 to 9 on the next (Figure 3.1), implementing the stretched stem-and-leaf plot. This is used mostly for EDA work, as there is some level of confusion that can arise when reading those.[1] The squeezed stem-and-leaf plot with even smaller breaks has not been implemented at this time, as more understanding of the perception on these graphs is needed.

Number of data points

The next issue to address is large or very large data sets. We know from Tukey that we can display 300 or so values in a text plot, and by our own experimentation 900-1000 seems to be an equivalent value for a graphical plot (Figure 1), although a vertical stem-and-leaf plot in IVANE or Jupyter [19] notebook can easily display over 5000 readable points, with some scrolling. How do we handle data sets larger than those values? One approach (taken by R's stem) is to display only the first *n* leaves of a given stem, then count all the others and append that count. In practice, we end up with a lot of 0s followed by a count. The plot ends up representing *neither a stem-and-leaf plot nor a histogram.* The other option is to reduce the set before we try to figure out the details of how to display the data.

For a text plot, this implementation provides *by default* the behavior of sampling 300 data points (900 for a graphical plot with *stem_graphic*) randomly without replacement using Pandas *DataFrame.sample.*[2] As a consequence, the plot is much less impacted by outliers, providing more detail about the distribution than would otherwise be possible.

The solution had room for improvement. Many times a day, the sample turned out a plot with more than one third of the data points on a single stem, due to a combination of factors. Usually, it was sufficient to plot the data once more, forcing another sampling. Other times, plotting a second time with a slightly reduced number of samples provided a more usable plot. This logic was added to the current version of the sampling code where a second try was generated automatically if more than one

[1] Preliminary studies show that the 5 to 9 break is sometimes associated with the wrong leaf, and adding '*' and '.' as suggested by Tukey does not help avoid this confusion.

[2] Assuming we are passing a Pandas dataframe. For Dask, this would convert number of points into percentage and add a *compute()* call.

third of the values ended up on a single stem, and the best of the two plot was selected.

Command line tool

In the data science department at Inmar, the notebook environment (IVANE) is used extensively. As such, no initial thought was given to making the stem-and-leaf plot functionality easily available from the command line, apart from launching the python interpreter and importing the module. User feedback however, particularly outside data science suggested that a command line tool wrapper to the python module would be beneficial. This tool is automatically added to the bin directory for the system when the module is installed using *pip3 install stemgraphic* or *python3 setup.py install*.

A session at the command line might combine *ls* to identify the file, *head* to get the column names and *stem* to generate a text or even saving a graphic stem-and-leaf plot to a png file:

```
$ ls *.csv
$ head file.csv
$ stem file.csv --c 3 --o plot.png
```

3.2 Stem_graphic

With a solid text stem-and-leaf functionality, the *stem_graphic* work is mostly about the graphical representation of the data itself, with additional design and implementation required for the legend, the secondary plot, the persistence and sane defaults. For this first version, matplotlib [17] was selected as the base graphic module for the implementation, due to its proven flexibility and stability, reducing risk and time to implement. The downside meant that an interactive version would probably require a complete re-implementation.

Focusing solely on the basic graphical representation, the implementation is quite straightforward. A *simplified* version of the python code of the main loop:

```
for cnt, item in enumerate(rows):
    stem, leaf = item.split(PIPE)
    tot += int(len(leaf.strip()))
    if tot > n/2 and med is None:
```

```
med = abs(tot - n / 2 - len(leaf.strip()))
ax.text(2.1 + med / 2.2, cnt, ' ', fontsize=ft_sl,
        bbox={'facecolor':median_color,'alpha':alpha,'pad':2})
ax.text(-0.5, cnt + 0.5, tot, fontsize=ft_t, va='center')
ax.text(1, cnt + 0.5, stem, fontsize=ft_sl, va='center')
ax.text(2, cnt + 0.5, leaf[::-1] if mirror else leaf,
        fontsize=ft_sl,
        bbox={'facecolor':facecolor, 'alpha':alpha, 'pad':2})
```

After calculating the size of the graphical canvas based on the number of stems (rows) and the length of the longest list of leaves for a stem (the data calculated by *stem_text*, using *len(max(rows, key=len))*), *stem_graphic* iterates through each row, increments the aggregate (sum of number of values to that point), calculates and display the median if it is on that row, and positions the text for the aggregate, the stem, and the leaves (all the leaves of one stem as one text glyph, *for speed*).

Once we have iterated through all the values, we add the vertical bars, the outliers and the legend.

Mirroring, sorting and inverted axes

One of the advantages of providing a graphical stem-and-leaf plot is that it is possible to display it in the standard vertical form, or in a horizontal form, as is typically the case of histograms.

Flip_axes is set to False, so that the display will be wider than tall in a majority of cases as suggested by Edward Tufte regarding the shape of graphics [32]. *Mirror* is also set to False, while *ascending* is set to True, providing a familiar ascending axis orientation from left to right and from bottom to top. On the implementation side the main challenge was in making sure that all possible combinations work as intended, particularly in regards to alignments and rotations (see Figure 3.2 for a small sample of combinations).

Legend

The first graphical version did not include any legend except for a scale statement. This worked well only with people very familiar with the stem-and-leaf plot. The second version included a black and white text key in the form:

32|6 = 32.6x10000 = 326000

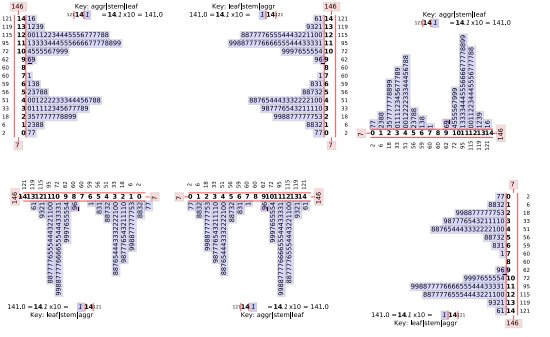


Figure 3.2: *Stem_graphic* output showing various True/False ascending, mirror and flip_axes combinations (data from Rosetta Code).

The final iteration of the legend added visual cues into the legend (color, bold, italic) and an optional unit, improving readability:

32|*6* = **32**.*6* x 10000 = 326000 $

Several display modes are available: no legend, 'top', 'bottom', 'left', 'right', 'short' and 'best'. The legend supports all the mirrored and inverted axes in best mode in order to provide the best placement on the plot. When aggregation (cumulative sum) is displayed, it also adds a key showing the label of the three columns of the plot. Namely: aggregate, stem and leaf (see top right corner of Figure 0.1).

Secondary plot

Stem_graphic uses matplotlib and integrates with it quite well. First, by accepting an instance of axes as an argument, and secondly by returning an instance of figure and axes. The next subsection will cover a few possible scenarios. In many cases however, all that is needed is a secondary plot, in the margin or as overlay. For that reason, *stem_graphic* accepts as optional arguments a keyword dictionary (secondary_kw=) for the sec-

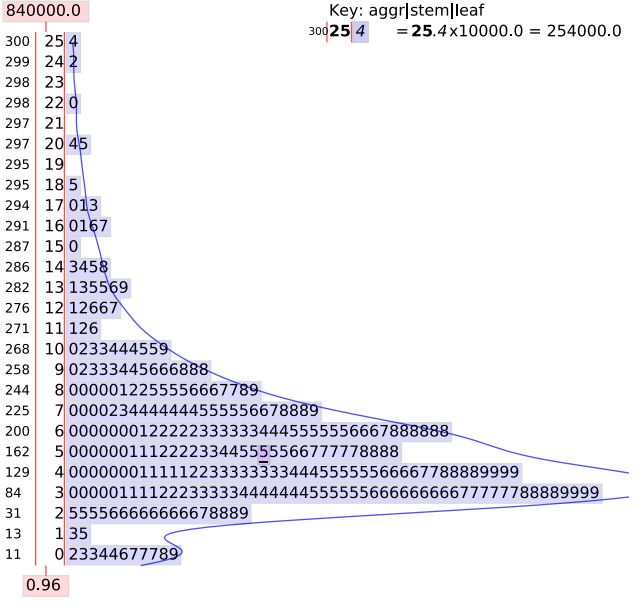


Figure 3.3: Stem-and-leaf plot with secondary plot (overlay_kde)

ondary matplotlib plot(), and a string type (*secondary_plot=* one of 'kde', 'dot' or 'rug' for margin plots, or 'overlay_kde' at the time of this writing) or any user defined function. Scale and sampling is automatic in all cases, which makes it very easy to use.

In Figure 6, a stem-and-leaf plot is limited to only 300 data points, but the density curve is calculated on the whole dataset and plotted at a higher resolution than the stem rows. For the selection of the default, a survey was sent to the whole data science department at Inmar, and to potential users outside the department. The result was fairly close, with about 22% without a response, 34% in favor of a kde overlay by default and 44% in favor of leaving the kde overlay disabled by default. Given the results, a user settings file was added to ensure the expected default behavior for all users.

Persistence

The *stem_graphic* function returns two variables, instances of matplotlib figure and axes:

```
fig, ax = stem_graphic(x)
```

Figure can be used for a quick and easy way to save the resulting plot by using *fig.savefig('file.png')*. All formats supported by matplotlib are available, including *png* for bitmap needs and *svg* for vector based applications. For print work, *svg* is a convenient way to import the plot into a tool like Inkscape for further manipulations without any resolution loss, while *pdf* can be used directly in LaTeX using *graphicx*, with no loss of transparency. It is also possible to first define a figure (with subplots) and pass *ax=subplot* to multiple *stem_graphic()* calls to build more elaborate combinations of plots. In this case, each plot would share the same figure, instead of creating it. Once more, *fig.savefig()* would save all the subplots in one graphic file.

Stem_graphic and *stem_text* also take an optional data *persistence* keyword. This is a file name with or without path. When specified the (reduced) data set will be pickled to disk as a Pandas dataframe if the *.pkl* extension is used, or a csv otherwise. This is particularly convenient to quickly get a sample to disk when dealing with large datasets.

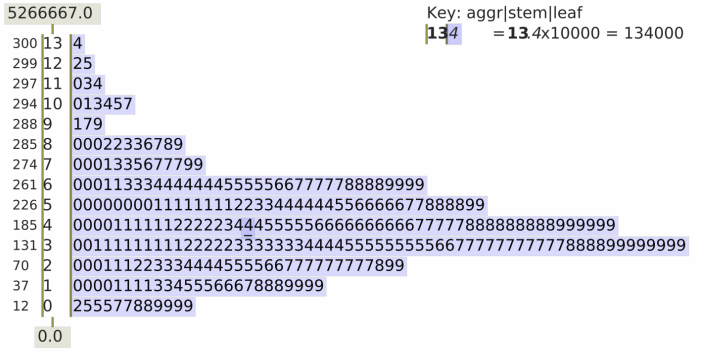


Figure 3.4: Default colors with simulation of deuteranopia

Sane defaults

Many of the defaults and handling of edge cases were selected through the elaboration of many functional tests and from user feedback. In this subsection, a selection of some of the other options, along with the justification for their default settings is presented. *Aggregation* is set to *True*, adding a vertical delimiter and a cumulative count from the first stem up. The final count is the total of data points displayed, a quick way to know the sample size when we display less than the complete dataset (Figure 0.1 shows a plot with aggregation, while Figure 3.1 shows a plot without aggregation). It is also possible to optionally specify the number of points to show with *display=*, to specify the scale (*scale=*), or to alter it from the calculated scale to get the shorter plot by adding the *zoom=-1* option or longer with *zoom=+1*.

The *alpha* (transparency) of the background fill is set to 0.15 (85% transparent), providing a good contrast with the default text color while at the same time providing enough hinting to clearly see the histogram and estimate the distribution curve. Colors default to red (for outliers and vertical bars) and blue (for leaf background), with purple (median), not fully saturated (similar to some of the default Seaborn colors). Since

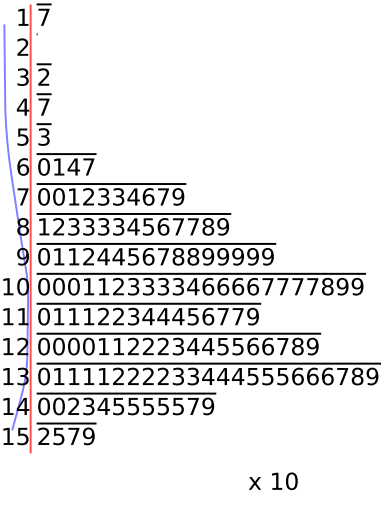


Figure 3.5: Several changes from the defaults (including marginal kde)

we use red, we avoid green to provide a color-blind (deuteranopia (Figure 3.4), protanopia and tritanopia) and grayscale friendly color set, each tested with a color filter (see [7] for color-blind friendly palettes). Text color is set to black but can be specified. Figure 0.1 shows the default fixed width font for screen and print. Figure 3.1 shows an alternate monospaced font. It is also possible to use a box outline (see cover) or an underline of the leaves instead of a background fill. Finally, various other aspects of the visuals of the plot can be changed (Figure 3.5).

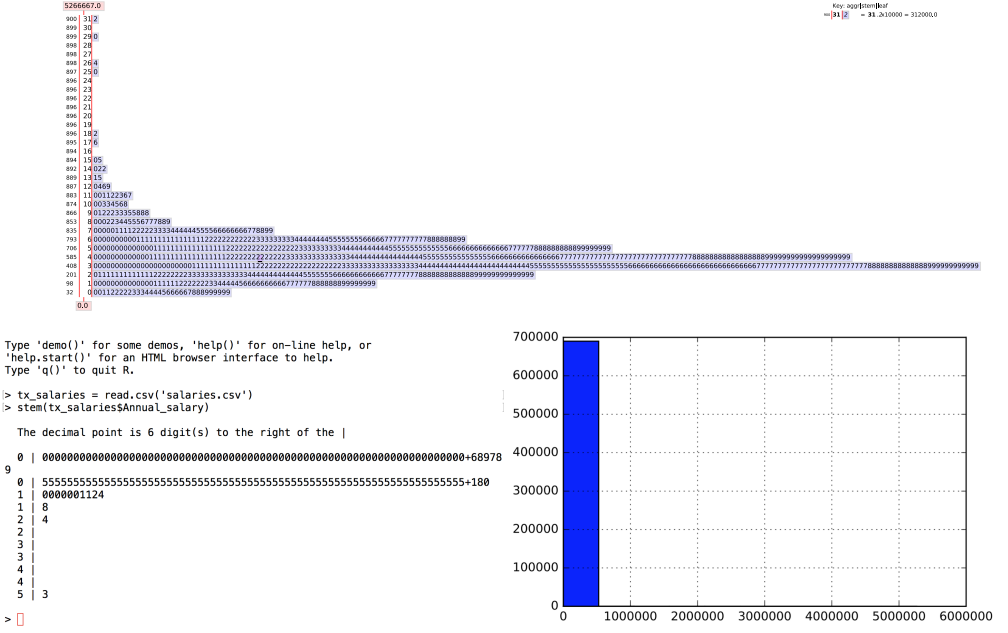


Figure 3.6: Output of stem_graphic() (above) and R's stem() and Pandas hist() (below) using default settings, from the same dataset with 700,000 data points. Stem_graphic not only provides a much better overall picture of the distribution, at normal size it also provides details not found elsewhere.

Chapter 4

Evaluation

4.1 Comparison with corpus of small datasets

To have a good variety of use cases of small datasets, functional tests were built from exercises and examples found in many of the books in the reference list at the end of this paper. In particular, all the exhibits from Hoaglin [14], non categorical examples of Tukey [34], Rosetta Code [29], Chambers *et al.* [4], Aczel [1], Harris [13] and others served to define the test sets, the expected text output and expected scale calculation. In the few cases where the output diverged from the depiction in those sources, the output was usually the right compromise between a plot that would be too tall or too wide. In Figure 3.1 the version calculated by *stem_graphic* is shown. While the original source (Aczel) discussed 3 different possible plots ranging from too short, just right and arguably too tall, *stem_graphic* provides the middle version by default. The functional tests also generate various *png* and *pdf* files for each of the tests.

Once all the tests are run, it is possible to assess the quality of the generated plot by looking at them in the output folder. This is done before each release of the software to prevent regression and to continue to improve the output. When a new set is not properly handled (per the user), it is added to the functional test suite, the code is adjusted, and so on.

4.2 Large datasets

In Figure 3.6, we compare the output of *stem_graphic* with the R *stem()* output and Pandas *hist()* on large data. We can see where our sampling implementation shines. Although this technique could be applied to any type of visual display such as histograms or existing stem and plot functions, *stem_text* and *stem_graphic* as default behaviour provide the most useful view of the actual data for our use case, including the values of the outliers. If the sampling were done externally to the visual display function, it would not know anything at all about the outliers. All 3 approaches require the data to first be loaded in memory before doing calculations and creating the output. The speed of execution was benchmarked (part of functional tests), with the dataset of Figure 7. For 10 consecutive runs, R's *read.csv()* and *stem()* execution took 19.3s on average, Pandas *read_csv()* and *df.hist()* took 1.8s and Pandas *read_csv()* and *stem_graphic*, 2.2s.

Using NYC taxi trip data [25] for 2015 (close to 150M records and 22GB of data requiring over 60GB of ram), we get into big data territory. On a laptop, *stem_graphic* is the only option (passing it a dask [6] dataframe). As a reference, loading the files individually in Pandas and adding the times we measured 8m29s, while a simple *wc* (word count) ran in 5m1s. With *stem_graphic* we measured 6m6s. Using distributed dataframes and enough worker nodes, this scales to interactive speed.

4.3 Use in EDA tasks

Once the complete module was deployed to IVANE, usage grew quickly and provided many examples and opportunities to improve the tool as was mentioned in section 3.1.2. Initially, sampled plots were used 40 to 60 times over the course of a day, with a great number of unusable plots by default. With the sampling method refinement, this simple change improved the number of usable plots without adjustments to over 94%, leaving two to three plots a day which needed optional arguments or other treatment. By comparison, Pandas *hist()* rarely provided any usable plot. A functional and qualitative test simulating repetitive EDA tasks was also added to the suite of tests.

Chapter 5

Conclusion

Stemgraphic is an open source python module that implements a fast, scalable and easy to use, highly configurable graphical stem-and-leaf plot. It is fully integrated in Inmar's IVANE EDA environment where it provides a superior alternative to histograms. Reception of the graphical stem-and-leaf plot generated by *stem_graphic* has been extremely good in communication work in team meetings as well as in printed form. In fact, one CxO's comment indicated that the output quality was such that the settings had to have been tweaked on purpose and was quite surprised to learn that he was in fact looking at *stem_graphic*'s results using only the default options. It is actually the best feature of *stem_graphic*: without any data wrangling or having to think about options, it provides a visually appealing, usable plot.

There are some limitations in the current version of *stemgraphic*. Future improvements to *stemgraphic* will address support for multivariate dataframes in single and multiple plots, a method to pack a large number of graphical stem-and-leaf plots on a single page through optimal placement, and creating an interactive drill down version of the plot, each to be released to the *stemgraphic* github repository. Future research areas include how the stem-and-leaf plot is perceived and understood for communications with the public at large, and how the combination of graphics and numbers impact the persuasive power of the plot.

Acknowledgements

I would like to thank my wife Yiyi for her support of my scientific endeavours, Jean-Pierre Dion for general guidance and help in tracking down certain publications, the Data Science team at Inmar for the brainstorming, enthusiasm and feedback, survey participants for helping to shape the product, Don Jennings, Esteban Anaya, Jaafar BenAbdallah, Jeff Clouse and Rob Agle for many constructive comments, feedback and suggestions, Edward Tufte for providing feedback to improve print quality, and Lionel Mendoza for additional insights on his research. I gratefully acknowledge the works of the late Jacques Bertin and John Tukey who, many moons ago, laid the foundation to this paper. This work was done in part at Inmar Inc.

Bibliography

[1] A. Aczel. *Complete Business Statistics*. Irwin, 1989.

[2] L. P. Ayres. *The War With Germany, a Statistical Summary*. Washington Government Printing Office, 1919.

[3] J. Bertin. *La Sémiologie Graphique*. Paris-La Haye, Mouton, Gauthier-Villars, 1973.

[4] J. M. Chambers, W. S. Cleveland, B. Kleiner and P. A. Tukey. *Graphical Methods for Data Analysis*. Chapman & Hall, 1983

[5] W. S. Cleveland. *The Elements of Graphing Data*. Wadsworth, 1985

[6] Dask Development Team. "Dask", 2016. [online]. Available: http://dask.pydata.org

[7] F. Dion. *The 10 colors of Pi*, 2015. [online]. Available: http://artchiv.es/rpy2015/10colors

[8] W. J. Dixon, R. A. Kronmal, "The Choice of Origin and Scale for Graphs", Journal of the Association for Computing Machinery 12, 259-261, 1965.

[9] S. H. C. du Toit, A. G. W. Steyn, R. H. Stumpf. *Graphical Exploratory Data Analysis*. Springer-Verlag, 1986.

[10] J. W. Dudley Jr. *Examination of Industrial Measurements*. McGraw-Hill, 1946.

[11] D. S. Dunn. *Statistics and Data Analysis for the Behavioral Sciences*. McGraw-Hill Higher Education, 2001.

[12] Florida Department of Management Services, "Search State Payroll", 2016. [online]. Available: http://floridahasarighttoknow.myflorida.com/search_state_payroll

[13] R. Harris. *Information Graphics, A Comprehensive Illustrated Reference*. Management Graphics, 1996

[14] D. Hoaglin, F. Mosteller, J. W. Tukey. *Understanding Robust and Exploratory Data Analysis*. Wiley, 1983.

[15] D. Hoaglin, S. Wasserman. ROSEPACK Document No. 2: Automating Stem-and-Leaf Displays, Working Paper No. 109. NBER, 1975

[16] Z. Hongbing, W. Yu, H. Lijuan, "The Application of Making Back-to-back Stem and Leaf Graph using Excel in Statistic", 2012 IEEE Symposium on Robotics and Applications (ISRA), pp. 769-772, 2012.

[17] J. D. Hunter. "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, 9:90-95, 2007.

[18] J. J. Kaplan, J. G. Gabrosek, P. Curtiss, C. Malone, "Investigating Student Understanding of Histograms", Journal of Statistics Education, Volume 22, Number 2, 2014.

[19] Jupyter Development Team. "Jupyter", 2016. [online]. Available: http://jupyter.org

[20] E. M. Leerkes. SPSS Manual for Howell's Fundamental Statistics for the Behavioral Sciences, 5th ed. Thomson & Brooks/Cole, 2004.

[21] D. Mackay, A.Wilks, University of Toronto, Institute for Environmental Studies and Joint Program in Transportation and Princeton University, Dept. of Statistics and Canada, Transport Canada, Research and Development Centre, "A statistical analysis of oil spills in Canada", Research report (Joint Program in Transportation). University of Toronto/York University Joint Program in Transportation, 1978.

[22] Matplotlib Development Team, "pylab_examples stem_plot.py", 2012. [online]. Available: http://matplotlib.org/examples/pylab_examples/stem_plot.html.

[23] W. McKinney, "Data Structures for Statistical Computing in Python", Proceedings of the 9th Python in Science Conference, S. van der Walt and J. Millman, eds, 2010.

[24] D. S. Moore, G. P. McCabe, B. A. Craig. *Introduction to the Practice of Statistics, 8th edition.* W. H. Freeman, 2015.

[25] NYC Taxi and Limousine Commission, "TLC Trip Record Data", 2015. [online] Available: http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml

[26] L. Pereira-Mendoza, A. Dunkels. "Stem-and-Leaf Plots in the Primary Grades", Teaching Statistics, 11: 34-37. doi: 10.1111/j. 1467-9639.1989.tb00045.x, 1989.

[27] Public Schools of North Carolina State Board of Education. Mathematics Standard Course of Study and Grade Level Competencies, K-12. Department of Public Instruction, 2003.

[28] J. A. Rice. *Mathematical Statistics and Data Analysis.* Wadsworth & Brooks/Cole, 1988.

[29] Rosetta Code Contributors, "Stem-and-leaf plot", 2016. [online]. Available: http://www.rosettacode.org/mw/index.php?title=Stem-and-leaf_plot&oldid=223644

[30] Texas Education Agency. Texas Essential Knowledge and Skills for Grade 4. http://tea.texas.gov/WorkArea/linkit.aspx? LinkIdentifier=id&ItemID=25769806958&libID=25769806961, August 2015.

[31] Société de transport de Montréal, "Brief history of public transit tickets in Montréal", 2016. [online]. Available: http://www.stm.info/en/about/discover_the_stm_its_history/history/brief-history-public-transit-tickets-montreal

[32] E. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, 1983.

[33] J. W. Tukey. *Exploratory Data Analysis, Vol I & II limited preliminary edition.* Addison-Wesley, 1970.

[34] J. W. Tukey. *Exploratory Data Analysis.* Addison-Wesley, 1977.

[35] J. W. Tukey, "Some Graphic and Semi-Graphic Displays", *Statistical Papers in Honor of George W. Snedecor*, T. A. Bancroft, ed, pp. 293-316, Ames, Iowa, 1972.

[36] J. W. Tukey. *The Collected Works of John W. Tukey, Volume V*: Graphics (W. S. Cleveland, ed.). Wadsworth, 1988.

[37] U.S. Census Bureau, "Millenials Outnumber Baby Boomers and Are Far More Diverse, Census Bureau Reports", Release Number CB15-113, June 25, 2015.

[38] M. Waskom *et al.* seaborn: v0.7.0 (January 2016). Zenodo. 10.5281/zenodo.45133, 2016.

[39] H. Wickam. *ggplot2: Elegant Graphics for Data Analysis.* Springer-Verlag New York, 2009.

[40] N. Yau. *Visualize This: The FlowingData Guide to Design, Visualization, and Statistics.* Wiley, 2011.

Companion to the PyData Carolinas 2016 talk, 1st printing, Aug. 2016.
This document was composed in LaTeX and uses the libertine font

F. Dion. *STEMGRAPHIC: A Stem-and-Leaf Plot for the Age of Big Data*, 2016. [online]. Available: http://artchiv.es/pydata2016/stemgraphic

 fdion@dionresearch.com